Digital Literacy

Competency Level: Advanced Digital Literacy (Version 1.0)

Nominal Duration: 240 Hours

Prepared By: Digital Amhara Initiative

October 11, 2025

Bahir Dar- Ethiopia

Acknowledgment

Digital Amhara was initiated by the **Amhara Regional State President** and coordinated by the **President’s Delivery Unit**. The program is generously financed by **Big Win Philanthropy**.

The **Advanced Digital Literacy Competence** is one component of the Digital Amhara Initiative. The Initiative extends its sincere appreciation to all partners who contributed to the development of this program.

Special recognition goes to the dedicated experts and teams from the **Amhara Education Bureau**, **Bahir Dar University**, **Bahir Dar Polytechnic College**, and the **Amhara Innovation and Technology Bureau**. Their expertise, commitment, and collaboration made it possible to design and implement a comprehensive program that strengthens digital competence across the region.

This accomplishment stands as a testament to genuine partnership and a shared commitment to advancing digital skills, fostering innovation, and driving educational transformation in the Amhara region and beyond.

Contents

List of Figure

List of Table

Acronym

• **2FA**:	Two-factor authentication
• **ADL**:	Advanced Digital Literacy
• **AI**:	Artificial Intelligence
• **API**:	Application Programming Interface
• **ATS**:	Applicant Tracking System
• **CPU**:	Central Processing Unit
• **CSS**:	Cascading Style Sheet
• **CV**:	Curriculum Vitae
• **DOCX**:	Microsoft Word Open XML Document
• **DOM**:	Document Object Model
• **E-commerce**:	Electronic Commerce
• **HTML**:	Hypertext Markup Language
• **HTTP**:	Hypertext Transfer Protocol
• **HTTPS**:	Hypertext Transfer Protocol Secure
• **ICT**:	Information and Communication Technology
• **IoT**:	Internet of Things
• **IP**:	Internet Protocol
• **IT**:	Information Technology
• **ML**:	Machine Learning
• N/**A**:	Not Applicable
• **NumPy**:	Numerical Python
• **OS**:	Operating System
• **OTP**:	One-Time Password
• **P2P**:	Peer-to-Peer
• **PDF**:	Portable Document Format
• **PHP**:	Hypertext Preprocessor
• **PIN**:	Personal Identification Number
• **QR**:	Quick Response
• **SEO**:	Search Engine Optimization
• **SMS**:	Short Message Service

• **SMEs**:	Small and medium enterprises
• **SQL**:	Structured Query Language
• **SSL**:	Secure Sockets Layer
• **SWOT**:	Strengths, Weaknesses, Opportunities, and Threats
• **UI**:	User Interface
• **URL**:	Uniform Resource Locator
• **USSD**:	Unstructured Supplementary Service Data
• **UX**:	User Experience
• **XAMPP:**	**X** (cross-platform), **A**pache, **M**ariaDB, **P**HP, and **P**erl.

Introduction

The **digital literacy program** is one of the components of the **Digital Amhara Initiative**. This program consists of three levels of competence: **basic, intermediate, and advanced digital literacy**.

The **Advanced Digital Literacy Competence** is the final program in this series, designed to meet **21st-century digital skills** in alignment with Ethiopia's **Digital Education Strategy and Implementation Plan (2023–2028)**, issued by the Ministry of Education, as well as the **Technical Guidelines for Commissioning Digital Literacy Training Platforms for African Governments** by Big Win Philanthropy and the **UNESCO Global Framework of Reference on Digital Literacy Skills for Indicator 4.4.2**.

This program equips learners with advanced practical skills to navigate and excel in today's digital landscape. Through these advanced pathways, learners are empowered to leverage digital tools and technologies confidently for professional development, academic success, and entrepreneurial innovation, preparing them to actively participate and lead in an increasingly technology-driven society.

By completing this program, learners will gain the skills to navigate digital technologies confidently, including digital financing, professional online presence, web publishing, problem-solving, and Python for data analysis, preparing them to participate responsibly and innovatively in the digital world.

Modules Covered in the Program

The program is structured into **five modules,** each focusing on a **advanced digital skills competence:**

1. Digital Financing
2. Digital Citizenship and Online Participation
3. Web Authoring and Publication
4. Advanced Problem-Solving
5. Python for Data Analysis

Learning Objectives

By the end of the program, learners will be able to:

- Use digital financial tools safely and effectively.
- Build a professional and responsible digital presence.
- Create and manage web content.
- Apply analytical and digital strategies to solve complex problems.
- Analyze and interpret data using Python.

Instruction

The program is delivered through a **competence-based approach**, integrating theoretical knowledge with practical, hands-on activities. Each module includes:

- Interactive lessons introducing key concepts and tools.
- Practical exercises and demonstrations to develop applied skills.
- Review questions and case studies to assess understanding.
- Individual activities that encourage collaboration and problem-solving.

Facilitators are encouraged to adapt examples to local contexts, ensuring that training is both relevant and accessible. Learners are expected to actively participate, practice regularly, and apply skills to real-life scenarios.

Learners will demonstrate proficiency in using digital financial tools safely and effectively, building a professional and responsible digital presence, creating and managing web content, applying analytical and digital strategies to solve complex problems, analyzing and interpreting data using Python.

Upon successful completion, learners will receive a **Certificate of Advanced Digital Literacy Competence**, recognizing their mastery of advanced digital skills and readiness to apply these skills in education, work, and daily life.
Note: Successful completion of the final evaluation is a prerequisite for receiving the certificate.

Module 1: Digital Financing

Introduction

This module provides an overview of the world of digital finance, emphasizing its significance in the modern economy and the opportunities it offers to businesses, individuals, and financial inclusion. You will learn the fundamental ideas, comprehend the difficulties and dangers, and learn about the platforms and tools that make safe, quick, and easy financial transactions possible.

By the end of this module, you will be equipped with the knowledge and skills to participate confidently in the digital financial ecosystem.

Session 1: Basic Concepts of Digital Financing and Its Importance

Introduction

Dear Learners, in this session, you will learn the basics of digital financing and its importance in improving access, efficiency, and security in financial transactions.

Digital finance is the term used to describe the impact of new technologies on the financial services industry. It includes a variety of products, applications, processes and business models that have transformed the traditional way of providing banking and financial services. Therefore, this session introduces you the basic concepts of digital financing, and its impact on the finance industries. It helps the learners to gain basic knowledge of digital finance and its influence on the finance industries in the digital world.

Let's start on your digital journey by asking one basic question about yourself:

Have you ever sent or received money from a friend using mobile banking? If so, how do you feel about the time saved with 24/7 services and the time saved for go to bank physically? This session helps you to answer these questions and get basic knowledge about digital financing and its importance efficiency and accessibility.

Learning objectives

At the end of this session, learners will be able to:

- Describe the basic concepts of digital financing
- List the importance of digital financing

Content outline

1.1 Basic concepts digital financing

This lesson introduces you the basic concepts of digital financing and its impact in the digital world. Digital finance is the term used to describe the facility, access, and management of financial services through digital technology, including mobile phones, the internet, and electronic platforms. Digital payments, e-wallets, mobile money transfers, online banking, and other financial services that can be carried out electronically without the use of actual cash or in-person banking are all included. It encompasses a wide range of applications, including mobile money, mobile banking, online payments, and the use of crypto-currencies. Digital finance is driving the development of new financial products and services, such as mobile money, digital wallets, mobile banking and peer-to-peer lending by replacing or supplementing traditional banking systems.

Figure 1: Digital financing

Examples of common digital finance:

1. Mobile banking: Accessing bank accounts and managing finances through mobile apps.
2. Online payments: Making and receiving payments through websites and apps.
3. Digital wallets: Storing and managing digital money and payment cards.
4. Cryptocurrencies: Using digital currencies like Bitcoin for transactions.

5. Fintech startups: Companies offering innovative financial solutions using technology.
6. Crowdfunding: Raising capital through online platforms.

1.2 Importance of Digital Financing

Numerous benefits provided by digital finance result in faster, safer, and easier access to financial services, which promotes financial inclusion, economic growth, and transaction efficiency. Here below are list of importance of digital finance:

i. Convenience and Accessibility
   - Digital financing allows individuals and businesses to send, receive, and manage money anytime and anywhere using mobile phones, computers, or other digital devices. It eliminates the need to visit banks physically, making financial services accessible 24/7.

ii. Time and Cost Efficiency
   - Transactions through digital platforms are faster and often cheaper than traditional banking methods. Users save travel costs, reduce paperwork, and avoid long queues.

iii. Financial Inclusion
   - Digital financing extends financial services to underserved or remote areas where traditional banks may not have a presence. This helps unbanked populations participate in the economy.

iv. Enhanced Security
   - Many digital finance platforms use encryption, two-factor authentication, and secure payment gateways, which reduce the risks associated with carrying cash or traditional check payments.

v. Transparency and Record-Keeping
   - Digital transactions generate automatic records and statements, making it easier for users and businesses to track spending, manage budgets, and maintain financial accountability.

vi. Support for Businesses
   - Small and medium enterprises (SMEs) benefit from digital financing through faster payments, easier access to loans or credit, and simplified payroll and supplier payments.

vii. Promotion of Cashless Economy

- Digital financing reduces reliance on cash, helping governments and economies reduce the risks and costs associated with handling physical money, including counterfeiting and theft.

viii. Innovation and Financial Services Expansion

- Digital financing encourages the development of new services such as mobile wallets, peer-to-peer lending, and digital investment platforms, making financial services more flexible and tailored to user needs.

Summary

In the digital world, digital financing plays a key role in the finance industry as well as the world economy as a whole. It uses IoT infrastructure (mainly internet, mobile, and software applications) to deliver financial services such as payments, lending, savings, and investments to make finance more inclusive, faster, and cost-effective, transforming how individuals and businesses manage money.

Review questions

1. Which of the following is a major benefit of digital finance?
   A. Limited accessibility
   B. High transaction costs
   C. Greater financial inclusion
   D. Dependence on physical banks
2. What is digital finance?
   A. Using physical currency for all transactions
   B. Accessing and managing financial services through digital technology
   C. Visiting banks to apply for financial services
   D. Exchanging goods and services without money
3. What is one key benefit of digital finance?
   A. It increases the need for paper-based transactions
   B. It limits access to financial services
   C. It offers convenience and accessibility anytime, anywhere
   D. It requires in-person banking
4. In digital financing, who typically controls the assets?
   A. Central banks

   B. Commercial banks
   C. Users themselves
   D. Governments

5. Which is NOT a digital finance tool or service?
   A. Cryptocurrencies
   B. Digital wallets
   C. Online payments
   D. Manual bookkeeping

Session 2: Challenges and Risks of Digital Financing

Introduction

Dear learners, welcome to this session where you will explore the challenges and risks of digital financing.

Although digital finance plays a key role in the finance industry to make financial services to be more accessible and inclusive, it has the issues of security and privacy. In this session you will learn more about common challenges of digital financing mainly security, privacy, literacy, trust and adoption issues in the financial services.

Let's start on your digital journey by asking few questions.

- Have you sent money to the wrong account?
- Do you have any experience that, you transfer money to your family or friend but the transaction takes a long time and upset you?
- Have you ever tried to withdraw money from an ATM, completed all the steps, but didn't receive the cash because of a power failure?

This session helps you to gain knowledge why the above events are occurred and strategies to minimize these risks and challenges that affects the growth of digital finance.

Learning objectives

At the end of this session, learners will be able to:

- Identify security and privacy challenges of digital financing
- Differentiate strategies for mitigating challenges in digital financing

Content outline

2.1 Security issues and challenges of digital financing
2.2 Mitigate strategies to challenges and risks of digital financing

2.1 Security issues and challenges of digital financing

While using digital finance tools such as online payments, mobile money, and mobile banking is convenient, there are risks involved. It's critical for you to understand the primary security issues and how to prevent them. Here below are list of risks and challenges:

- Cyber Attacks: Hackers can break into systems and steal your personal or financial information.
- Phishing Scams: You might get fake emails, texts, or calls asking for your PIN, password, or OTP.
- Weak Passwords: If you use simple or repeated passwords, criminals can easily guess them.
- Fake Apps or Websites: Scammers create apps or websites that look like the real ones to trick you.
- Public Wi-Fi Risks: Using banking apps over public Wi-Fi can expose your data to hackers.
- Device Theft or Loss: If someone steals your phone or laptop, they may access your banking apps.
- Lack of Awareness: Not knowing about digital security increases your risk.

2.2 Mitigate strategies to challenges and risks of digital financing

Mitigation strategies refer to specific actions or measures taken to reduce the likelihood of a risk occurring or minimize its negative impact if it does occur.

- Mitigation: Reducing the effect or chance of a problem.
- Strategies: Planned steps or methods to handle those issues.

In the context of digital financing challenges and risks, mitigation strategies might include:

i. Use Strong Passwords and Security Settings
    - Create strong, unique passwords for each financial app.
    - Change passwords regularly.
    - Use two-factor authentication (2FA) where possible.

ii. Avoid Using Public Wi-Fi for Transactions
    - Never access financial accounts over public or unsecured Wi-Fi.
    - Use mobile data or secure private networks for transactions.

iii. Keep Devices and Apps Updated
    - Regularly update your smartphone, banking apps, and antivirus software.
    - Updates often contain security patches that protect you from new threats.

iv. Beware of Scams and Phishing
    - Don’t click suspicious links or respond to unknown messages asking for personal information.

- Verify sender identities before taking any action.
- Banks and financial platforms never ask for PINs or passwords via SMS or email.

v. Review Transactions Regularly

- Monitor your bank or mobile money transactions frequently.
- Report any suspicious or unauthorized activity immediately.

vi. Log Out After Use

- Always log out of apps and websites after completing a financial transaction.
- Avoid saving login information on shared devices.

vii. Stay Informed and Digitally Literate

- Learn how your digital finance tools work.
- Attend trainings or read guides provided by your bank or service provider.

viii. Use Official and Trusted Platforms Only

- Only download apps from official app stores (e.g., Google Play, Apple Store).
- Use recognized and regulated digital finance providers.

ix. Enable Alerts and Notifications

- Turn on SMS or app notifications for every transaction.
- This helps you track activity and detect fraud early.

x. Secure Personal Information

- Don’t share PINs, passwords, or account numbers even with friends or family.
- Shred or delete old statements and sensitive data.

Summary

Digital financing makes sending and receiving money easier, but it comes with challenges like online scams, system errors, weak rules, lack of digital skills, and poor internet or electricity.

To stay safe, learners should use strong passwords, enable two-factor authentication, avoid public Wi-Fi, keep devices updated, check transactions regularly, use trusted apps, turn on alerts, and protect personal information.

Review Questions

1. Why is using public Wi-Fi a risk when performing digital transactions?
   A. It uses more data
   B. It slows down transactions
   C. It exposes your information to hackers

D. It blocks transaction notifications

2. What is a major risk of using weak or reused passwords for digital financial services?
   A. Account will be deactivated
   B. Password may expire faster
   C. It is easier for hackers to guess and break into accounts
   D. It will require 2FA
3. What is the best way to protect financial accounts from unauthorized access?
   A. Share passwords with family
   B. Use the same password everywhere
   C. Use strong passwords and enable 2FA
   D. Avoid installing antivirus software
4. Which of the following is a safe practice when accessing mobile banking apps?
   A. Use them over public Wi-Fi
   B. Keep the app always logged in
   C. Use secure, private networks
   D. Share OTP with friends
5. What should you do immediately after completing a digital financial transaction on a shared device?
   A. Turn off the device
   B. Switch to a different app
   C. Log out from the financial app
   D. Close the browser tab only

Case Study: Digital Finance Security

Mr. Tesfaye is a small business owner who uses mobile banking and online payment platforms to pay suppliers and receive customer payments. One day, he received a fake email claiming to be from his bank, asking for his account details. He almost entered the information but remembered guidance from a recent digital finance workshop. He also sometimes uses public Wi-Fi in cafes for business transactions and rarely updates his passwords.

Questions:

1. What security risks did Mr. Tesfaye face?
2. Which mitigation strategies should he implement?
3. How can he further strengthen his digital finance security?

Session 3: Digital Financing Services and Platforms

Introduction

Dear learners in this session you will explore the digital financing services and platforms.

Now a day's there are different types of platforms and technologies available to deliver financial services digitally. Each category covers a distinct way how people can store, send, borrow, invest, or manage money online. In this session you will learn more about common digital financing platforms.

Let's start on your digital journey of digital financial platforms by asking few questions.

- Have you ever completed a payment or transfer without visiting a bank? Which tool did you use?
- How do digital finance tools save you time or money in your daily life or business?

This session helps you to know the available digital finance services and platforms and the way how to register to these services and using them.

Learning Objectives

At the end of this session, learners will be able to:

- Identify different digital finance services and platforms.
- Explain how these services and platforms operate.
- Demonstrate basic use of selected digital finance platforms.

Content outline

3.1.Digital financing services and platforms
3.2.USSD Payment System in Ethiopian Banking

3.1 Digital financing services and platforms

Digital financing platforms

Digital financing platforms are online or mobile-based systems that facilitate financial transactions and services. Features: Send/receive money instantly, pay bills and buy goods/services and top-up airtime or transfer funds.

Digital Financing Platforms in Ethiopia are listed below:

Telebirr:

- **Provider**: Ethio Telecom,
- **Access**: Available via USSD (*127#*), mobile app (Android/iOS), and SMS

CBE Birr:

- **Provider**: Commercial Bank of Ethiopia
- **Access**: Available via USSD, mobile app, and agent network

Apollo:

- **Provider**: Abyssinia Bank
- **Access**: Available via USSD and mobile app.

Dashen bank superapp:

- **Provider**: Dashen Bank
- **Access**: Available via mobile app

Awash Birr:

- **Provider:** Awash Bank
- **Access**: Available via mobile app and USSD

Efoyta:

- **Provider:** Wegagen Bank
- **Access**: Available via mobile app and USSD

Abol:

- **Provider:** Buna Bank
- **Access**: Available via mobile app and USSD

Aba:

- **Provider:** Amhara Bank
- **Access**: Available via mobile app and USSD

NIBtera:

- **Provider:** Nib International Bank
- **Access**: Available via mobile app and USSD

HIBIR:

- **Provider:** Hibret Bank
- **Access**: Available via mobile app and USSD

Ahadu Mobile banking:

- **Provider:** Ahadu Bank
- **Access**: Available via mobile app and USSD

HalalPay:

- **Provider:** Hijira Bank
- **Access**: Available via mobile app and USSD

M-Pesa Ethiopia: the provider is Safaricom Ethiopia and accessed by mobile app.

OrooDigital:

- **Provider**: Oromia Bank
- **Access**: Available through the bank's mobile application and USSD.

EthSwitch:

- **Role**: National payment switch

Digital financing services

Digital financing services encompass the broader range of financial products and services offered through digital channels. By using digital platforms and technologies, digital financing services enable users to manage, transfer, and access funds without exclusively depending on traditional banking methods. They are intended to improve accessibility, speed, and convenience of financial transactions at any time and from any location. Common digital financing services are listed below:

- **Mobile Banking** allows users to access their bank accounts via mobile devices for checking balances, transferring money, and paying bills. To use it, you can download your bank's mobile app or use USSD codes. Registration usually requires your account number and linked phone number.

- **Mobile Money / E-wallets** provide services to send and receive money, pay merchants, and save digitally. Examples in Ethiopia include M-Birr, HelloCash, and Amole. You can access these services by visiting an agent or downloading the mobile money app, then registering with your phone number and personal identification.
- **Online Banking** is internet-based banking that allows you to manage accounts, transfer funds, and pay bills. To use online banking, register on your bank's website and log in using your credentials. Some services may require an OTP (one-time password) for added security.
- **Digital Payments** enable paying for goods and services electronically without cash, using methods such as QR codes, POS, or online payment gateways. You can make payments through mobile banking apps, mobile money, or merchant-provided QR codes, linking your bank account or e-wallet to pay.
- **Digital Lending/Microfinance services** provide short-term loans via apps or digital platforms. To access these services, download the lender's app, register with your ID and phone number, and apply for a loan. Approval and repayments are handled digitally.
- **Digital Savings and Investment** Platforms allow you to save money or invest in stocks, bonds, or mutual funds online. Access is typically through a bank or fintech app, where you can open an account and deposit funds via mobile banking or e-wallet.
- **Business Payment Platforms** help merchants receive payments, track transactions, and manage finances digitally. To use these, register your business on a bank or fintech platform and link a merchant account or e-wallet for payments.
- **Crowdfunding and Digital Investment Platforms** are online platforms that enable individuals or businesses to raise funds or invest through digital channels. These platforms allow users to raise money for projects or startups, invest in businesses or initiatives digitally, and track and manage their investments conveniently online.

Here's an overview linking the digital financing platforms to their corresponding types of financial services:

Table 1: Digital financing platform and services

Platform	Provider	Access	Services Provided
Telebirr	Ethio Telecom	USSD (127#), Mobile App (Android/iOS), SMS	Mobile Money / E-wallet, Digital Payments, P2P Transfers
CBE Birr	Commercial Bank of Ethiopia	USSD, Mobile App, Agent Network	Mobile Banking, Mobile Money / E-wallet, Digital Payments, P2P Transfers
Apollo	Abyssinia Bank	USSD, Mobile App	Mobile Banking, Mobile Money / E-wallet, Digital Payments
Dashen Bank SuperApp	Dashen Bank	Mobile App	Mobile Banking, Mobile Money / E-wallet, Digital Payments, Digital Savings & Investment
Awash Birr	Awash Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
Efoyta	Wegagen Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
Abol	Buna Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
Aba	Amhara Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
NIBtera	Nib International Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
HIBIR	Hibret Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
Ahadu Mobile Banking	Ahadu Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments

HalalPay	Hijira Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
M-Pesa Ethiopia	Safaricom Ethiopia	Mobile App	Mobile Money / E-wallet, Digital Payments, P2P Transfers
OrooDigital	Oromia Bank	Mobile App, USSD	Mobile Banking, Mobile Money / E-wallet, Digital Payments
EthSwitch	National Payment Switch	N/A	Digital Payments Infrastructure, Interbank Transfers

Here is how you can access the digital financial services provided by the listed platforms:

i. Mobile Banking
   - What it offers: Checking balances, transferring funds, paying bills, managing accounts.
   - How to access:
     - Download the bank's mobile app (e.g., Dashen Bank SuperApp, Ahadu Mobile Banking, NIBtera).
     - Register using your bank account number and personal identification (e.g., ID or passport).
     - Login and follow the app instructions to perform transactions.
     - Alternative: Use USSD codes if your bank supports it (e.g., Apollo, Aba, Awash Birr).

ii. Mobile Money / E-wallets
   - What it offers: Sending/receiving money, paying merchants, saving digitally.
   - How to access:
     - Download the app (e.g., Telebirr, M-Pesa Ethiopia, CBE Birr, and Awash Birr).
     - Register with your phone number and personal ID.
     - Visit an agent if needed to deposit cash into your e-wallet.
     - Use the app or USSD code to send money, pay bills, or shop.

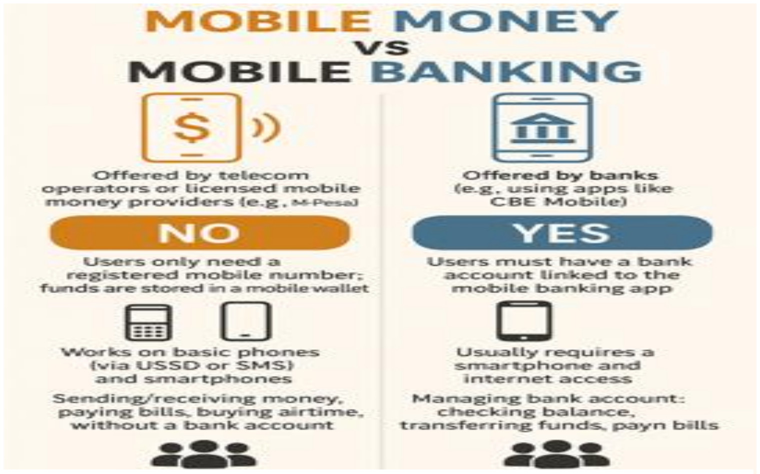
iii. Digital Payments
   - What it offers: Pay for goods and services without cash using QR codes, POS, or online platforms.
   - How to access:

        - Link your bank account or e-wallet to the payment platform.
        - Scan a merchant’s QR code or enter their payment details via the app.
        - Confirm the payment through the app or OTP.

iv. Digital Lending/Microfinance
    - What it offers: Apply for loans and make repayments digitally.
    - How to access:
        - Download the lender’s app (some banks integrate this into their main app, e.g., Dashen SuperApp).
        - Register with your ID and phone number.
        - Submit a loan application via the app.
        - Repay digitally through mobile banking or e-wallet.

v. Digital Savings & Investment
    - What it offers: Save money or invest in stocks, bonds, or mutual funds online.
    - How to access:
        - Open an account through a bank or fintech app.
        - Deposit funds digitally using your e-wallet or mobile banking.
        - Track and manage your savings/investments directly in the app.

vi. Business Payment Platforms
    - What it offers: Receive payments, track transactions, and manage finances digitally.
    - How to access:
        - Register your business on the bank’s platform (e.g., CBE Birr, Telebirr).
        - Link a merchant account or e-wallet for receiving payments.
        - Use the app or dashboard to monitor transactions and manage funds.



MOBILE MONEY vs MOBILE BANKING

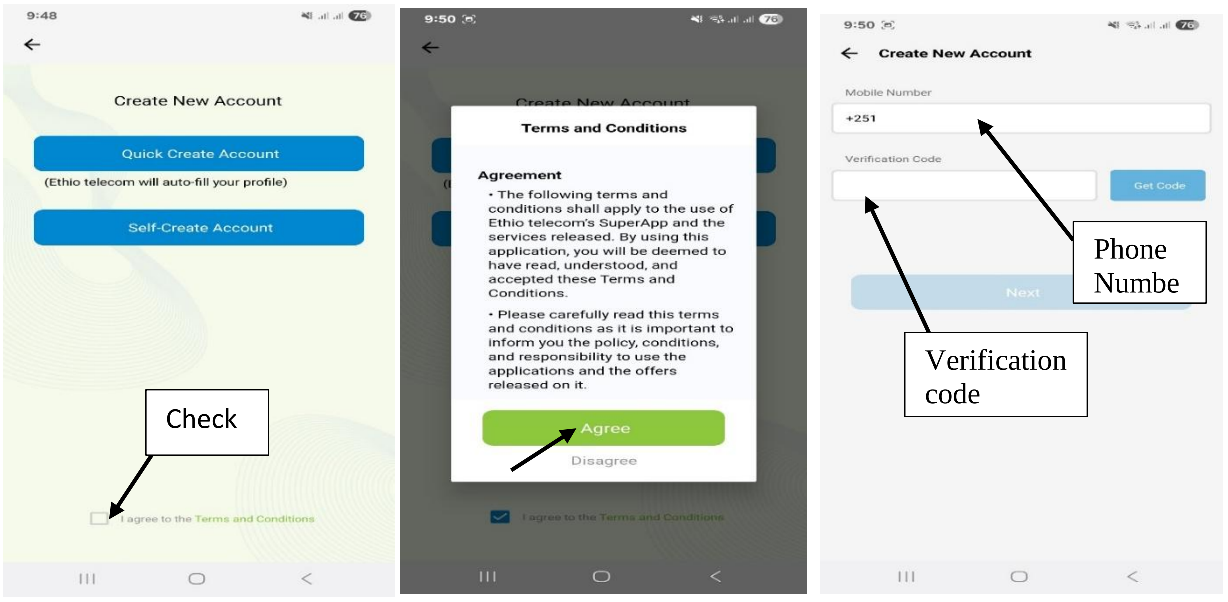
MOBILE MONEY	MOBILE BANKING
Offered by telecom operators or licensed mobile money providers (e.g., M-Pesa)	Offered by banks (e.g., using apps like CBE Mobile)
NO	YES
Users only need a registered mobile number; funds are stored in a mobile wallet	Users must have a bank account linked to the mobile banking app
Works on basic phones (via USSD or SMS) and smartphones	Usually requires a smartphone and internet access
Sending/receiving money, paying bills, buying airtime, without a bank account	Managing bank account: checking balance, transferring funds, payn bills

Figure 2: Mobile money Vs mobile Banking

Step-by-step of the Telebirr process for a customer in Ethiopia:

1. Registration

- **Method:** Telebirr mobile app (Android/iOS).
- **Requirements:** Ethiopian mobile number, personal ID (for verification), and a minimum deposit if required.
- **Steps:**
  1. Download the Telebirr app.
  2. Choose **Register**.
  3. Enter your personal details and set a **PIN**.
  4. Receive confirmation of account activation.



Agreement

• The following terms and conditions shall apply to the use of Ethio telecom's SuperApp and the services released. By using this application, you will be deemed to have read, understood, and accepted these Terms and Conditions.

• Please carefully read this terms and conditions as it is important to inform you the policy, conditions, and responsibility to use the applications and the offers released on it.

Figure 3: Telebirr registration

2. Adding Funds (Deposit/Top-up)

- **Methods:** Bank transfer, cash deposit at authorized agents, or airtime conversion.
- **Steps:**
    1. Select **Cash In** on the app menu.
    2. Choose the deposit cash then from agent (via voucher or Scan QR)
    3. Enter the amount and confirm.
    4. Funds are credited instantly to your Telebirr wallet.

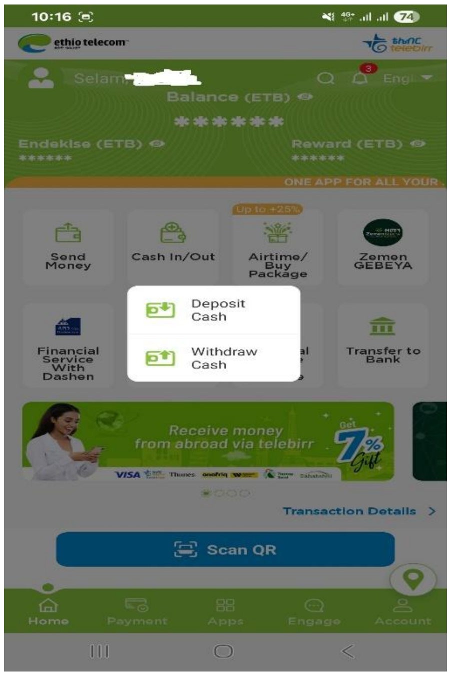


Figure 4: Telebirr cash in/deposit

Or you can send from your bank account to your telebirr wallet.

3. Sending Money / Transfers

- **Steps:**
    1. Select **Send Money**.
    2. Enter the recipient's mobile number or Telebirr ID then select next.
    3. Enter the amount and your PIN to confirm.
    4. The recipient receives the money instantly.

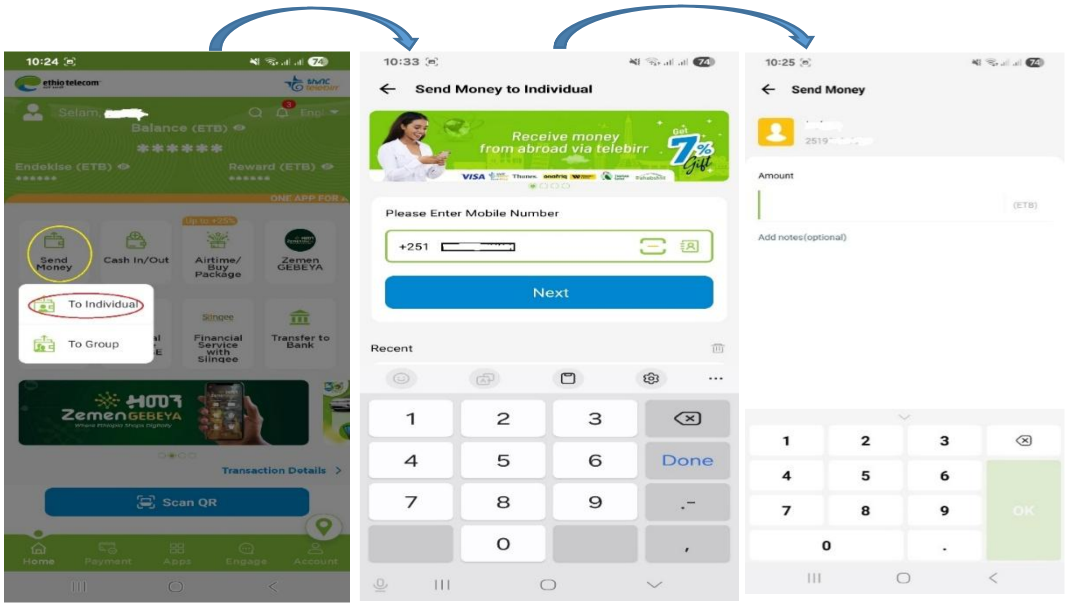


Figure 5: Telebirr transfer

4. Payments & Purchases

- **Available services:** Utility bills, school fees, online shopping, airtime top-up.
- **Steps:**
    1. Choose **Payments**.
    2. Select the service provider.
    3. Enter the account number or reference.
    4. Confirm the payment with your PIN.

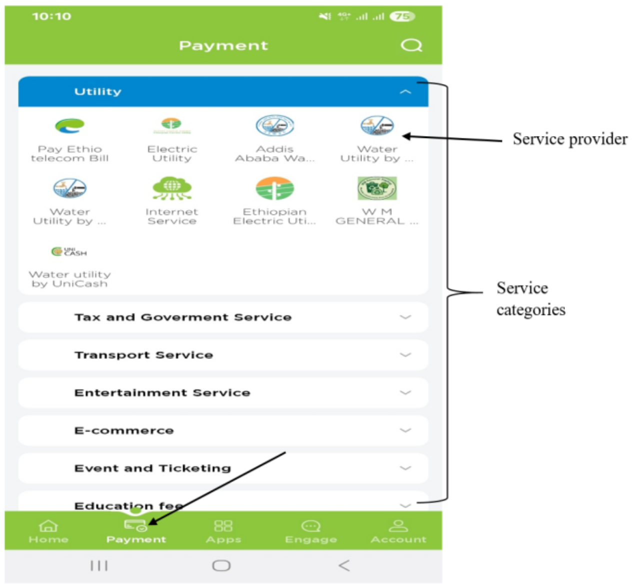


Figure 6: Telebirr payment for service providers

3.2 USSD Payment System in Ethiopian Banking

The USSD (Unstructured Supplementary Service Data) payment system has become one of the most vital financial technologies in Ethiopia's banking and digital finance landscape. It enables customers to access banking and payment services directly from their mobile phones without requiring internet access or smartphones.

Ethiopian banks such as Commercial Bank of Ethiopia (CBE), Dashen Bank, Awash Bank, Abyssinia Bank, Hibret Bank, Nib International Bank, and others use USSD codes to offer services that range from checking balances to transferring funds and paying bills.

For example:

- CBE: *889#
- Awash Bank: *901#
- Dashen Bank: *847# (via Amole platform)
- Abyssinia Bank: *815#
- Nib Bank: *804#

These codes have become a crucial bridge to financial inclusion, connecting millions of Ethiopians, especially in rural areas to formal banking services.

How USSD Technology Works for Payments in Ethiopian Banks

1. Dialing a USSD Code: The user dials a short code provided by their bank (e.g., *889# for CBE) on any GSM phone.
2. Menu Navigation: A text-based menu appears, offering options like:
   - Balance inquiry
   - Money transfer
   - Airtime top-up
   - Utility bill payment
   - Mini statements
3. Data Input: The user selects an option and enters the required information such as recipient account number, amount, or reference number.
4. Authentication: To complete the transaction, the system prompts for a secure PIN code to verify the customer's identity.

5. Processing and Confirmation: The bank's USSD gateway communicates with the core banking system to process the transaction in real-time. A confirmation message is displayed on-screen once it's successful.

Because all communication occurs through the mobile network's signaling channel (not over the internet), USSD payments are fast, lightweight, and accessible across all regions of Ethiopia.

Unique Advantages of USSD Payments in the Ethiopian Context

- No Internet Required
- Works on All Phones
- Financial Inclusion
- Real-Time Transactions
- Affordable and Accessible

Unique Challenges of USSD Payments in Ethiopia

- Session-Based Operation
- Slow Speed and Step-by-Step Menus
- Limited Data Transmission
- Network Dependency
- Security Risks

Common Transaction Errors in Mobile and USSD Banking and Their Resolving Mechanisms

While mobile and USSD banking have greatly simplified financial services in Ethiopia, users often experience transaction issues due to network interruptions, system delays, or user mistakes. Understanding these common errors and the ways banks resolve them helps ensure a smoother and safer banking experience.

Common Transaction Errors

- Network Timeout or Session Expiry
- Duplicate or Delayed Transactions
- Wrong Account or Mobile Number Entry
- Insufficient Balance or Limit Exceeded

- Service Unavailable or System Downtime
- PIN or Authentication Failure

Common Resolution Mechanisms Used by Ethiopian Banks

- **Network Timeout or Session Expiry:** Check internet connection, retry, and avoid long inactivity.
- **Duplicate or Delayed Transactions:** Wait for confirmation before retrying; check history; contact support for refunds.
- **Wrong Account or Mobile Number Entry:** Double-check details before confirming; report errors immediately for reversal to the bank.
- **Insufficient Balance or Limit Exceeded:** Verify balance and limits; add funds or adjust limits if needed.
- **Service Unavailable or System Downtime:** Wait and retry later; check provider updates or maintenance notices.
- **PIN or Authentication Failure:** Enter correct PIN/password; reset credentials if forgotten; contact support if locked.

Best Practices for Users

- Double-check all input details before confirming a transaction.
- Wait for confirmation messages before retrying.
- Keep transaction reference numbers for follow-up.
- Ensure stable network connection before initiating payments.
- Report any unresolved issues immediately to the bank's helpline.

Exercise

Using a telebirr application, answer the following questions

1. Check telebirr Account Balance: Show how to view current balance via the app or USSD.
2. Send Money (P2P Transfer**)**: Demonstrate how to send money to a friend's number (can be a simulation).
3. Buy Mobile Airtime: Show how to top-up mobile credit for yourself or another number.
4. Pay a Utility Bill (e.g., water or electricity**)**: Select a utility provider and simulate a payment.
5. Show how to transfer to telebirr to bank

Summary

This session introduces key digital financing platforms in Ethiopia, such as Telebirr, CBE Birr, Dashen Bank SuperApp, and M-Pesa, which provide financial services through mobile apps, USSD, and agent networks. These platforms enable users to perform tasks like sending and receiving money, paying bills, topping up airtime, and making purchases.

It also covers digital financing services including mobile banking, mobile money/e-wallets, digital payments, online banking, digital lending, and investment tools. These services enhance financial access, speed, and convenience for individuals and businesses, supporting transactions anytime, anywhere.

Review questions

1. What is the main advantage of digital financing platforms?
    A. Require visiting a bank for all transactions
    B. Only work with internet connection
    C. Enable access to financial services anytime, anywhere
    D. Limited to urban areas only
2. What is required to access mobile banking services?
    A. A smartphone only
    B. Internet connection only
    C. A bank account and registration on the app or USSD
    D. Visiting a branch every time
3. What is the main function of digital lending platforms?
    A. Selling electronics online
    B. Paying utility bills
    C. Providing short-term loans digitally
    D. Offering foreign exchange services

Case Study:

Sara, owns a small clothing shop in Bahir Dar. She recently started using Telebirr and M-Pesa to receive payments from customers. She also uses her Awash Birr mobile app to check her balance and transfer money to suppliers. To grow her business, she plans to apply for a digital loan using her bank's mobile app.

4. What service is she using to transfer money and check her balance?
    A. Digital Lending
    B. Mobile Banking
    C. Crowdfunding
    D. Digital Payments
5. When Sara applies for a loan through her bank app, what service is this?
    A. Mobile Money
    B. Digital Lending
    C. Business Payments
    D. Digital Investment

Scenario-Based Questions: Choosing the Right Financial System

Ethiopia has many types of mobile financial systems. People in different areas use different systems depending on their phone type, internet access, and daily needs.

1. Alemu is a farmer in a rural kebele. He owns a basic phone without internet and wants to send money to his son.
2. Mekdes is a teacher in a small town. She has a simple smartphone and uses mobile data sometimes to pay bills and buy airtime.
3. Samuel is a university student in Addis Ababa. He uses mobile apps for quick transfers and online payments.
4. Wubit is a trader who travels between Wollega and Addis Ababa. She uses both a basic phone and a smartphone, depending on where she is.

Questions (Answer based on the scenario)

1. Which person should use a USSD-based banking system (like *889# or *901#)? Why?
2. Which person would benefit most from a digital wallet such as Telebirr or Amole?
3. Who is best suited to use a mobile banking app? Give one reason.
4. Who should use a hybrid system (App + USSD)? Explain why this system fits them.
5. How does internet access affect the choice of financial system for each person?
6. In your opinion, which system is the most inclusive for rural communities? Explain briefly.

Module 2: Digital Citizenship and Online Participation

Introduction

In today’s interconnected world, building a professional online identity is essential for anyone seeking career growth or business opportunities. Your digital presence on platforms like LinkedIn, personal websites, or portfolios acts as your online resume and reputation. This module will guide you on how to create and manage a professional online image that showcases your skills, experiences, and personal brand effectively.

In addition, the module explores the growing landscape of digital job platforms and opportunities for online income. You will learn how to navigate job search websites, apply for remote and freelance work, and understand the basics of digital entrepreneurship. From freelancing and content creation to e-commerce and online services, this module introduces practical tools and strategies to help you earn and grow in the digital economy.

By the end of this module, you'll be empowered to take control of your digital reputation, navigate job markets confidently, and unlock new income-generating possibilities in the digital world.

Session 1: Building a Professional Online Identity

Introduction

Dear learners, this session will help you understand how to craft and maintain a professional online identity.

In today’s competitive digital world, your online identity is often your first impression. Employers, clients, and collaborators search for you online before they meet you in person. A strong professional digital presence not only highlights your skills but also establishes your credibility, reputation, and trustworthiness.

This session helps you move beyond casual online use (e.g., social media posting) into deliberate, professional self-presentation. You will learn how to design a personal portfolio that showcases your strengths, experiences, and aspirations. Finally, you’ll connect your portfolio to professional networks such as LinkedIn or cloud platforms (Google Drive) to ensure accessibility and visibility.

Let’s start on your digital journey by asking a few questions about yourself.

- Have you Create CV or Resume to describe yourself digitally?
- Do you have any experience that you apply jobs online?

This session helps you to gain knowledge how to create CV or resume and portfolio to describe yourself online and create job opportunities in the digital world. Think of this as building your "digital handshake" a portfolio that speaks for you even when you're not in the room.

Learning Objectives

By the end of this session, learners will be able to:

- Explain the importance of professional digital identity.
- Create a personal portfolio using Canva.
- Integrate the portfolio with LinkedIn or Google Drive.

Content outline

1.1 Concepts of Digital Portfolio
1.2 Building Digital Portfolio
1.3 Creating CV and Resume

1.1 Basic Concepts of Digital Portfolio

A digital portfolio is an online collection of your work, achievements, and professional identity. Think of it as a modern resume with visuals and interactivity a space where you not only list your skills but also *show* them through examples, projects, and creative presentation.

Unlike a printed CV, a digital portfolio is dynamic, easy to update, and accessible worldwide. Employers, clients, or collaborators can view it anytime, making it one of the most powerful tools for professional visibility.

Why a Digital Portfolio Matters?

A digital portfolio is valuable across professions:

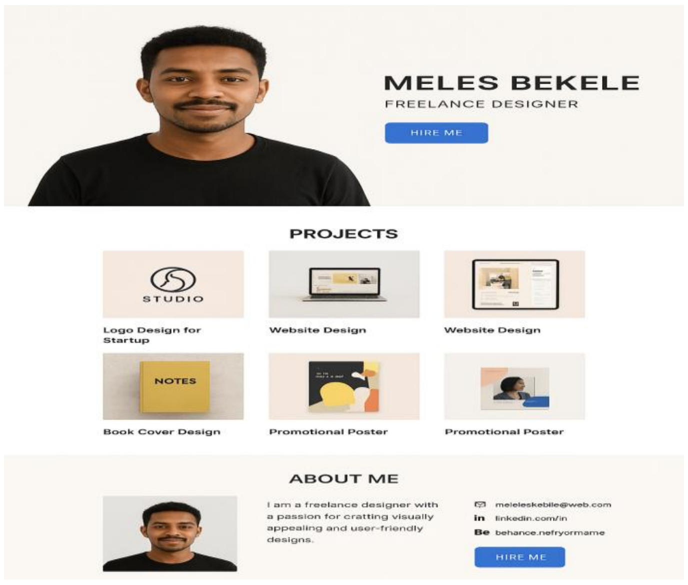
- *For Freelancers:*
  - Showcases projects and services to attract new clients.
  - Builds credibility by demonstrating real results and testimonials.

    - Makes you stand out in competitive platforms like Upwork or Fiverr.
- *For Teachers & Educators:*
    - Provides a space to highlight lesson plans, digital resources, and teaching philosophy.
    - Demonstrates commitment to professional growth.
    - Helps with applications for new teaching positions or promotions.
- *For Job Seekers (Students/Professionals):*
    - Strengthens applications beyond a resume.
    - Shows employers evidence of skills (e.g., projects, presentations, designs).
    - Helps recruiters form a strong first impression when they "Google" your name.

Examples of Good Digital Portfolios

When evaluating impressive portfolios, pay attention to clear presentation, thoughtful design, and a strong emphasis on skills.

- Example 1: Freelance Designer (Canva template):
    - Clear and visually appealing design.
    - Highlights creative projects with images and links.
    - Clear "Hire Me" button.



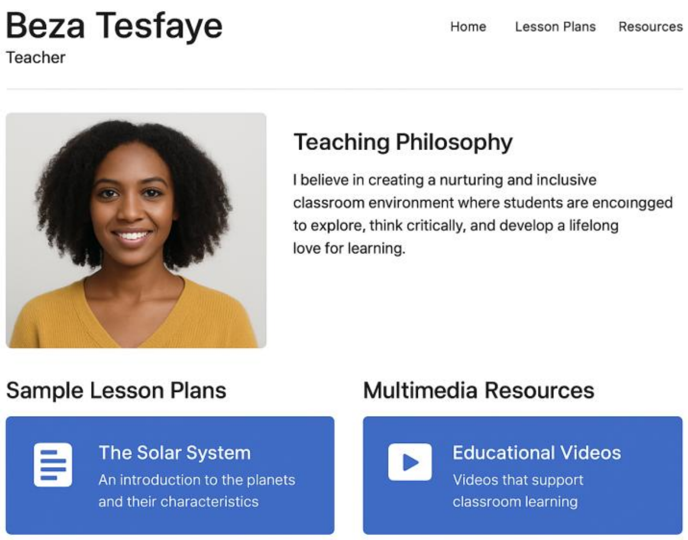
I am a freelance designer with a passion for cratting visually appealing and user-friendly designs.

meleleskebile@web.com
in linkedin.com/in
Be behance.nefryormame

Figure 7: Canva portfolio

- Example 2: Teacher Portfolio (Google Sites):
    - Includes teaching philosophy, sample lesson plans, and multimedia resources.

- Uses an easy-to-navigate menu.



Beza Tesfaye

Teacher

Teaching Philosophy

I believe in creating a nurturing and inclusive classroom environment where students are encoıngged to explore, think critically, and develop a lifelong love for learning.

Sample Lesson Plans

The Solar System

An introduction to the planets and their characteristics

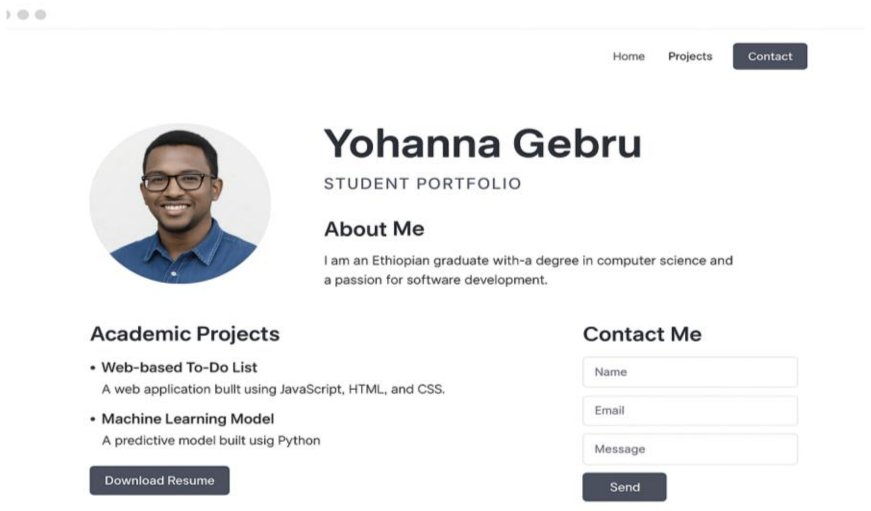
Multimedia Resources

Educational Videos

Videos that support classroom learning

Figure 8: Google Sites portfolio

- Example 3: Student/Graduate Portfolio (Wix):
  - Bio + academic projects.
  - Resume downloadable as PDF.
  - Contact form for networking opportunities.



Yohanna Gebru

STUDENT PORTFOLIO

About Me

I am an Ethiopian graduate with-a degree in computer science and a passion for software development.

Academic Projects

- Web-based To-Do List
  A web application built using JavaScript, HTML, and CSS.
- Machine Learning Model
  A predictive model built usig Python

Contact Me

Figure 9: Student's Wix portfolio

1.2 Build Digital Portfolio

You don’t need to be a web developer to build a great portfolio. There are several user-friendly tools that used to build a digital portfolio, here are the common widely used tools:

- Canva: Drag-and-drop design tool, perfect for creating PDF portfolios or mini websites.
- Wix: Website builder with customizable templates, ideal for those who want a personal website.
- Google Sites*:* Free, simple, and integrates with Google Drive for easy updates.

Each platform has the following advantages:

- Canva → Best for visual storytelling, quick setup. Address is https://www.canva.com/
- Wix → Best for a long-term professional website. Address is https://www.wix.com/
- Google Sites → Best for simple, free, and collaborative portfolios. Address is https://sites.google.com/

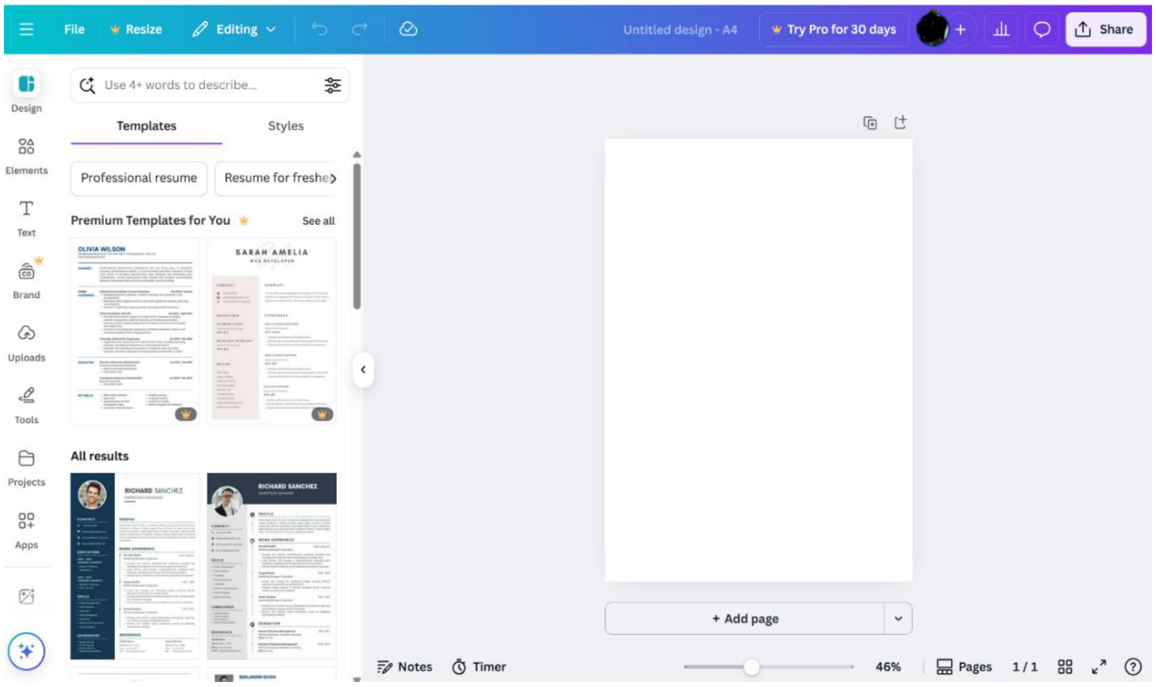


Figure 10: Canva window for resume preparation

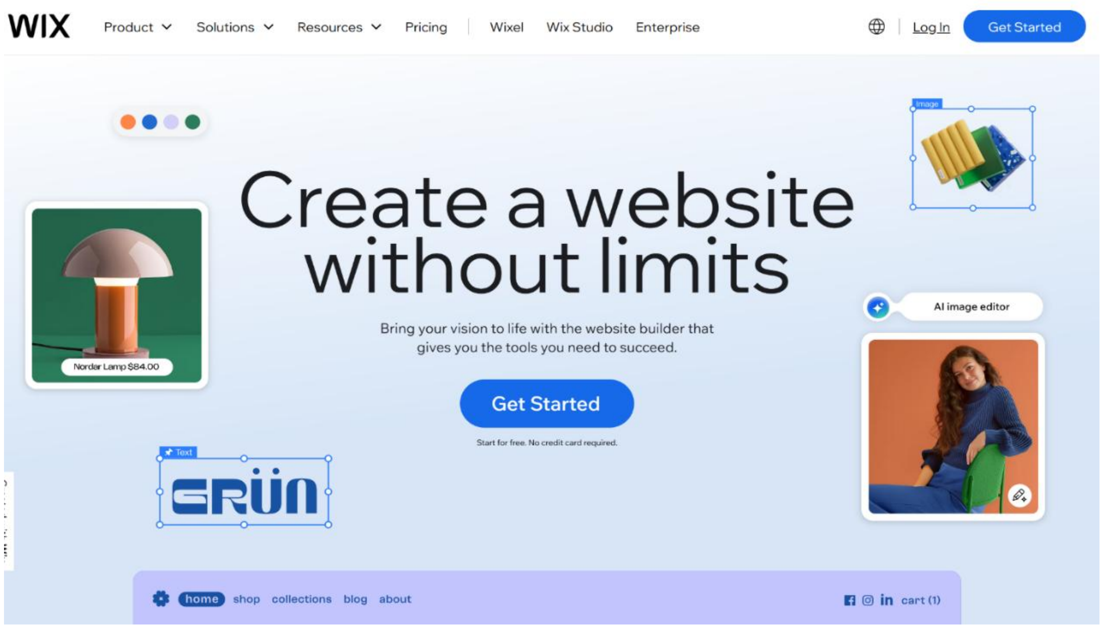

WIX
Product
Solutions
Resources
Pricing
Wixel
Wix Studio
Enterprise
Log In
Get Started
Create a website without limits
Bring your vision to life with the website builder that gives you the tools you need to succeed.
Get Started
Start for free. No credit card required.
Nordar Lamp $84.00
AI image editor
GRÜN
home shop collections blog about
cart (1)

Figure 11: Wix interface

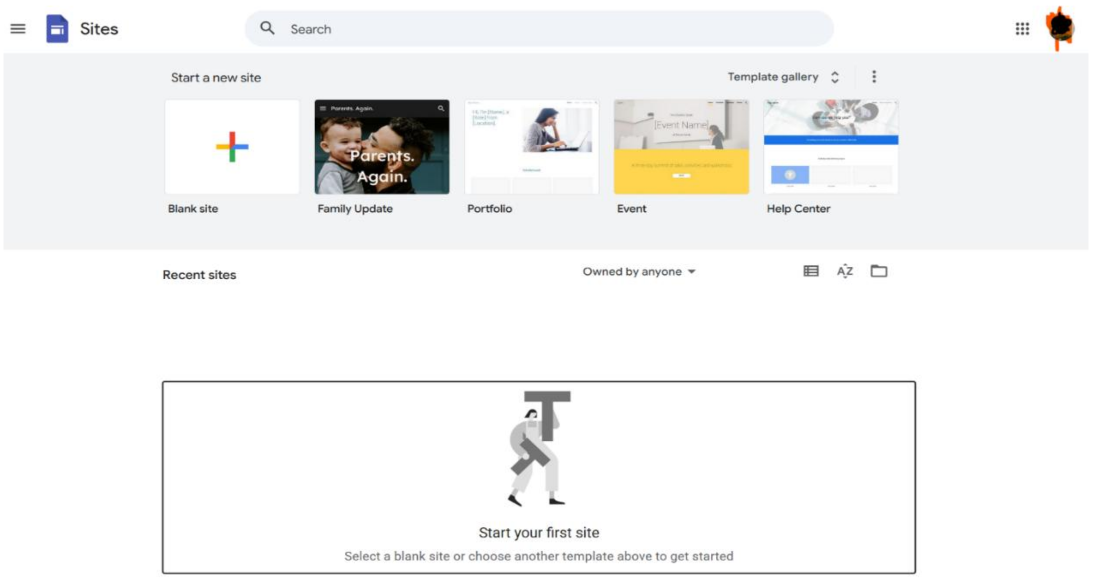


Figure 12: Google sites

Key components to include in your portfolio

A strong portfolio balances who you are, what you can do, and how people can reach you. Regardless of the platform you choose (Canva, Wix, Google Sites), these four sections are essential.

i. About Me (Bio)

This is the first impression. Keep it professional, concise, and personal enough to show your unique identity.

What to include:

- A short introduction (2–3 sentences).
- Your background (education, experience).
- Your passions and career goals.

Example:

“Hi, I’m Beza Tesfaye, an instructional designer with a passion for inclusive digital learning. I specialize in creating accessible online courses that empower diverse learners. My goal is to leverage technology and education to expand opportunities for students in Ethiopia and beyond.”

ii. Skills / Services

This section should highlight what you *can do*. Keep it clear and easy to scan with icons or bullet points.

What to include:

- Technical skills (e.g., coding, design, data analysis).
- Creative skills (e.g., graphic design, writing, video editing).
- Soft skills (e.g., communication, leadership, problem-solving).

iii. Work Samples / Projects

This is the heart of your portfolio which show evidence of your skills in action.

What to include:

- Project descriptions (lesson plans, designs, apps, websites, and reports).
- Screenshots, links, or downloadable files.
- Outcomes or impact of your work.

iv. Contact Information

Make it easy for people to reach out to you for opportunities.

What to include:

- Professional email address (*avoid casual emails like cuteboy123@gmail.com*).
- Links to LinkedIn, GitHub, Behance, or other professional platforms.
- A contact form (if using Wix/Google Sites).

Example:

- Email: Beza.Tesfaye@protonmail.com
- LinkedIn: linkedin.com/in/Beza-Tesfaye
- GitHub: github.com/Beza

Your portfolio should tell your story (About Me), show your abilities (Skills), prove your impact (Work Samples), and open the door to opportunities (Contact Info). Together, these sections create a professional, trustworthy online presence.

Exercise:

Building your Digital Portfolio

A. Step-by-Step Instructions

1. Sign in to Canva

- Open your web browser and go to www.canva.com.
- Log in using your existing Canva account, or create a free account if you don't have one.
- Tip: A free Canva account is sufficient to access portfolio templates and basic features.

2. Search for Portfolio Templates

- In the Canva search bar, type "Portfolio".
- Browse through the results and select a template that aligns with your personal style and professional tone.
- Consider the following when choosing a template:
  - Layout clarity
  - Visual appeal
  - Professional color schemes and fonts

3. Customize Your Portfolio

A. Name + Short Bio

- Add your full name and professional title.
  - Example: *Beza Tesfaye – Instructional Designer*

- Write a concise introduction (2 - 3 sentences) highlighting your background, skills, and career goals.
    - Example: *"I am an instructional designer passionate about creating inclusive and accessible online learning experiences. I specialize in developing digital modules that enhance student engagement and learning outcomes."*

B. Skills Section

- Include at least 3 key skills.
- Use icons or bullet points to make the section visually appealing.
    - Example:
        - Graphic Design (Canva, Adobe Photoshop)
        - Data Analysis (Excel, SPSS)
        - Team Collaboration & Training

C. Work Samples / Projects

- Add 2 examples of your work: class projects, writing samples, mock designs, or other professional outputs.
- Include visuals (images, screenshots, or Canva graphics) along with a brief description of each project.

D. Contact Information

- Include a professional email address.
- Add at least one professional link, such as LinkedIn or GitHub.
    - Example:
        - Email: beza.tesfaye@protonmail.com
        - LinkedIn: linkedin.com/in/beza-tesfaye

4. Export / Share Portfolio

- Once your portfolio is complete, export it as a PDF or copy the shareable Canva link.
- Ensure your sharing settings allow others to view your portfolio.
    - For Canva links: Click Share → Link → Anyone with link can view

Exercise: Peer Review

Instructions

1. Upload Your Portfolio

- Submit your completed PDF portfolio or Canva link to the Moodle Workshop activity.

- Ensure your portfolio is accessible to peers:
  - For PDF: Confirm it opens properly.
  - For Canva link: Set sharing settings to “Anyone with the link can view.”

2. Evaluate Peers Using the Rubric

Learners will evaluate at least 2 peer portfolios using the following criteria:

Table 2: Rubrics for peer review

Criteria	*What to Look For*
Design (Visual Appeal & Readability)	Is the layout visually appealing, professional, and easy to read? Are fonts, colours, and spacing consistent and accessible?
Content (Completeness & Relevance)	Are all key sections included (Bio, Skills, Work Samples, and Contact Info)? Are descriptions clear and informative?
Professional Tone	Is the language appropriate, clear, and consistent with professional standards? Does the portfolio reflect a professional online identity?

2. Provide Constructive Feedback

When reviewing peer portfolios:

- *Compliment*: Highlight at least one strength in their portfolio.
  - Example: *“Your skills section is very clear and visually appealing with the icons it’s easy to read.”*
- *Suggestion for Improvement:* Offer at least one area for enhancement.
  - Example: *“Consider adding a brief description of each project to explain its impact it will help viewers understand your work better.”*

Guidelines for Effective Feedback:

- Be specific and actionable.
- Focus on design, content, and tone, not personal taste.
- Be respectful and professional.
- Encourage peers to reflect and make improvements based on your suggestions.

1.3 Creating CV & Resume

Your CV or Resume is often the first impression you make on employers, academic institutions, or clients. While a strong portfolio demonstrates your work visually, a CV/Resume provides structured information about your education, experience, skills, and achievements.

A well-crafted CV or Resume can:

- Secure interviews for jobs, freelance projects, or academic positions.
- Showcase your relevant skills and achievements clearly.
- Demonstrate professionalism and attention to detail.

CV vs. Resume

When applying for opportunities, it is important to know the difference between a CV (Curriculum Vitae) and a Resume. While both documents showcase your professional background, they differ in length, purpose, and content. Choosing the right format ensures your application is relevant and effective.

Table 3: CV vs Resume

Feature	*CV (Curriculum Vitae)*	*Resume*
Length	Detailed, often multiple pages	Short, usually 1 - 2 pages
Purpose	Academic positions, research roles, teaching, grants, fellowships	Job applications, internships, freelance opportunities, professional positions
Content	Full academic history, including education, publications, research projects, teaching experience, awards, professional affiliations	Targeted information: relevant skills, work experience, key projects, achievements directly aligned with the position

1. Create a CV or resume using online tools

In today’s digital world, creating a professional and attractive CV (Curriculum Vitae) or resume is easier than ever. You don’t need to be a design expert or spend hours formatting documents online tools can guide you through the process with ready-made templates, customization options, and even tips on what employers look for. Here are the three popular online tools you used to create CV or Resume:

i. Canva

Best for: Creative and visually appealing CVs

- *Features*:
  - Drag-and-drop design interface (easy to use, no design skills needed).
  - Hundreds of free and premium templates.
  - Ability to add icons, graphics, and colors to make your CV stand out.
  - Download options: PDF, PNG, or share via a link.
- *Strengths*:
  - Great for job roles in creative industries (design, marketing, media, and education).
  - Helps your CV look modern and visually engaging.
- *Limitations*:
  - May not be fully Applicant Tracking System (ATS) friendly some designs might confuse automated systems used by employers.

Tip: Use Canva for human-reviewed applications (when you send your CV directly to a person, not through an automated system).

ii. NovoResume

Best for: Professional, structured, and ATS-friendly resumes

- *Features*:
  - Step-by-step CV builder with guided prompts.
  - Templates optimized for ATS systems, ensuring your CV can be read by employer software.
  - Option to create multiple versions of your CV for different job applications.
  - Includes sections for skills, achievements, and keywords relevant to your field.
- *Strengths*:
  - Ideal for corporate or professional jobs (finance, law, administration, engineering).
  - Ensures your CV passes through digital screening systems.
- *Limitations*:
  - Free version has limited customization; premium plan unlocks more templates.

Tip: If applying through online job portals or large companies, NovoResume helps your CV get through the first automated filter.

iii. Zety

Best for: Customizable resumes with helpful writing tips

- *Features*:

    - Offers structured templates with flexible customization.
    - Built-in writing tips and examples for each section (summary, skills, experience).
    - Allows you to adjust fonts, layouts, and color schemes.
    - Exports in multiple formats (PDF, DOCX).
- *Strengths*:
    - Great for job seekers who need guidance on writing content, not just formatting.
    - Balances professional design with ATS-friendly structure.
- *Limitations*: Some downloads require a paid plan.

Tip: Zety is especially useful if you're unsure what to write in each CV section it suggests text you can adapt.

Table 4: CV and Resume creation tools

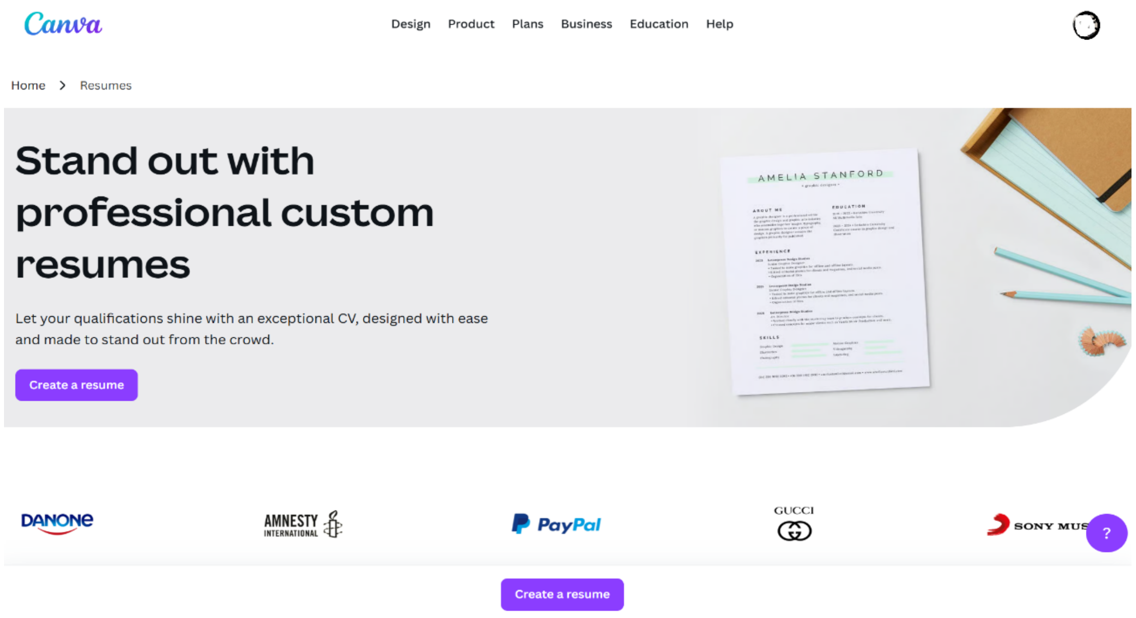
Tool	**Best For**	**Key Strength**	**Limitation**
Canva	Creative roles	Beautiful, modern designs	Not always ATS-friendly
NovoResume	Professional jobs	ATS compatibility	Limited free options
Zety	General job seekers	Writing guidance + templates	Some features paid

How to Create a Resume Using Canva

Step 1: Go to Canva

i. Open your browser and go to https://www.canva.com/resumes/

ii. Sign in or **create a free account** (you can use your Google, Facebook, or email).



Stand out with professional custom resumes

Let your qualifications shine with an exceptional CV, designed with ease and made to stand out from the crowd.

Figure 13: Canva resume create

Then select '**Create a resume**' button to create

Step 2: Choose a Resume Template

i. Browse the **"Resume"** templates.

ii. Use the search bar if you want specific styles like:

- "Professional resume"
- "Simple resume"
- "Creative resume"

Click on a template you like, then click **"Customize this template"**.

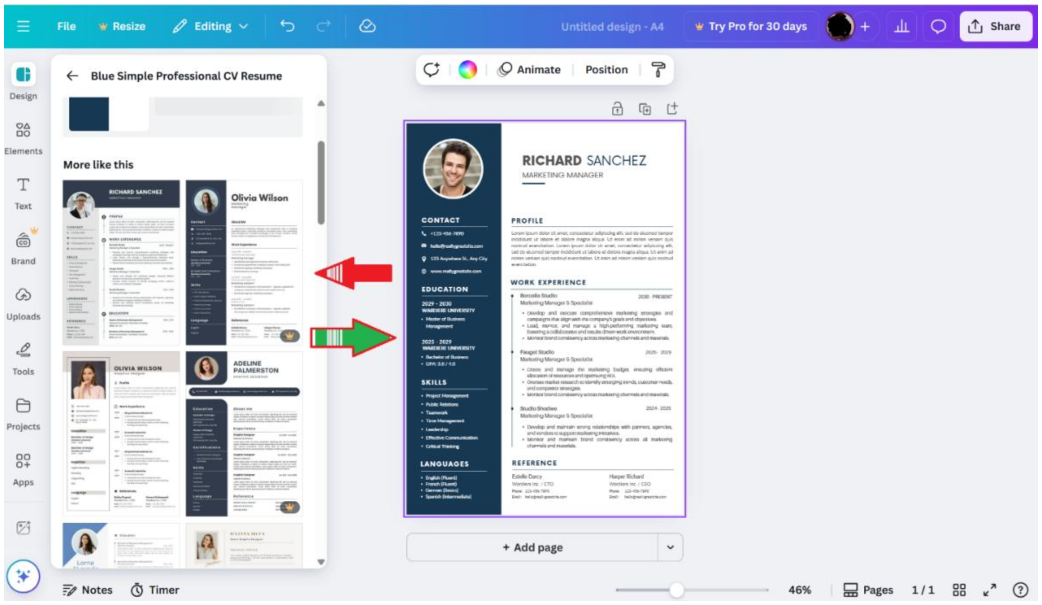


Figure 14: Resume template

Step 3: Fill In Your Information

Replace the placeholder text with your details:

- Full Name
- Job Title or Objective
- Contact Info (Phone, Email, Location)
- Summary or Profile: A short paragraph about yourself.
- Education**:** Include school, degree, and dates.
- Work Experience: Include job title, company, dates, and tasks.
- Skills: List technical and soft skills.
- Certifications or Projects (optional)

2. Modifying CVs and Resumes

A generic CV or resume often gets overlooked. To make your document stand out, you need to adapt it for each opportunity. Employers, clients, or academic committees look for specific alignment between your experience and their needs.

i. Modify Keywords

- Why it matters: Many organizations use Applicant Tracking Systems (ATS) that scan for keywords. If your CV or resume lacks these words, it may never be seen by a human.
- How to apply:

  - Review the job posting or academic call carefully.
  - Identify recurring keywords (e.g., “curriculum design,” “data analysis,” “project management”).
  - Naturally integrate those terms into your CV/Resume.
  - Example: Instead of writing *“Designed lesson plans”*, write *“Developed curriculum and lesson plans aligned with learning outcomes.”*

ii. Highlight Relevant Skills

- Why it matters: Employers/committees want proof that you have the right skills for their context.
- How to apply:
  - Prioritize role-specific skills.
  - Academic CV: Emphasize teaching, publications, grants, research methods.
  - Freelance Resume: Showcase technical skills, creative projects, or client work.
  - Example:
    - Academic CV - “Peer-reviewed publication in Journal of Digital Learning.”
    - Resume (freelance) - “Delivered 20+ Canva-based marketing designs for small businesses.”

iii. Order Sections Strategically

- Why it matters: The first sections are the most visible. Place the most relevant information up front.
- How to apply:
  - Academic CV: Start with *Education, Research, Publications, Teaching Experience*.
  - Resume for Jobs: Start with *Professional Experience and Achievements*.
  - Freelance Resume: Start with *Portfolio/Projects*, then *Skills and Services*.
  - Example:
    - A graphic designer applying for a freelance role should highlight “Portfolio of Designs” before “Education.”
    - A lecturer applying for a grant should place “Research Publications” at the top.

A personalized CV/Resume tells the reviewer:

“I am not just qualified I am the *right fit* for this specific role, project, or opportunity.”

Exercise:

1. Create your CV or Resume using Canva and download to pdf or share the link

➢ *Tip1: Choose a clean and professional design. Avoid overly flashy colours or fonts that distract from content.*

Example:

- An academic applicant may choose a simple and text-focused layout.
- A freelance designer may choose a creative, colourful layout with space for visuals.

➢ *Tip2: Add Personal Information, Education, Skills, and Work Experience*

- *Replace template placeholders with your own details.*
- *Personal Info: Full name, professional email, LinkedIn (avoid casual emails like coolguy99@gmail.com).*
- *Education: List degrees in reverse order (most recent first).*
- *Skills: Use bullet points or icons for quick scanning.*
- *Work Experience: Add role, organization, dates, and key achievements.*

Example:

- Education: *B.Ed. in Information Technology, Bahir Dar Institute of Technology, 2024.*
- Skills: Data Analysis | Digital Teaching Tools | Team Collaboration.
- Experience: *Intern ICT Assistant, XYZ School (2023)* → "Supported 200+ students with digital literacy training."

➢ *Tip 3: Customize for an Academic CV*

- *Add Academic Sections: create sections like Education, Academic Projects, or Coursework.*
- *Research & Projects: Include major assignments, group projects, or research papers relevant to your field.*
- *Teaching & Volunteering: List experiences such as tutoring classmates, mentoring, or leading study groups.*
- *Awards & Scholarships: Highlight academic honors, scholarships, or certificates of achievement.*

Example:

- Peer tutor for first-year programming students (2022–2023)
- Volunteer at local community tech workshop teaching basic computer skills

- Academic Excellence Scholarship, Bahir Dar University, 2021–2023
- Best Capstone Project Award, Computer Science Department, 2023

Or Customize for a Freelance Resume

Customize for a Freelance Resume

- Showcase practical, creative, and client-oriented skills.
- Projects: Include design work, coding projects, or lesson plans.
- Creative Samples: Add visuals, links, or brief project descriptions.
- Services Offered: List clear offerings (e.g., "Web Design, Graphic Design, and Digital Training").

Example:

- Projects: *Designed a digital skills training website for Ethiopian secondary schools.*
- Samples: *Created 10+ Canva-based infographics for local NGOs.*
- Services: *Freelance Web Design* | *Social Media Graphics* | *Training Modules.*

For Export PDF or Share Link

- Once finalized, click Download → PDF Print for a polished copy.
- Alternatively, click Share → Copy Link to generate a Canva link.
- Double-check that sharing permissions are set to "*Anyone with the link can view.*"

Summary

This Session focuses on building a professional online identity such as CV and resume, and portfolio which is essential for making a strong first impression in today's competitive digital landscape. The session defines a CV, resume and digital portfolio as a modern, visual, and interactive alternative to a traditional resume, and explains its importance for freelancers, educators, and job seekers. Participants learn about popular user-friendly platforms like Canva, Wix, and Google Sites for creating these portfolios. The core of the session is a guided, hands-on activity where learners use Canva to build their own portfolio, ensuring it includes four key sections: an "About Me" bio, skills, work samples, and contact information. The session concludes with a peer review activity where participants evaluate each other's portfolios based on design, content, and professional tone, reinforcing their understanding and providing constructive feedback to improve their professional digital presence.

Review Questions

1 Which of the following BEST defines a professional portfolio?

A. A collection of random personal photos

B. A structured collection showcasing skills, experiences, and achievements

C. A private diary of learning experiences

D. A list of social media profiles

2 In a digital portfolio, including work samples is important because:

A. It makes the portfolio longer

B. It provides evidence of your skills and achievements

C. It replaces the need for a CV

D. It fills empty space

3 Which of the following is NOT typically a section of a professional portfolio?

A. Personal Statement/Introduction

B. Hobbies unrelated to career goals

C. Work Samples/Projects

D. Education & Certifications

4 The “Contact Information” section in a portfolio should include:

A. Full address, including street and house number

B. Professional email and LinkedIn profile

C. Nicknames and personal social media handles

D. Only a phone number

Answer the Following Questions

1 Explain why a professional digital portfolio is more effective than a traditional printed CV in today’s job market.

2 Describe how a professional online identity can influence employers’ perceptions of you.

3 Compare and contrast the design and functionality of two portfolio platforms (e.g., Canva vs. Google Sites). Which one best suit your professional goals, and why?

4 Analyze a sample portfolio and identify areas of strength and improvement in design, content, and tone using the given rubric.

5 Assess the suitability of your chosen CV format (academic or professional) for your current career path. Justify your choice.

Session 2: Exploring Digital Job Platforms

Introduction

Dear learners, in this session, you will explore digital job platforms and unlock ways to connect with job opportunities.

In today's digital world, more and more people look for job opportunities online through freelancing platforms. Such platforms include Upwork, Fiverr, PeoplePerHour, and Toptal are used to connect skilled individuals with clients around the world who need services like design, writing, tutoring, programming, and more. This session introduces learners to job searching platforms and helps them understand how to create a professional presence on freelancing platforms. By exploring platforms and designing a mock profile with a service listing (gig), learners will practice how to showcase their skills in ways that attract potential clients or employers.

Let's start on your digital journey by asking a few questions to explore your experience on digital job platforms.

- Which digital job platforms you use to search a job online?
- Which platform is better? In what area?

This session helps you step toward to build a strong online presence and preparing for future freelance or remote career opportunities.

Learning Objectives

By the end of this session, learners will be able to:

- Identify major freelancing platforms
- Explain what the concert economy is and how digital platforms support it.
- Create a mock freelancing profile for (gig) relevant to their skills.

Content Outline

2.1. Freelancing Platforms Overview

Freelancing platforms are digital marketplaces where individuals can offer their skills and services to clients around the world. Each platform has its own style, target audience, and level of competitiveness. Understanding the strengths of different platforms helps you choose the right one to start or grow your freelancing journey.

1. Upwork

- Overview: One of the largest professional freelancing platforms.
- Address: https://www.upwork.com/
- How it works: Clients post projects, and freelancers submit proposals with bids (hourly rate or fixed price).
- Best for: Skilled professionals in fields like IT, writing, design, marketing, and consulting.
- Unique Features: Secure payment protection, client reviews, and milestone-based contracts.
- Challenge: Highly competitive; beginners may find it difficult to land their first job.

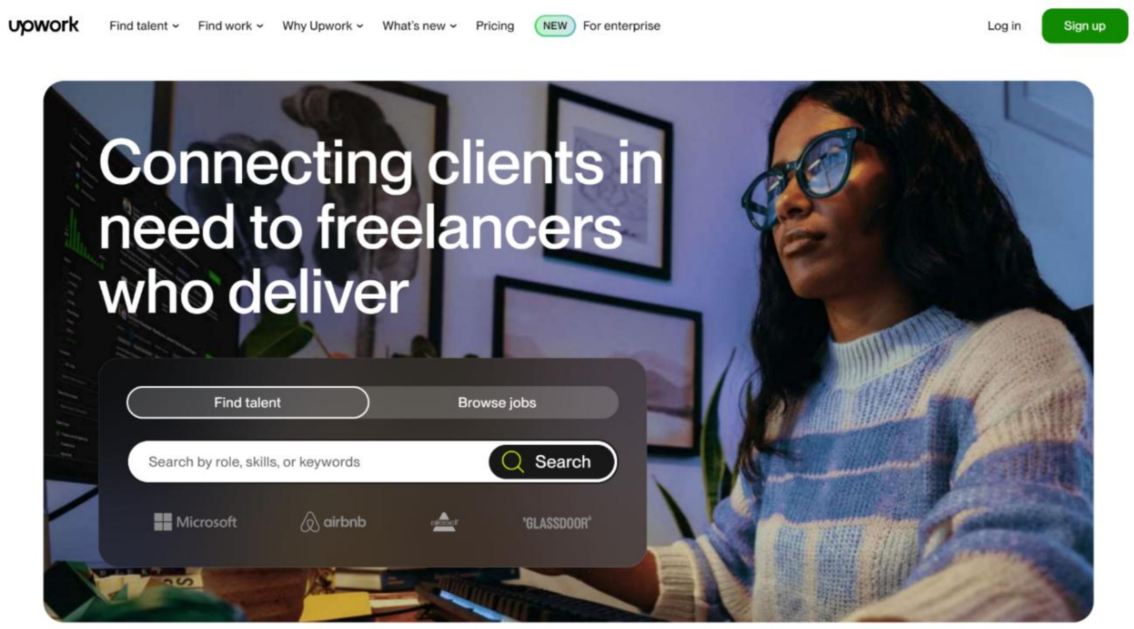


Figure 15: Upwork landing page

2. Fiverr

- Overview: A gig-based platform where freelancers create service listings called “gigs.”
- Address: https://www.fiverr.com/

- How it works: Instead of bidding, freelancers set up their services (e.g., “I will design a logo for your business”) with clear pricing tiers.
- Best for: Beginners and creative professionals in graphic design, writing, music, social media, and digital marketing.
- Unique Features: Easy to start, no bidding required, strong focus on creative services.
- Challenge: High competition at the entry level; requires good marketing and reviews to stand out.

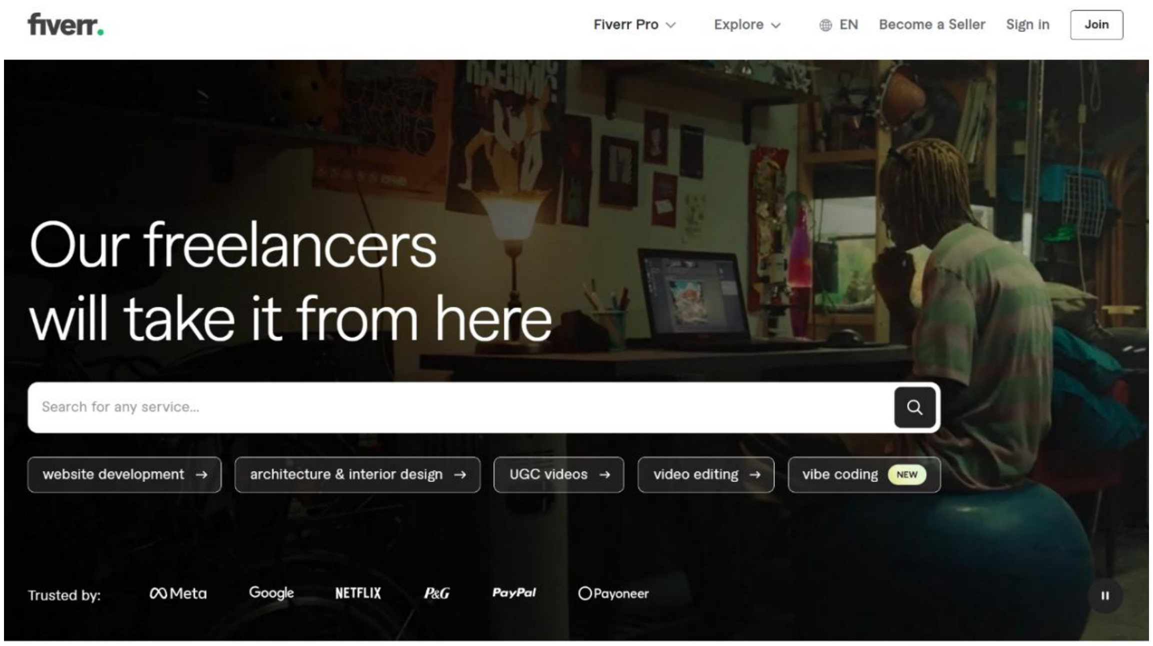


Figure 16: Fiverr landing page

3. PeoplePerHour

- Overview: A UK-based freelancing platform combining elements of Upwork and Fiverr.
- Address: https://www.peopleperhour.com/
- How it works: Freelancers can create hourly gigs or apply to posted projects.
- Best for: Flexible work seekers in design, development, writing, translation, and admin support.
- Unique Features: Option to work on both short-term tasks and long-term contracts.
- Challenge: Smaller client pool compared to Upwork or Fiverr, but less crowded in some niches

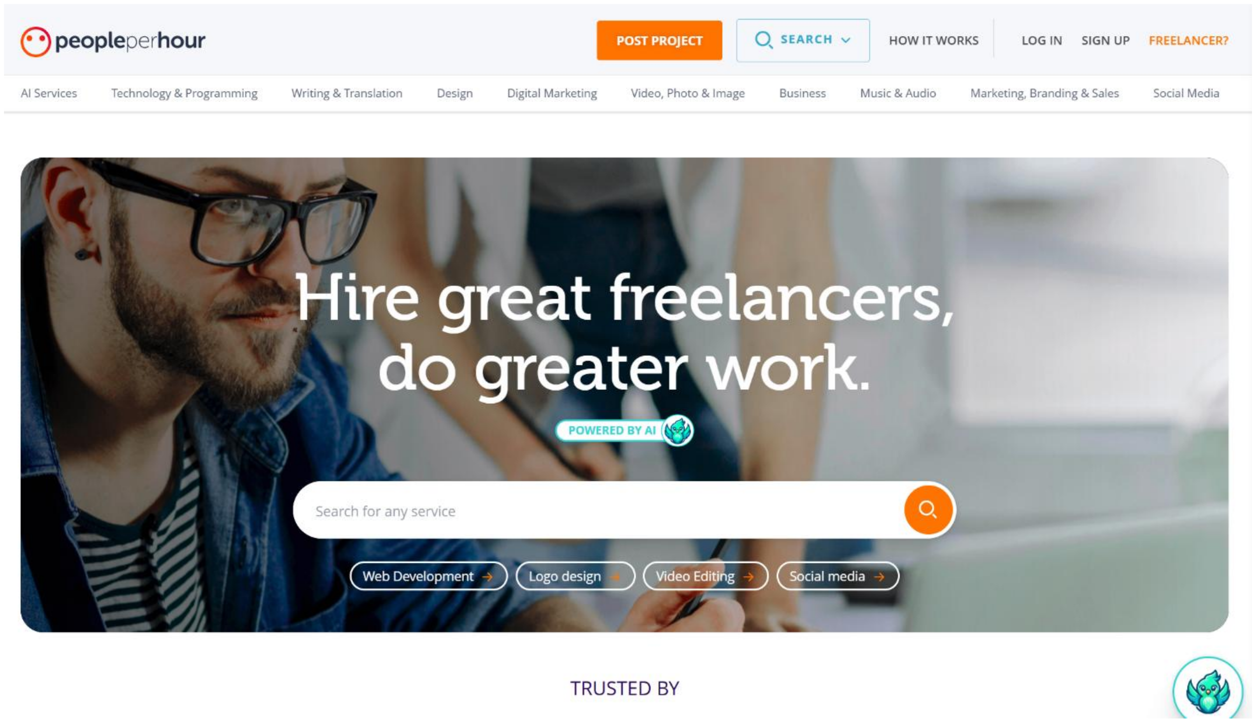


Figure 17: Peopleperhour landing page

4. Toptal

- Overview: A premium freelancing platform that accepts only the top 3% of applicants.
- Address: https://www.toptal.com/
- How it works: Freelancers go through a rigorous screening process (tests, interviews, skill verification).
- Best for: Highly experienced professionals in software development, finance, project management, and design.
- Unique Features: Access to top-tier clients (startups and large companies like Airbnb, Microsoft).
- Challenge: Very competitive entry; not ideal for beginners.

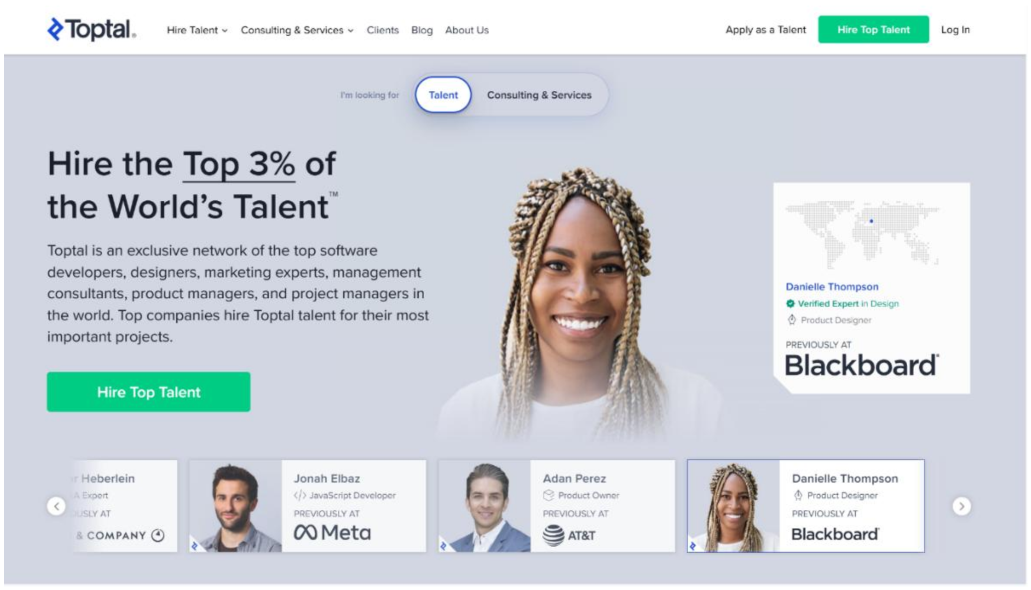


Figure 18:Toptal landing page

Tip for Learners:

- If you're a beginner, start with Fiverr to build confidence and collect reviews.
- If you already have professional experience, try Upwork or PeoplePerHour.
- For advanced experts, Toptal can open doors to high-paying global clients.

2.2. Understanding the Gig Economy

The gig economy is a modern labour market where individuals work on short-term projects, freelance jobs, or "gigs" instead of holding traditional, long-term employment and being tied to a single employer, freelancers connect with multiple clients and earn income project by project. This system is powered by digital platforms (like Fiverr, Upwork, and Toptal), which make it easy for clients to find skilled professionals across the globe.

Key Features of the Gig Economy

1. *Flexibility*
   - Freelancers decide when, where, and how much they want to work.
   - Example: A student might design websites in the evening while studying during the day.
2. *Global Reach*
   - Work is no longer limited to your local area.
   - Freelancers in Ethiopia can work for clients in the USA, UK, or Asia.
   - This opens opportunities for diverse projects and higher earning potential.

3. *Skills-Based Earnings*
   - Success depends on the quality of work, communication, and reliability.
   - The more in-demand your skill (e.g., coding, design, copywriting), the higher you can charge.
   - Building a strong portfolio and client reviews increases your chances of getting projects.

Examples of Digital Gigs:

Freelancers can offer almost any digital service, including:

- Logo & Graphic Design: Creating brand identities for businesses.
- Website & App Development: Building websites, apps, or online stores.
- Online Tutoring: Teaching languages, math, coding, or digital skills remotely.
- Data Entry & Virtual Assistance: Organizing spreadsheets, emails, and documents.
- Voiceovers & Audio Editing: Recording voice for ads, audiobooks, or explainer videos.
- Digital Marketing: Running social media campaigns, SEO optimization, or ad management.

Why It Matters for Learners?

- The gig economy provides real-world opportunities to earn money while building skills.
- It can serve as a stepping stone toward full-time freelancing or supplementing income alongside studies or jobs.
- Learners should view freelancing not just as temporary work, but as a way to practice entrepreneurship and self-branding.

The gig economy is fast-growing and skill-driven. By learning how to market your abilities online, you can tap into a global marketplace and create flexible career opportunities.

2.3 Procedures for Set up Fiverr

This section will guide learners step-by-step on how to set up a Fiverr freelancing account and create their first service listing (called a Gig).

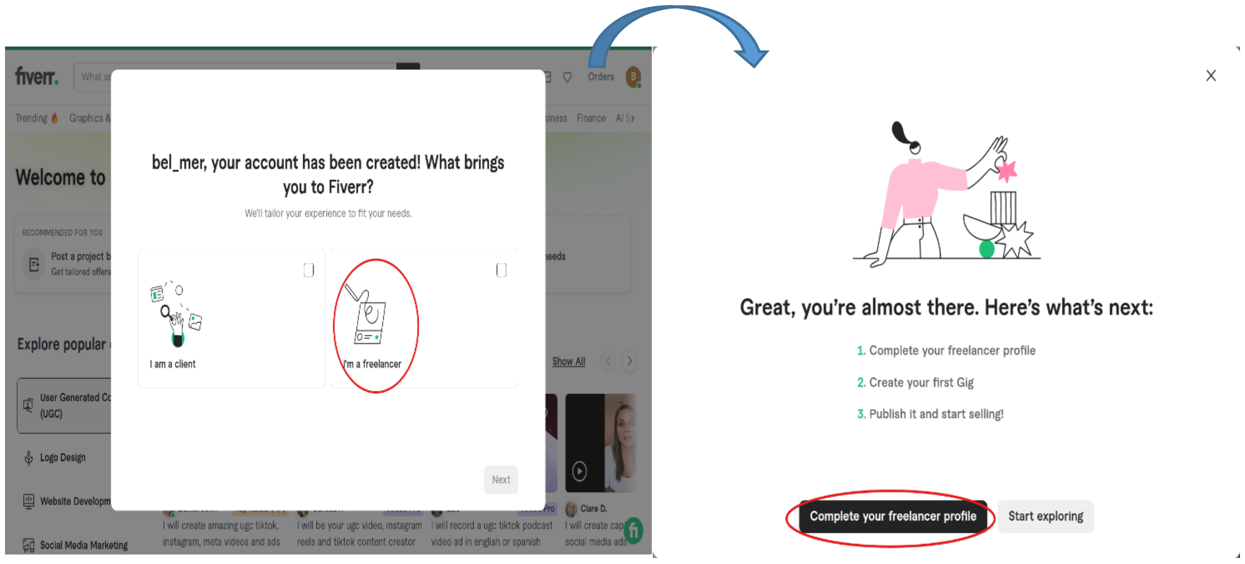
Step 1: Visit Fiverr & Sign Up

- Go to www.fiverr.com.
- Click Join. You can sign up using Google, Facebook, or email.

- Choose “Seller Account” to work as a freelancer (not a buyer).

Tip: Use your professional email address to keep accounts consistent with your freelancing brand.

Step 2: Create Profile


bel_mer, your account has been created! What brings you to Fiverr?

We'll tailor your experience to fit your needs.

Great, you're almost there. Here's what's next:

1. Complete your freelancer profile
2. Create your first Gig
3. Publish it and start selling!

Figure 19: Fiverr account creating

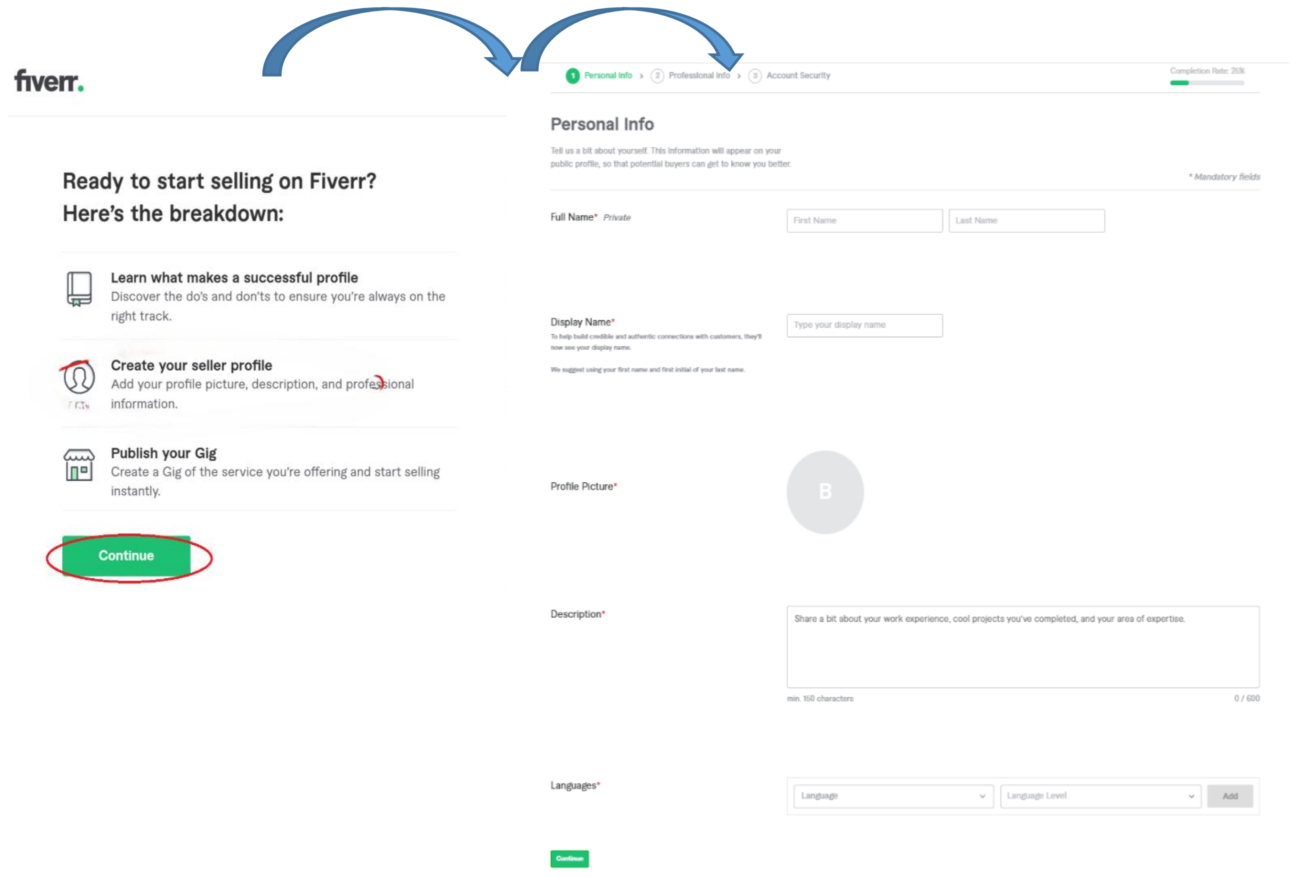
Your Fiverr profile is your digital identity it’s the first impression clients get.

What to include:

- Profile Picture: A clear headshot, smiling, well-lit (avoid selfies with distractions).
- Bio (50 - 100 words): State who you are, your expertise, and what you offer.

Example Bio:

“I am a creative graphic designer with 2+ years of experience in Canva and Adobe tools. I specialize in designing social media graphics, business cards, and professional portfolios tailored to client needs. My goal is to help brands stand out with clean, modern, and impactful designs.”


fiverr.

Ready to start selling on Fiverr?
Here's the breakdown:

Learn what makes a successful profile
Discover the do's and don'ts to ensure you're always on the right track.

Create your seller profile
Add your profile picture, description, and professional information.

Publish your Gig
Create a Gig of the service you're offering and start selling instantly.

Continue

1 Personal Info › 2 Professional Info › 3 Account Security

Completion Rate: 25%

Personal Info

Tell us a bit about yourself. This information will appear on your public profile, so that potential buyers can get to know you better.

* Mandatory fields

Full Name* *Private* — First Name / Last Name

Display Name* — Type your display name
To help build credible and authentic connections with customers, they'll now see your display name.
We suggest using your first name and first initial of your last name.

Profile Picture*

Description* — Share a bit about your work experience, cool projects you've completed, and your area of expertise.
min. 150 characters — 0 / 600

Languages* — Language / Language Level / Add

Continue

Figure 20: Fiverr profile creation

Step 3: Add Your First Gig (Service)

On Fiverr, your services are called Gigs.

- Click Create a New Gig.
- Write a clear title in the format:

  "I will [do this] for [client need]."

Example Title:

"I will design a modern, eye-catching logo for your business."

Tip: Be specific. A good title is both descriptive and client-focused.

Step 4: Write Description & Add Keywords

This is where you explain what you offer and why clients should hire you.

Example Description:

"Need a unique logo to represent your brand? I will design a professional, high-quality logo that reflects your business identity. I focus on creativity, originality, and attention to detail. You will receive fast delivery, unlimited revisions, and a design that truly represents your vision."

- Add Keywords (tags): These help clients find your Gig.
  Example Tags: *logo design, graphic design, and branding, creative logo.*

Step 5: Set a Price Range (Mock for Class Activity)
Fiverr allows you to create packages (Basic, Standard, and Premium).
Example Pricing Structure:

- Basic - $5: Simple logo design.
- Standard - $20: Logo + 1 revision.
- Premium - $50: Full branding package (logo, business card, social media kit).

Tip: In real freelancing, prices should reflect your skill level and time investment. For beginners, start low to build reputation.

Exercise
Create a mock Fiverr profile and add one Gig. The focus is not on earning money but on understanding the platform's workflow.

Summary

This session introduced learners to freelancing platforms and the gig economy, focusing on how digital marketplaces such as Upwork, Fiverr, PeoplePerHour, and Toptal connect skilled individuals with global clients. Learners explored the features, benefits, and challenges of these platforms and learned how to choose the right one based on their skills and experience. The session explained the concept of the gig economy, its flexibility, global reach, and skills-based earning potential, highlighting its importance in creating real-world opportunities for income and professional growth.

Participants engaged in a step-by-step walkthrough of Fiverr, learning how to set up a seller account, create a professional profile, and design a service listing (Gig).

Review Questions

1. Which platform is known for connecting businesses with top vetted freelancers, often for high-end projects?
   A. Fiverr
   B. Upwork
   C. Toptal
   D. PeoplePerHour
2. On which platform do freelancers usually create “gigs” or service listings to attract clients?
   A. Upwork
   B. Fiverr
   C. Toptal
   D. LinkedIn
3. PeoplePerHour primarily matches freelancers with clients by:
   A. Posting gigs for fixed prices only
   B. Allowing freelancers to send proposals to client projects
   C. Offering only hourly contracts
   D. Connecting only tech professionals

Answer the Following Questions

1. Define what a freelancing platform is and explain its primary purpose.
2. List four examples of major freelancing platforms and one key feature of each.
3. Choose one freelancing platform (e.g., Upwork, PeoplePerHour) and explain step-by-step how you would create your profile there.
4. Compare Upwork and Fiverr in terms of accessibility for beginners, earning potential, and competition.
5. Identify at least three digital skills you can further develop to become more competitive on global freelancing platforms.

Session 3: Digital Entrepreneurship & Online Income

Introduction

Dear learners, in this session, you will explore digital entrepreneurship and online income generation.

In today's digital world, earning money is no longer limited to traditional office jobs. The internet offers endless opportunities to monetize skills, create online businesses, and reach global audiences. This session introduces learners to different methods of online income generating, safe digital payment systems, and how to draft a simple online business plan.

Let's start your journey by asking one questions to explore your experience in digital income generation.

Learning Objectives

By the end of this session, learners will be able to:

- Explain ways to monetize digital skills in the online economy.
- Identify digital payment systems and secure transactions.
- Draft a simple online business plan to organize and test a digital business idea.

Content outline

3.1 Ways to monetize digital skills in the online economy
3.2 Online Payment Systems and secure transactions
3.3 Preparing online business plan and test digital business idea.

3.1 Ways to monetize digital skills in the online economy

Do you have any experience that you paid online using digital platforms?
Digital entrepreneurship provides a wide range of opportunities for individuals to earn income by leveraging their skills, creativity, and digital tools. In this section, you will explore practical ways to monetize digital skills.
a) Freelancing

Freelancing involves offering services to clients on a project-by-project basis. Unlike traditional jobs, freelancers are not tied to one employer and can work with multiple clients simultaneously.

- Common Services: Graphic design, programming, data entry, tutoring, translation, digital marketing, writing.
- Popular Platforms:
    - *Upwork:* Professional freelancing with project contracts and competitive bidding.
    - *Fiverr:* Gig-based platform where services start at $5. Great for beginners.
    - PeoplePerHour: Flexible platform for short- and long-term projects.

Example: A computer science student can offer basic web development services on Upwork, creating simple websites for small businesses.

b) Content Creation

Content creators earn money by producing and sharing digital content that attracts audiences. The larger and more engaged the audience, the higher the potential for income through advertisements, sponsorships, and direct fan support.

- Types of Content:
    - YouTube videos: Tutorials, entertainment, educational content and etc.
    - Blogs: Written articles on niche topics (travel, tech, lifestyle).
    - Podcasts: Audio shows on topics like business, culture, or education.
    - TikTok: Entertainment, education, marketing, and community.
- Revenue Streams:
    - Ads (Google AdSense on blogs or YouTube).
    - Sponsorships (companies paying for brand exposure).
    - Memberships/Donations (Patreon, memberships/community, Buy Me a Coffee).
    - In-App Purchases (Virtual Gifts & Coins)

Example: A learner who enjoys explaining math concepts could create a YouTube channel offering short tutorials. Over time, they can monetize through ads and sponsorships.

c) E-commerce

E-commerce is the buying and selling of goods or services online. Entrepreneurs can sell both physical products and digital items directly to customers.

- Platforms to Start:
    - Etsy: Handmade crafts, digital downloads, and templates.

    - Shopify: Build a personal online store.
    - Amazon: Sell products to a global marketplace.
- Types of Products:
    - Physical: Clothing, accessories, handmade crafts.
    - Digital: printable, templates, software, and design assets.

Example: A learner skilled in digital art can sell custom illustrations as printable wall art on Etsy.

d) E-books & Digital Products

Digital products can be created once and sold multiple times, making them a scalable source of income.

- Types of Digital Products:
    - E-books (self-published guides, stories, or tutorials).
    - Templates (resumes, social media posts, presentations).
    - Online courses (via Udemy, Teachable, or personal websites).
    - Toolkits or downloadable resources.
- Advantages:
    - Low production cost (mainly time and creativity).
    - Can be sold globally without shipping.
    - Passive income potential.

Example: A student with strong Canva design skills can create resume templates and sell them on Fiverr, Etsy, or Gumroad.

Each monetization method has its own requirements, benefits, and challenges. Learners should:

- Match skills to the right method (e.g., writers → freelancing or e-books, designers → freelancing or templates).
- Start small, experiment with one platform, and grow gradually.
- Combine methods for multiple income streams (e.g., freelancing + selling digital products).

3.2 Online Payment Systems and secure transactions

To successfully earn income online, freelancers and digital entrepreneurs must know how to receive payments safely and conveniently. Trusted digital payment systems bridge the gap between clients (often global) and local bank accounts or mobile wallets.

a) PayPal

- What it is: A globally recognized digital payment platform.
- Best for: Freelancers, e-commerce sellers, and small businesses that need a widely accepted payment method.
- How it works:
  - Create a free account at https://www.paypal.com/pk/home
  - Link your bank account or card.
  - Receive payments from clients via your PayPal email.
  - Withdraw funds to your bank account.
- Advantages:
  - Accepted in most countries worldwide.
  - Provides buyer and seller protection.
  - Easy to integrate with e-commerce sites like Shopify or WooCommerce.
- Limitation: In Ethiopia, full PayPal functionality may be limited.

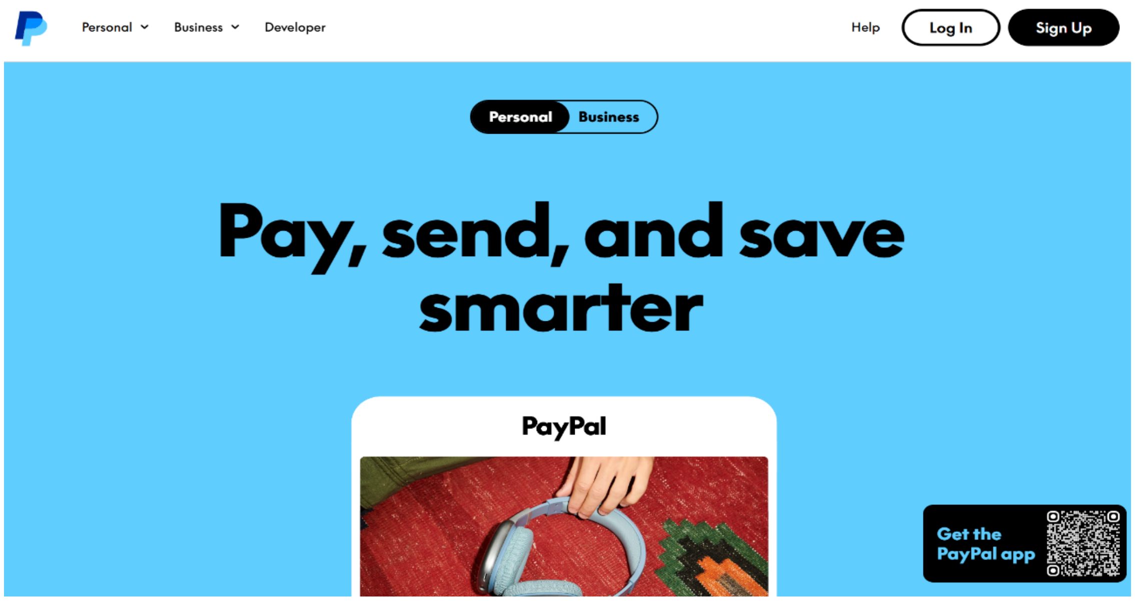


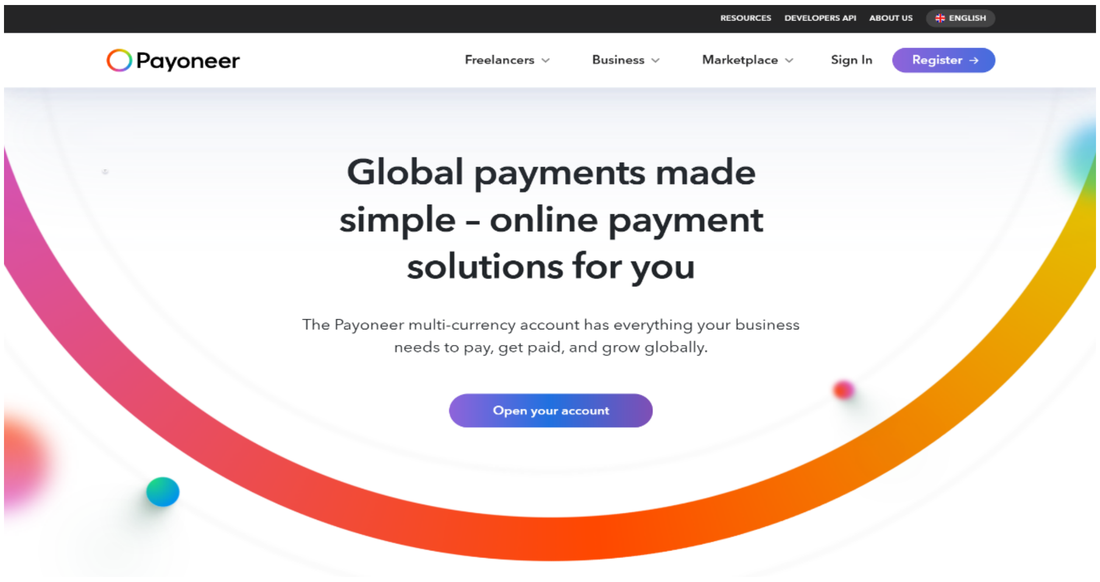
Figure 21: PayPal page

Example: A freelance graphic designer working with a U.S. client can receive payment directly into their PayPal account.

b) Payoneer

- What it is: A payment solution designed for freelancers and international transactions.
- Best for: Freelancers on platforms like Upwork, Fiverr, and PeoplePerHour.
- How it works:
    1. Sign up at payoneer.com.
    2. Connect your freelancing platforms (e.g., Fiverr, Upwork).
    3. Clients pay through the platform, and Payoneer transfers money to your local bank.
    4. Optionally, order a Payoneer Prepaid Mastercard for ATM withdrawals.
- Advantages:
    - Accepted by most global freelancing platforms.
    - Direct withdrawal to local banks in many countries.
    - Competitive exchange rates for currency conversion.
- Limitation: Service fees apply per withdrawal or transfer.

Example: An Ethiopian freelancer on Fiverr can use Payoneer to receive payments and then withdraw to a local bank account.



Global payments made simple – online payment solutions for you

The Payoneer multi-currency account has everything your business needs to pay, get paid, and grow globally.

Figure 22: Payoneer page

c) Mobile Money Systems

Mobile money platforms provide simple, local solutions for sending and receiving money via mobile phones.

- Examples:
  - M-Pesa Ethiopia: One of the most successful mobile money systems worldwide.
  - Telebirr: Enables digital transactions, bill payments, and transfers without needing a bank account.
- Advantages:
  - Works with just a mobile phone (no bank needed).
  - Convenient for daily transactions and local withdrawals.
  - Safer than carrying cash.
- Limitations:
  - May not directly integrate with international freelancing platforms.
  - Often requires linking with services like Payoneer for global transactions.

Example: A local online tutor in Ethiopia can charge students using Telebirr for lesson fees.

Tips:

1. Link your payment system to a secure bank account or mobile wallet you control.
2. Enable two-factor authentication (2FA) for account security.
3. Keep personal financial details private (never share full card or bank numbers with clients).

3.3 Preparing online business plan and test digital business idea

In today's digital economy, even small business ideas can grow quickly if planned properly. A mini business plan helps you clarify your idea, identify your audience, choose the right platform, and set up income and payment methods.

In this activity, learners will draft their own simple online business plan during class, step by step.
Step 1: Define Your Business Idea
What to do:
Write down one clear sentence describing the product or service you want to offer online. Keep it simple and specific.

Example:

"Online tutoring in maths for high school students."

Other examples learners could try:

- Selling handmade jewellery through Instagram.
- Offering graphic design services on Fiverr.
- Starting a food delivery service in Bahir Dar with WhatsApp ordering.

Step 2: Identify Your Target Audience

What to do:

Ask yourself: Who are my customers? Define their age, location, needs, and interests.

Example:

High school students aged 14–18 needing help in algebra and geometry.

Other examples learners could try:

- University students who need affordable digital art commissions.
- Office workers in Bahir Dar who want healthy lunch deliveries.
- Parents looking for safe, educational online games for children.

Step 3: Choose Your Platform

What to do:

Decide where you will promote and deliver your service or product. Think about where your audience already spends time online.

Example:

- YouTube (free maths tutorials to attract viewers).
- Fiverr (paid 1-to-1 tutoring sessions).
- Facebook & Telegram for promoting handmade crafts.
- Instagram for fashion products.
- TikTok for short cooking tutorials.
- A personal website for online bookings.

Step 4: Outline Your Income Method

What to do:

Decide how your business will generate money. Will you charge per product, earn from ads, or set up subscriptions?

Example (Online Tutoring):

- Subscriptions (YouTube channel members).
- Ad revenue from YouTube videos.
- Service fees for private tutoring sessions.
- Selling products directly (e.g., jewellery, crafts, clothing).
- Affiliate marketing (earning commission by recommending products).
- Sponsored posts (companies pay you to promote their brand).

Step 5: Plan Digital Payment Method

What to do:

Decide how your customers will pay you. Choose digital payment methods that are safe, reliable, and accessible to your audience.

Example (Tutoring Business):

- PayPal (international clients).
- Telebirr (Ethiopian clients).

- CBE Birr for local transactions.
- Bank transfer or Visa/Mastercard for global clients.
- Mobile money wallets like M-Pesa.

Exercise:

Draft Your Mini Business Plan

Combine the answers into a short, 5-step business plan:

1. Business Idea: ______________________
2. Target Audience: ______________________
3. Platform: ______________________
4. Income Method: ______________________
5. Payment Method: ______________________

Summary

This session explored how individuals can leverage digital skills to create sustainable online income. Learners examined key opportunities, including freelancing, content creation, e-commerce, and selling digital products. Popular platforms such as Upwork, Fiverr, YouTube, and

Shopify were introduced, along with secure payment systems like PayPal, Payoneer, and Telebirr. The importance of security measures, including two-factor authentication, was highlighted. Finally, participants practiced creating a mini business plan covering their idea, target audience, platform choice, income method, and payment options.

Review Questions

1. Digital entrepreneurship refers to:
   A. Selling physical goods in a local store
   B. Using digital technologies to create and grow businesses online
   C. Working only in traditional office jobs
   D. Investing only in stock markets
2. Which of the following is an example of digital entrepreneurship?
   A. Starting a YouTube channel that earns ad revenue
   B. Opening a roadside cafe
   C. Teaching only in a physical classroom
   D. Running a traditional taxi service
3. A key advantage of online income opportunities is:
   A. Limited market reach
   B. High entry costs
   C. Global access to customers
   D. Inflexible working hours
4. What is a common way content creators earn money online?
   A. Buying and reselling products
   B. Doing data entry jobs
   C. Monetizing through ads, sponsorships, or donations
   D. Teaching in physical classrooms
5. What monetization method fits a learner who enjoys teaching and wants to build a YouTube audience?
   A. Selling crafts on Instagram
   B. Offering private lessons only
   C. Posting tutorials and monetizing with ads
   D. Writing blog articles only

Answer the Following Questions

1. Define digital entrepreneurship in your own words.
2. Compare freelancing and e-commerce as income generation methods. What are the advantages and challenges of each?
3. Evaluate which digital payment method (Payoneer or Telebirr) would be most suitable for Ethiopian freelancers and justify your choice.
4. Propose an innovative digital product or service you could offer online using your personal or academic skills.

Capstone Project: Digital Citizenship and Online Participation

1. Purpose: This Capstone Project enables learners to apply digital citizenship principles in a real-life Ethiopian scenario. Learners will analyze a case, design a responsible online identity plan, and demonstrate how to engage safely and professionally in Ethiopia's digital environment.
2. Case Scenario – "Hiwot's Digital Journey"
   Hiwot, a young graduate from Bahir Dar, wants to start freelancing online. She creates a LinkedIn profile and posts her design portfolio. She receives two offers, one from a verified NGO and another from an unknown person requesting her bank details. Hiwot must decide how to stay safe, verify opportunities, and maintain a professional online identity.
3. Project Tasks: Learners act as digital consultants for Hiwot or a similar Ethiopian professional.
   Tasks:

   I. Analyze the case: Identify the digital citizenship issues and risks.
   II. Develop a Digital Identity Plan:
      - Recommend professional online platforms (LinkedIn, Gebeya.com, Fiverr, etc.).
      - Suggest at least three online safety practices.
      - Identify two possible online income opportunities relevant to Ethiopia.
      - Describe ethical online behavior.

   III. Reflect: Explain what you learned about responsible online participation in Ethiopia.
   IV. Submit a Report (2–3 pages): Present your analysis, plan, and reflection.

Module 3: Web Authoring and Publication

Introduction

This module will equip you with the essential skills to create and manage modern websites. You will start with Webpage Development Basics using HTML, then learn Cascading Style Sheets (CSS) to enhance visual design.

Next, you will explore JavaScript to make web pages interactive and Basics of PHP for dynamic server-side functionality. Finally, you will apply your knowledge by Developing a Website Using WordPress, learning to design, customize, and maintain professional websites efficiently.

By the end of this module, you will be able to build functional, interactive, and visually appealing websites, laying a strong foundation for further web development skills.

Session1: Webpage Development Basics

Introduction

Dear learners, in this session, you'll explore the fundamentals of building and designing web pages.

In this session, you will explore the basics of web development and understand how websites are built. You will learn the structure of an HTML document and how to use common tags to create content such as text, lists, tables, images, and links. You will also practice creating a simple webpage to see how these elements work together in action. Finally, you will be introduced to HTML forms, which allow users to interact with websites by entering and submitting information. By the end of this session, you will have the foundational knowledge needed to begin designing and building your own webpages using HTML.

Learning Objectives

By the end of this session, learners will be able to:

- Explain the role of HTML in webpage creation.
- Identify and apply common HTML tags.
- Develop a simple webpage.

- Design and implement basic HTML forms for user input.

Content Outline

1.1 Introduction to web development

1.2 Basic of HTML

1.3 Creating a simple webpage

1.4 HTML Forms

1.1 Introduction to web development

A web page is a digital page you see in your browser. It can contain text, images, videos, links, and multimedia elements.

To build any webpage, three main building blocks work together:

- HTML (HyperText Markup Language): gives the structure
- CSS (Cascading Style Sheets): adds the style and layout
- JavaScript: Adds interactivity and dynamic behavior.

Web pages are the building blocks of websites. A website may consist of a single web page or multiple interconnected pages.

There are two types of webpages

A. Static Web Pages

Static pages are delivered exactly as stored on the server, built with HTML, CSS, and JavaScript. Common uses: portfolios, resumes, informational sites, and blogs.

B. Dynamic Web Pages

Dynamic pages generate or change content at runtime based on user interactions or databases, using server-side languages (PHP, Python, and Node.js) with HTML, CSS, and JavaScript. Examples: social media, online shopping, news sites.

What is Web Development?

Web development is the process of creating websites and making them work on the internet. It involves designing, building, and maintaining websites.

Web development is divided into two sides:

- Frontend: what users see on the screen like layout, text, images, buttons, colors, navigation

- Backend: what happens behind the scenes → database, server.

Example: When you visit Facebook, the frontend is the news feed, buttons, and design you see. The backend handles storing your posts, friends, and likes in a database.

How the Web Works

When you open a web page in your browser, here's what happens step by step:

1. Browser sends a request: Your browser asks a web server for the page you want.
2. Server processes the request: The server finds the requested files (HTML, CSS, JavaScript, images) and sends them back.
3. Browser renders the page: Your browser reads the HTML to structure the content, applies CSS for styling, and runs JavaScript to make the page interactive.
4. You see the web page: The page appears on your screen, ready to use.

1.2 Basics of HTML

What is HTML?

HTML stands for **Hypertext Markup Language**. It is the basic language used to create webpages.

- Hypertext means the text can contain links (called hyperlinks) that connect one page to another.
- Markup language means HTML uses tags (symbols written inside < >) to describe how the webpage should look and what elements it should include.

How HTML Works?

When a web browser shows a webpage, it first reads a text file written in HTML. Inside this file, there are special codes called tags. Tags are written between the symbols < and >.
A tag tells the browser what to display on the page (for example, a heading, a paragraph, or an image).
Most tags come in pairs:

- Opening tag: <tagname>
- Closing tag: </tagname> (notice the / slash)

The text between the opening and closing tag is the content that will appear on the page.

Example

<h3>What are HTML tags? </h3>: This tag tells a web browser to display the text 'What are HTML tags?' in the style of header level 3.

The Basic HTML structure

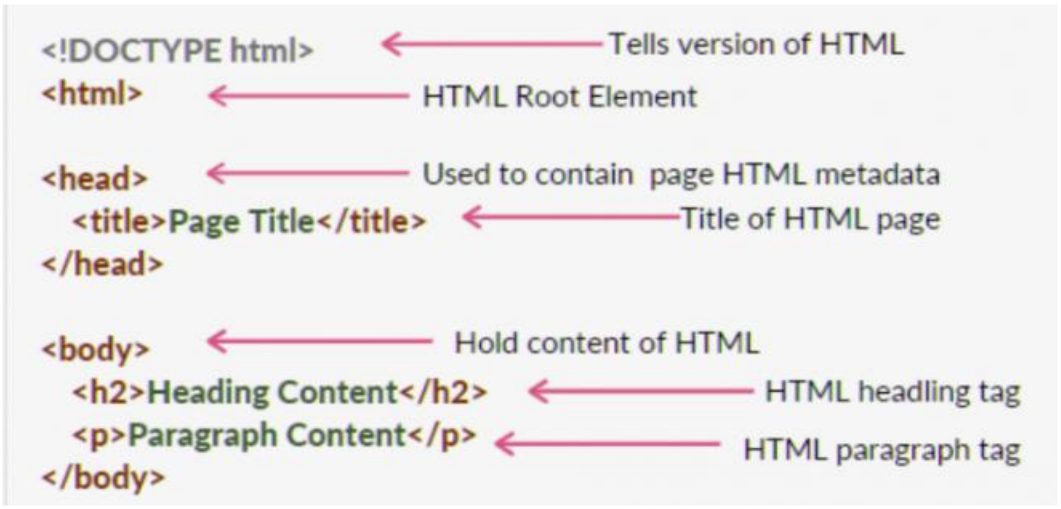
Every HTML page follows a basic structure made up of three main parts:

i. <html> … </html>
   - Wraps the entire HTML document.
   - Contains two main parts: <head> and <body>

ii. <head> … </head>
   - Provides document metadata (not shown on screen).
   - Includes tags like::
     - <title> → The page title (appears in the browser tab).
     - <meta> → Metadata (keywords, description, character set).
     - <style> → CSS styles for the page.
     - <script> → JavaScript for interactivity.

iii. <BODY>…. </BODY>
   - Contains all visible content: text, media, links, etc.

Putting It All Together to create a single webpage



```
<!DOCTYPE html>
<html>

<head>
  <title>Page Title</title>
</head>

<body>
  <h2>Heading Content</h2>
  <p>Paragraph Content</p>
</body>
```

Figure 23: HTML Page Structure

Common HTML Tags and Their Uses

A. Text Formatting tags: use to modify the appearance and structure of text on a webpage

Table 5:Text formatting Tags

Tag	Description	Example
<h1> to <h6>	Defines headings from largest (<h1>) to smallest (<h6>).	<h2>Heading</h2>
<p>	Defines a paragraph.	<p>This is a paragraph. </p>
<b>	Makes text bold.	<b>Bold Text</b>
<i>	Italicizes text.	<i>Italic Text</i>
<u>	Underlines text.	<u>Underlined Text</u>
<br>	Inserts a line break.	Line1<br>Line2
<hr>	Creates a horizontal line (divider).	<hr>
<strong>	Defines important (bold) text.	<strong>Important</strong>
<em>	Emphasizes text (italic by default).	<em>Emphasized</em>

B. Color and Style Tags: are used to define the text or background color and to add styles (like size, font, and decoration) to HTML elements.

Table 6: Color and Style Tags

Tag/Attribute	Description	Example
bgcolor	Sets background color of an element (deprecated).	<body bgcolor="yellow"> make the background color yellow <body text= “white”> make all the text on the body white
<style>	add multiple styles (color, size, alignment, etc.) directly inside an element.	<h1 style="color:red; background-color:yellow;">Hello</h1>
<span>	Defines inline text for applying styles.	<span style="color:green;">Green</span>
<div>	Defines a block container (often styled with CSS).	<div style="background-color:lightgrey;">Box</div>

C. List Tags: define the way the list items are displayed on the body

Table 7: List Tags

Tag	Description	Example
`<ul>`	Creates an unordered list (bulleted).	`<ul>` `<li>Apple</li>` `<li>Banana</li> </ul>`
`<ol>`	Creates an ordered list (numbered or alphabetic).	`<ol>` `<li>First</li>` `<li>Second</li> </ol>`
`<li>`	Defines a list item inside `<ul>` or `<ol>`.	`<li>Item</li>`
type (attribute)	Changes the bullet/number style. For `<ul>`: disc, circle, square. For `<ol>`: 1, A, a, I, i.	`<ol type="A">` `<li>Step 1</li>` `<li>Step 2</li> </ol>`

D. Table Tags: are used to inset and format table on the body

Table 8: Table tags

Tag	Description	Example
`<table>`	Defines a table.	`<table> ... </table>`
`<tr>`	Defines a table row.	`<tr> ... </tr>`
`<td>`	Defines a table cell (data).	`<td>Data</td>`
`<th>`	Defines a header cell (bold, centered by default).	`<th>Header</th>`
`<caption>`	Defines a table caption/title.	`<caption>Student Data</caption>`

E. Links and Images HTML Tags: are used to add hyperlinks and multimedia components (image, audio, video) on the body

Table 9: Link and Image Tags

Tag	Description	Example
`<a>`	Hyperlink (links to another page).	`<a href="https://example.com">Visit Site</a>`
target	Defines how a link opens (_blank = new tab).	`<a href="page.html" target="_blank">Open</a>`

<img>	Displays an image.	<img src="image.jpg" alt="My Image" width="200">
alt	provides alternative text for an image.	<img src="logo.png" alt="Logo">
<video>	Embeds video.	<video controls><source src="video.mp4" type="video/mp4"></video>
<audio>	Embeds audio.	<audio controls><source src="song.mp3" type="audio/mpeg"></audio>

F. **Comment Tags:** are used to add notes in the HTML code that are not displayed on the webpage.

Table 10: Comment

Tag	**Description**	
<!-- -->	Single-line comment: a comment that occupies one line. Useful for short notes	<! -- This is a single-line comment -->
	Multi-line comment: a comment that spans multiple lines. Useful for explaining a section of code.	<! -- This is a multi-line comment. It can be used to explain multiple lines of code or a section of the webpage. -->

Each group of HTML tags is used to **structure and style a webpage**.

1.3 Creating a Simple Webpage

Steps to Create a Simple Webpage:

Step 1: Open any Text Editor (Notepad, Notepad++, or VS Code).

Step 2: Write the Basic HTML Structure

```
<html>
<head>
  <title>My First Webpage</title>
</head>
<body>
</body>
</html>
```

Add some text

```
<h1>Hello, World!</h1>
  <p>Welcome to my very first webpage</p>
  <h2>About Me</h2>
  <p>My name is selam. I am learning how to design web pages.</p>
```

Add Colors and Styles

```
<p style="color:blue; background-color:yellow;">

This is a simple webpage I created myself! </p>
```

Add a List

```
<h2>My Favorite Fruits:</h2>
<ul>
  <li>Apple</li>
  <li>Banana</li>
  <li>Mango</li>
</ul>
```

```
<h2>My Favorite BOOK:</h2>
<ol type=i>
  <li>HTML</li>
  <li>JS</li>
  <li>CSS</li>
</ol>
```

Add a Table

```
<h2>My Study Schedule</h2>
<table border="1" style="border-collapse:collapse;">
  <tr style="background-color:lightblue;">
    <th>Day</th>
    <th>Subject</th>
  </tr>
  <tr>
    <td>Monday</td>
    <td>python</td>
  </tr>
  <tr>
    <td>Tuesday</td>
    <td>HTML</td>
  </tr>
</table>
```

Add Links and Images

```
<img src="H2.JPG" alt="My Photo" width=70 height=70 >
<p>Visit <a href="https://www.w3schools.com/"
target="_blank">W3Schools</a>   for tutorials</p>
```

Step 3: Save your file as yourfilename.html (eg.myfrist.html)

Step 4: Open the saved file in any browser (like Chrome, Firefox, or Edge) to see the result.

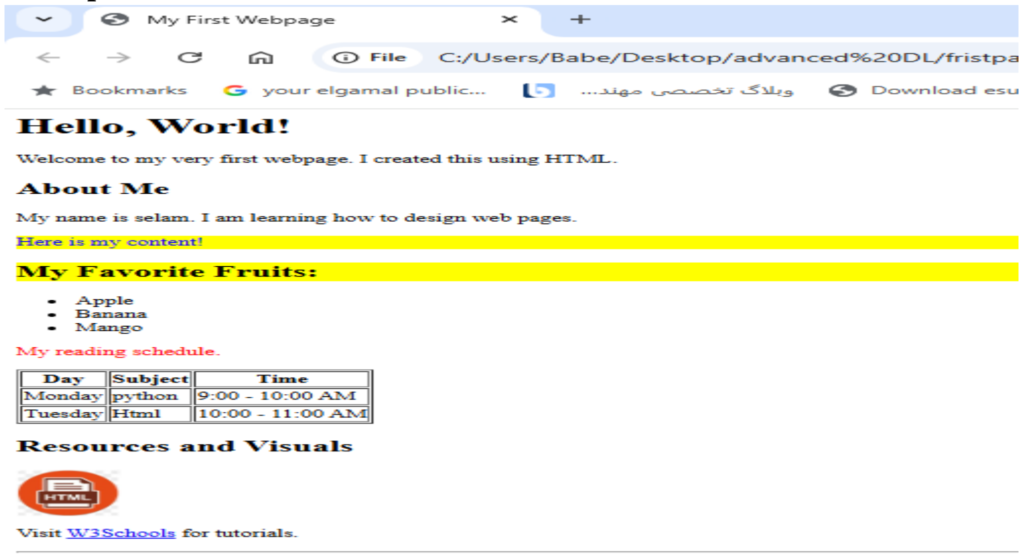
Now, let's combine them all into a single, complete structure.

Fristpage.html

```
<html>
<head> <title>My First Webpage</title></head>
<body>
 <!-- Text Formatting -->
  <h1>Hello, World!</h1>
  <p>Welcome to my very first webpage. I created this using HTML.</p>
  <h2>About Me</h2>
  <p>My name is selam. I am learning how to design web pages.</p>
        <!--Text and Background Colors -->
  <p style="color:blue; background-color:yellow;">Here is my content! </p>
   <p style="background-color: yellow;">My reading schedule. </p>
   <!—list-->
      <h2>My Favorite Fruits:</h2>
<ul>
  <li>Apple</li>
  <li>Banana</li>
  <li>Mango</li>
</ul>
  <!--Table -->
<table border="1">
  <tr>
    <th>Day</th>
    <th>Subject</th>
    <th>Time</th>
  </tr>
  <tr>
    <td>Monday</td>
    <td>python</td>
    <td>9:00 - 10:00 AM</td>
  </tr>
  <tr>
    <td>Tuesday</td>
    <td>Html</td>
    <td>10:00 - 11:00 AM</td>
  </tr>
</table>
  <h2>Resources and Visuals</h2>
 <!-- Links and Images -->
  <img src="H2.JPG" alt="My Photo" width=70 height=70 >
  <p>Visit <a href="https://www.w3schools.com/" target="_blank">W3Schools</a>   for
tutorials.</p>
```

```
    <hr>
</body> </html>
```

Expected output



Hello, World!

Welcome to my very first webpage. I created this using HTML.

About Me

My name is selam. I am learning how to design web pages.

Here is my content!

My Favorite Fruits:

- Apple
- Banana
- Mango

My reading schedule.

Day	Subject	Time
Monday	python	9:00 - 10:00 AM
Tuesday	Html	10:00 - 11:00 AM

Resources and Visuals

Visit W3Schools for tutorials.

Figure 24: Output of first html

1.4 HTML form

An HTML form is a container that holds input fields, buttons, and other elements used to collect user data. It specifies how data is sent to a server with the action and method attributes, enabling interaction between users and the server.

- Form Syntax:

```
<form action="" method="">
  <!-- form fields go here -->
</form>
```

- **action**: Specifies where to send the data (e.g., a PHP script or another page).
- **method** tells the form how to send the data:
  - **GET**: sends data in the URL (good for search forms).
  - **POST**: sends data hidden in the request body (good for sensitive info like login or registration).

Elements of Form

1. **<INPUT>:** used to create different kinds of fields where users can enter or select information. You can customize it using the following attributes

- **type**: Defines the kind of input (e.g., text, password, checkbox).
- **name**: Identifies the input when the form is submitted.
- **value**: Sets the default text shown in the field.
- **size**: Sets field width (in characters).
- **maxlength**: sets the maximum number of characters the user can type.
- **placeholder**: Displays hint text until user types.

Table 11: Attributes of <INPUT> tag

Type	Purpose	Example
text	Single-line text input	*<input type="text" name="username" placeholder="Enter name">*
password	Hidden text input (for passwords)	*<input type="password" name="pass" placeholder="Enter password">*
radio	Select one option from a group	*<input type="radio" name="gender" value="male"> Male*
checkbox	Select multiple options	*<input type="checkbox" name="hobby" value="reading"> Reading*
hidden	Stores invisible data	*<input type="hidden" name="userID" value="123">*
image	Clickable image button	*<input type="image" src="submit.png" alt="Submit">*
submit	Sends form data to server	*<input type="submit" value="Submit">*
reset	Clears all input fields	*<input type="reset" value="Reset">*

Here's how you can use **all these attributes** together in an <input> field and what each does:

2. **<textarea> … </textarea>**
   - Can be used to accommodate multiple text lines in a box.
   - Attributes are:
     - name: name of the field.
     - rows: number of lines of text that can fit into the box.
     - cols: width of the text area on the screen.

Example: *<textarea name="message" rows="5" cols="40"></textarea>*

3. **<select> ……</select>**

- used to create a **drop-down list** in a form. It allows users to pick one (or sometimes multiple) options from a list. Each choice inside the drop-down is defined with the <option> tag.
- Attributes are:
  - **Name**: gives the drop-down a name, so the selected value can be sent to the server.
  - **Size:** (optional) Defines how many options are visible without scrolling.
  - **Multiple:** (optional) Allows users to select more than one option.

4. **<label>:** Labels an input for better accessibility.

   Example *<label >name:</label><input type="text" id="username" name="username">*

5. **<button>:** Creates a clickable button (can submit or reset forms).

   Example *<button type="submit">Submit</button>*

   <button type="reset">Reset</button>

6. **<fieldset> and <legend>:** Group related fields and give a title.

   Example

```
 <fieldset>
<legend>Personal Info</legend>
<input type="text" name="firstname" placeholder="First Name">
<input type="text" name="lastname" placeholder="Last Name">
</fieldset>
```

Putting Form Elements Together

Once we understand each element of a form, the next step is to combine them into a complete form.

Practical Example

step1: open any text editor

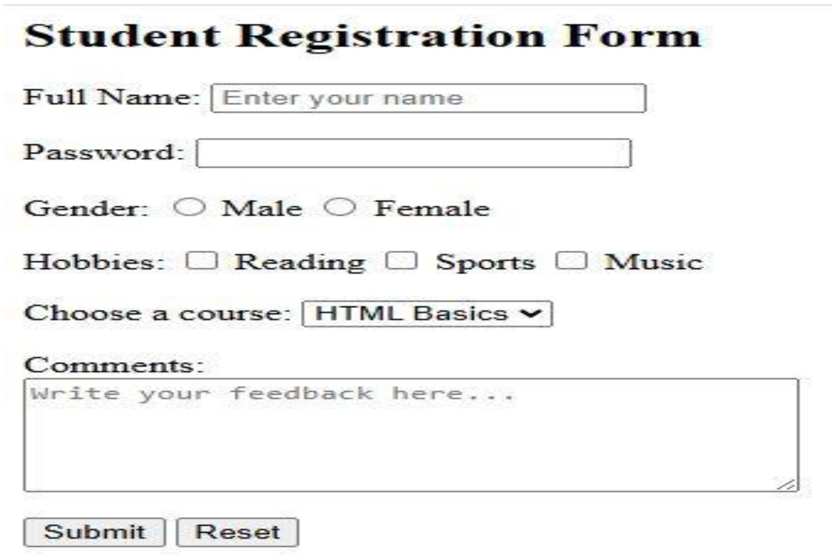
step2: write the following code

```
<html>
<head>
 <title>form </title>
</head>
<body>
<h2>Student Registration Form</h2>
<form action="" method="post">
  <!-- Text input -->
  <label >Full Name:</label>
  <input type="text" name="fullname" placeholder="Enter your name" >
  <br><br>
  <!-- Password -->
  <label >Password:</label>
  <input type="password" name="password" required>
  <br><br>
  <!-- Radio -->
  Gender:
  <input type="radio" name="gender" value="male"> Male
  <input type="radio" name="gender" value="female"> Female
  <br><br>
  <!-- Checkbox -->
  Hobbies:
  <input type="checkbox" name="hobbies" value="reading"> Reading
  <input type="checkbox" name="hobbies" value="sports"> Sports
  <input type="checkbox" name="hobbies" value="music"> Music
  <br><br>
  <!-- Select -->
  <label >Choose a course:</label>
  <select id="course" name="course">
    <option value="html">HTML Basics</option>
    <option value="css">CSS Styling</option>
    <option value="js">JavaScript</option>
  </select>
  <br><br>
  <!-- Textarea -->
  <label>Comments:</label><br>
  <textarea name="comments" rows="5" cols="40" placeholder="Write your feedback
here..."></textarea>
  <br><br>
  <!-- Buttons -->
  <input type="submit" value="Submit">
  <input type="reset" value="Reset">
</form>
</body>
</html>
```

step3: save

step4: run in any browser

The expected output



Student Registration Form

Full Name: Enter your name

Password:

Gender: ○ Male ○ Female

Hobbies: ☐ Reading ☐ Sports ☐ Music

Choose a course: HTML Basics

Comments:
Write your feedback here...

Submit Reset

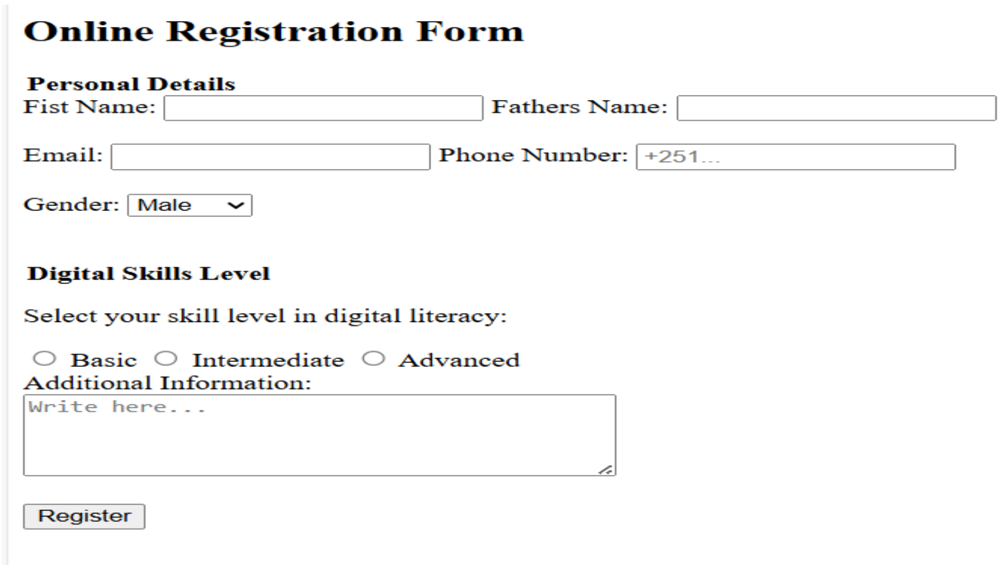
Figure 25: Output of registration form

Exercise:

Task 1: Design your webpage that includes the following:

Introduces yourself with a short paragraph including **bold**, *italic*, and underlined text. Add both an unordered (bullet) and an ordered (numbered) list, a table showing your weekly schedule, a clickable link that opens in a new tab, and an image with a meaningful alt description.

Task2: Create the Following registration Form using



Online Registration Form

Personal Details

Fist Name: Fathers Name:

Email: Phone Number: +251...

Gender: Male

Digital Skills Level

Select your skill level in digital literacy:

○ Basic ○ Intermediate ○ Advanced

Additional Information:
Write here...

Register

Figure 26: registration form for exercise

Summary

In this session, learners have gained a foundational understanding of web development and the role of HTML in building webpages. They have learned how to structure content using common HTML tags, create simple webpages, and implement basic HTML forms to collect user input. With this knowledge, learners are now equipped to start designing and building their own interactive webpages, forming a strong base for further learning in web development.

Review question

1. Which of the following is the basic building block of a web page?
   A. HTML
   B. CSS
   C. JavaScript
   D. PHP
2. Which HTML tag is used to create a hyperlink?
   A. <p>
   B. <a>
   C. <img>
   D. <h1>
3. What does the <img> tag require to display an image?
   A. href
   B. src
   C. alt only
   D. title
4. Which tag contains metadata like the page title and styles?
   A. <body>
   B. <html>
   C. <head>
   D. <title>
5. Which is the correct order of basic structure in an HTML file?
   A. <html>, <body>, <head>
   B. <!DOCTYPE>, <head>, <body>, <html>
   C. <!DOCTYPE>, <html>, <head>, <body>
   D. <head>, <html>, <body>

Session 2: Cascading Style Sheet

Introduction

Dear Learners, in this session, you'll dive into Cascading Style Sheets (CSS) the language that brings style, color, and design to your web pages!

You are already familiar with HTML, which provides the structure of web pages. CSS, which stands for Cascading Style Sheets, is used to style those pages. It is a language that controls the layout, colors, fonts, spacing, and other visual aspects of a webpage. In this section, we will cover the basics of CSS and how it works to make your web pages look attractive and well-organized.

Learning Objectives

By the end of this session, learners will be able to:

- Explain the basics of CSS and its importance.
- Apply CSS to HTML documents.
- Identify and use various types of CSS selectors.

Contents outline

2.1 Basics of CSS

What is CSS?

- CSS (Cascading Style Sheets) is a language used to control the appearance and layout of web pages. For example, CSS covers Fonts, colors, margins, lines, height, width, background images and overall design.
- CSS defines how HTML elements are to be displayed.
- CSS separates content from presentation and making websites easier to manage and update.
- With CSS, you can style multiple pages consistently, save time by writing styles once, and quickly make changes across an entire website.

Basic Syntax of CSS

CSS rule contains two main parts: the Selector and declarations.

```
Selector {property: value; property2: value2; etc...}
```

Selector

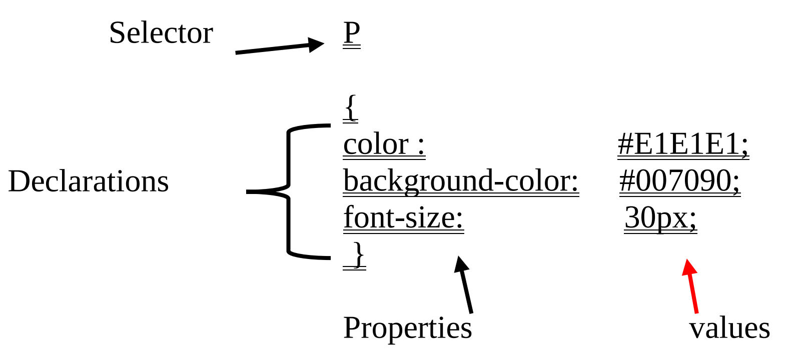
- Determines which HTML element(s) the style will apply to
- Can be an HTML element (p, h1, etc.), an ID (#idName), or a class (.className).

Declaration: Written inside curly braces { } and contains one or more property-value pairs separated by semicolons.

- Property: The attribute you want to change (e.g., color, font-size).
- Value: The setting you want the property to have (e.g., red, 16px).

Note: The property is followed by a colon (:) and the value is followed by a semicolon (;)

CSS rule with one selector and three declarations



```
P
{
color :             #E1E1E1;
background-color:   #007090;
font-size:           30px;
 }
```

- Comments: Used to add notes within your CSS code for readability. It doesn't affect the code itself or show up in the browser. Here is how you write comments

```
/*  for single line comments and  */

/* For multiple Liners of comments

*/
```

How to Apply CSS to HTML

We can apply styles in either of the following approaches to our documents:

- Inline CSS: Styling applied directly within an HTML tag using the style attribute.
- Embedded / Internal CSS: CSS placed within a <style> tag in the <head> section of an HTML document.
- External CSS: Linking a separate CSS file to an HTML document using the <link> tag.

1. **Inline Stylesheet**

- Add style information to HTML elements using style attribute

- useful for applying a unique style to a single HTML element

The following examples applies inline styles to elements to alter their alignment and color.

```
<html>
<body>
<h1 style="color:blue;text-align:center;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
</body>
</html>
```

When to Use Inline CSS

- For quick fixes and small changes that don’t require a separate CSS file.
- When you need to override other styles for a particular element.
- If you’re working on emails or HTML-based applications where external CSS is not supported.

2. **Internal / Embedded Styles**

- Add style information at the head section of a web page using <style>…</style> tag
- Styles placed in the head apply to matching elements in the entire document not just to a

```
<html>
<head>
<style>
  body {
    background-color: yellow;
      }
  h1 {     color: maroon; margin-
left: 40px; }
P { Color: green;  }
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph. </p>
</body>
</html>
```

Look, the style is written in the head section of the HTML document

The output will be:

This is a heading

This is a paragraph.

When to Use Internal CSS

- When designing a single-page website.
- If you need better control over styling than inline CSS.
- For small to medium-sized projects where external CSS might be unnecessary.

3. **External Styles sheets**

The most powerful way and a convenient way to create a document with a uniform theme. With external style sheets separate documents that contain only one CSS rules
Here’s how it works:

1. Write your CSS in a separate file and save it with a **".css"** file extension.
2. Link this CSS file to your HTML page in the <head> section using **<link>** tag
3. Make sure the CSS file is in the same folder as your HTML file. If it’s somewhere else, you’ll need to give the full path or URL.

Mystyle.css	Page1.html
body {	*<!DOCTYPE html>*
background-	*<html>*
color: yellow;	*<head>*
}	*<link rel="stylesheet" type=text/css href="mystyle.css">*
h1 { color: maroon;	*</head>*
margin-left: 40px; }	*<body>*
P { Color: green; }	*<h1>This is a heading</h1>*
	<p>This is a paragraph. </p>
	</body>
	</html>

When to Use External CSS

- For large-scale projects where multiple pages share a common design.
- When maintainability and scalability are priorities.
- To improve website performance and load times using CSS caching.

Multiple Style Sheets

Sometimes the same element (selector) may have styles defined in more than one style sheet. In this case, the browser decides which style to apply based on precedence (priority).

1) Same Selector in Different Style Sheets

If the same selector is defined in multiple style sheets, the style from the last read style sheet will be applied.

2) Order of Precedence (Priority Levels)

When styles come from different sources, the browser applies them in the following order (from highest priority to lowest):

1. Inline Styles: written directly inside an element (highest priority).
2. Internal Styles: written inside the <style> tag in the HTML <head>.
3. External Styles: written in a separate .css file and linked to the HTML page.
4. Browser Default: if no style is given, the browser applies its own default style.

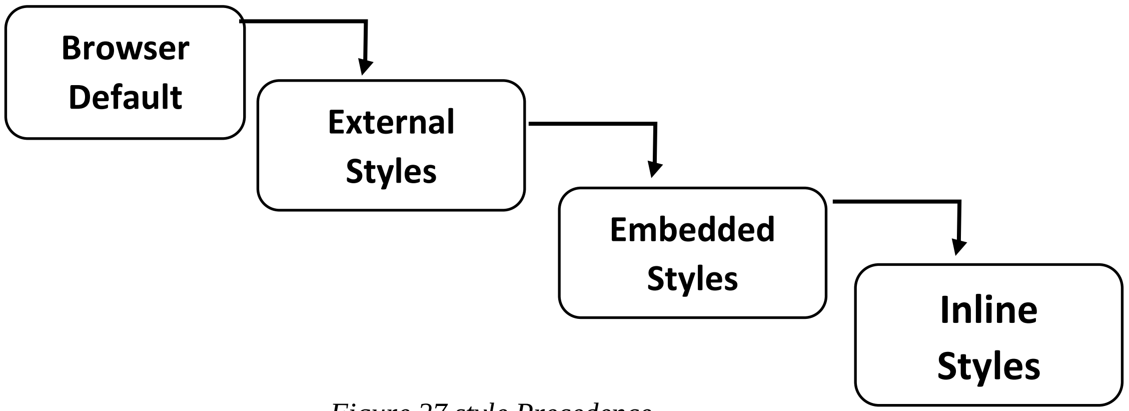


Figure 27 style Precedence

2.2 Types of Selectors

CSS selectors allow us to choose which HTML elements to style. Some selectors are simple, while others can be more specific or complex.

Here are some types of selectors.

a) **Element selectors**:
- Select elements by tag name (e.g., p, h1)
- Also called *tag selector* because it directly targets an HTML tag.

Example:

```
p { color: blue } /* styles all <p> paragraphs in blue */
h1, h2, h3, h4 { font-style: italic; } /* applies italic style to all headings */
```

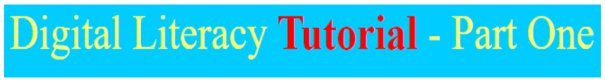
When you list multiple elements separated by commas, it's called a grouping selector.

b) **Descendant Selectors:**
- Used when tags are nested (one element inside another).
- Descendant selector matches an element that is inside (a child or deeper level) another element**.**

- Written as two selectors separated by a space**.**

In the following example only bold text inside a paragraph will be red.

```
<html>
<head><style>
P  b { color: red; }
p{color:#ff6; background-color: #00ccff; font-
size:30px;}
</style> </head>
<body>
<p>Digital <b>Literacy </b> - Part One</p>
</body> </html>
```



Digital Literacy Tutorial - Part One

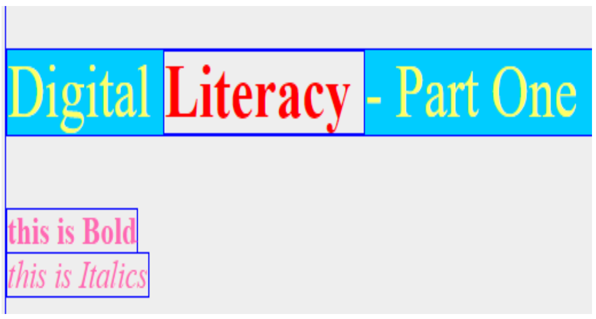
Descendant selectors are useful to target specific elements without adding extra classes or IDs, especially when working inside a particular container or section.

c) **Universal selector** (*)

- Also called a wildcard selector.
- It matches every element on the web page.
- This means the style you apply will affect all HTML elements, unless overridden by more specific selectors.

For example, if we wanted every element to have a solid 1px-wide border and hot pink color, we would use the following CSS rule:

```
<html>
   <head>
   <style>
P  b { color:red; }
p{color:#ff6; background-color: #00ccff; font-
size:30px;}
* {
  color: hotpink;
  background-color: #eee;
  border:1px solid blue;
}
</style>
</head>
<body bgcolor=#0090cc>
   <h1 style="text-align: center;">Welcome
Habesha Digital Literacy</h3>
```



Digital Literacy - Part One

this is Bold
this is Italics

<p>Digital <b>Literacy </b> - Part One</p> *<b>this is Bold</b></br>* *<i>this is Italics</i>* *</body></html>*	

d) **Class selector**

- A class selector in CSS starts with a dot (**.**)
- It is case-sensitive, so**.** Header and. header would be considered different classes.
- The class selector applies styles to all elements that have that class in the HTML document
- An HTML element can have one or more classes listed in its class attribute, separated by spaces.

Example:

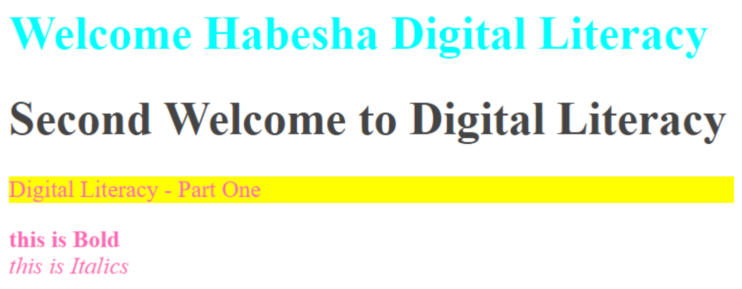
if the style is:	The HTML element to call that class will be
.my-class { color: #FF0000; }	<p class="my-class"></p>

Any element that has the class applied to it will get colored red:

e) **ID Selector**

- An ID selector in CSS starts with a **#** instead of a dot.
- It works similarly to a class selector but with some key differences:
  - Uniqueness: An ID can be used only once per page.
  - Single assignment: An element can have only one ID.
- You can use an ID selector to target a specific element with that ID.
- You can also combine it with a type selector to style an element only if both the element type and ID match.
- ID names are case-sensitive and must follow naming rules (cannot contain spaces, start with a number, or use special characters except - and _)

```
<html>
 <head>
 <style>
#My-id {
          background-color: yellow;
           }
h1#heading {
         color: cyan;
       }
</style>
</head>
<body >
    <h1 id="heading">Welcome
Habesha Digital Literacy</h1>
            <h1 >Second Welcome to
Digital Literacy</h1>
    <p id=My-id>Digital Literacy  -
Part One</p>
            <b>this is Bold</b></br>
            <i>this is Italics</i>
</body></html>
```



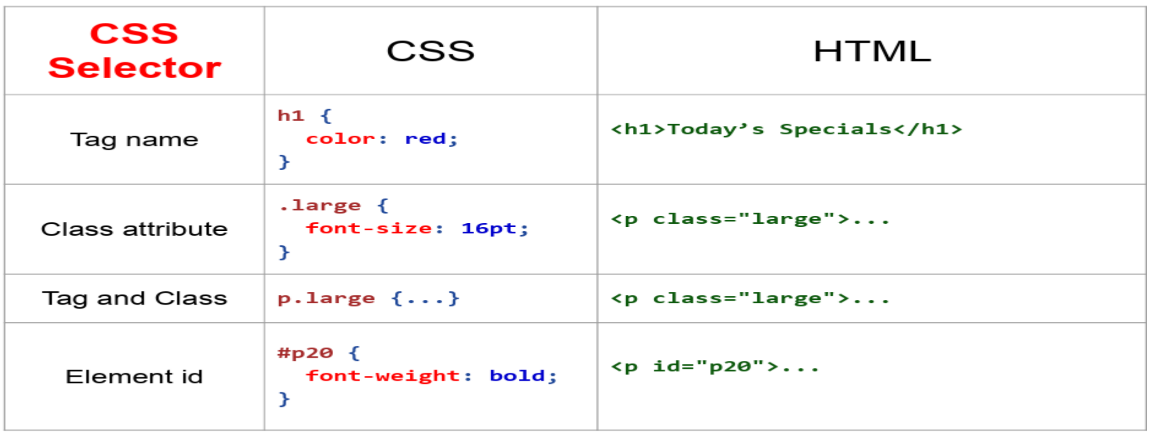
Welcome Habesha Digital Literacy

Second Welcome to Digital Literacy

Digital Literacy - Part One

this is Bold
this is Italics

Summarized CSS selectors



CSS Selector	CSS	HTML
Tag name	`h1 { color: red; }`	`<h1>Today's Specials</h1>`
Class attribute	`.large { font-size: 16pt; }`	`<p class="large">...`
Tag and Class	`p.large {...}`	`<p class="large">...`
Element id	`#p20 { font-weight: bold; }`	`<p id="p20">...`

2.3 CSS Basic Properties

Here ae some basic CSS properties to work with

Text Properties

Table 12: CSS properties

Property	Description	Values
Color	Sets the color of a text	RGB, hex, keyword

line-height	Sets the distance between lines	normal, *number, length, %*
letter-spacing	Increase or decrease the space between characters	normal, *length*
text-align	Aligns the text in an element	left, right, center, justify
text-decoration	Adds decoration to text	none, underline, overline, line-through
text-indent	Indents the first line of text in an element	*length, %*
text-transform	Controls the letters in an element	none, capitalize, uppercase, lowercase

List Properties

Property	Description	Values
list-style	Sets all the properties for a list in one declaration	*list-style-type, list-style-position, list-style-image,* inherit
list-style-image	Specifies an image as the list-item marker	URL, none, inherit
list-style-position	Specifies where to place the list-item marker	inside, outside, inherit
list-style-type	Specifies the type of list-item marker	none, disc, circle, square, decimal, decimal-leading-zero, armenian, georgian, lower-alpha, upper-alpha, lower-greek, lower-latin, upper-latin, lower-roman, upper-roman, inherit

Border Properties

Property	Description	Values
border	Sets all the border properties in one declaration	*border-width, border-style, border-color*
border-bottom	Sets all the bottom border properties in one declaration	*border-bottom-width, border-bottom-style, border-bottom-color*
border-bottom-color	Sets the color of the bottom border	*border-color*
border-bottom-style	Sets the style of the bottom border	*border-style*
border-bottom-width	Sets the width of the bottom border	*border-width*
border-color	Sets the color of the four borders	*color_name, hex_number, rgb_number,* transparent, inherit
border-left	Sets all the left border properties in one declaration	*border-left-width, border-left-style, border-left-color*
border-left-color	Sets the color of the left border	*border-color*

border-left-style	Sets the style of the left border	*border-style*
border-left-width	Sets the width of the left border	*border-width*
border-right	Sets all the right border properties in one declaration	*border-right-width, border-right-style, border-right-color*
border-right-color	Sets the color of the right border	*border-color*
border-right-style	Sets the style of the right border	*border-style*
border-right-width	Sets the width of the right border	*border-width*
border-style	Sets the style of the four borders	none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, inherit
border-top	Sets all the top border properties in one declaration	*border-top-width, border-top-style, border-top-color*
border-top-color	Sets the color of the top border	*border-color*
border-top-style	Sets the style of the top border	*border-style*
border-top-width	Sets the width of the top border	*border-width*
border-width	Sets the width of the four borders	thin, medium, thick, *length*, inherit

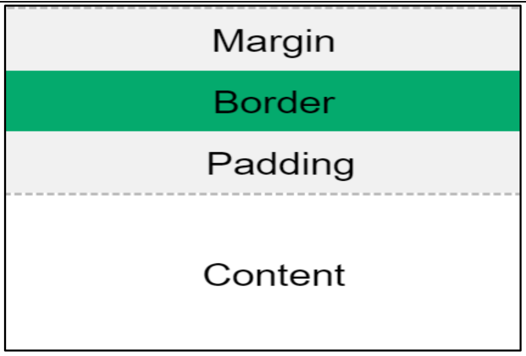
Font Properties

Property	Description	Values
font	Sets all the font properties in one declaration	*font-style, font-variant, font-weight, font-size/line-height, font-family,* caption, icon, menu, message-box, small-caption, status-bar, inherit
font-family	Specifies the font family for text	*family-name, generic-family,* inherit
font-size	Specifies the font size of text	xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger, *length, %,* inherit
font-style	Specifies the font style for text	normal, italic, oblique, inherit
font-variant	Specifies whether or not a text should be displayed in a small-caps font	normal, small-caps, inherit
font-weight	Specifies the weight of a font	normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900,

The Box Model:

The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins.

- Content: The actual content of the element.
- Padding: Space between the content and the border.
- Border: A line surrounding the padding and content.
- Margin: Space outside the border, separating the element from other elements



You can write your styles as follows

```
p
{
  width: 320px;
  height: 50px;
  padding: 10px;
  border: 5px solid gray;
  margin-left: 10;
  margin-top:30px;
}
```

CSS Properties Control many style properties of an element:

- Coloring
- Size
- Position
- Visibility
- Many more

Colors and Backgrounds:

- color: Sets the foreground color of an element.
- background-color: Sets the background color of an element.
- background-image: Adds an image as the background.

Display and Positioning:

- display property: Determines how an element is rendered and interacts with others (e.g., block, inline, inline-block, none).
- position property: Controls an element's placement on the page (e.g., static, relative, absolute, fixed, sticky).
- CSS units: Define lengths in absolute or relative terms for styling elements.

Absolute Units

- Fixed-size units that don't adjust to screen size.
- Best for print or fixed-size outputs, not responsive screens.

Unit	Description	How to use
Cm	Centimeters	P{width:100cm;}
Mm	Millimeters	P{width:100mm;}
In	Inches	P{width:100in;}
Px	Pixels	P{width:100px;}
Pt	Points	P{width:100pt;}
Pc	Picas	P{width:100pc;}

Relative unit

- Define length in relation to another property (like font size).
- More flexible and better for responsive design across devices. Some of the relative length units are:

Unit	Description	How to use
Em	my parent element's font-size	P{width:2em;}
%	Relative to the parent element	P{width:150%;}

Exercise

For your webpage (created in session 1), apply internal CSS to style a paragraph by changing the font family, text color, and line spacing. Then, apply styles using different selectors (element, class, and ID).

Summary

In this session, learners explored the fundamentals of CSS and its role in web design. They learned how to apply CSS to HTML, use different types of selectors, and style text, backgrounds, and layouts using basic CSS properties. With this knowledge, learners are now equipped to enhance the appearance of their webpages and create visually appealing, well-structured designs.

Review Questions

1. Which type of CSS is coded in the body of the web page as an attribute of an HTML tag?
   A. Embedded
   B. Internal
   C. Inline

   D. Imported

2. Which of the following is the CSS property used to set the background color?
   A. Bgcolor
   B. Bcolor
   C. Color
   D. Background-color
3. Which of the following describes two components of CSS rules?
   A. selectors and declarations
   B. properties and declarations
   C. selectors and attributes
   D. property and value
4. Which of the following configures a CSS class called news with red text (#FF0000) and light gray background (#EAEAEA)?
   A. news { color: #FF0000; background-color: #EAEAEA; }
   B. .news { color: #FF0000; background-color: #EAEAEA; }
   C. .news { text: #FF0000; background-color: #EAEAEA; }
   D. #news {color: #FF0000; background-color: #EAEAEA; }
5. Which of the following tags is used to embed CSS in an HTML page?
   A. <css>
   B. <!DOCTYPE html
   C. <script>
   D. <style>
6. Which of the following CSS property is used to make the text bold?
   A. text-decoration: bold
   B. font-weight: bold
   C. font-style: bold
   D. text-align: bold
7. Which of the following CSS property sets the font size of text?
   A. font-size
   B. text-size
   C. text

D. size

8. Which of the following is not the property of the CSS box model?
    A. margin
    B. color
    C. width
    D. height
9. Which of the following CSS properties is used to specify table borders in CSS?
    A. table: border
    B. Table
    C. Border
    D. none of the mentioned
10. Which of the following is the correct syntax to link an external style sheet in the HTML file?
    A. <link rel=”stylesheet” href=”style.css” />
    B. <link rel=”stylesheet” src=”style.css” />
    C. <style rel=”stylesheet” src=”style.css” />
    D. <style rel=”stylesheet” link=”style.css” />

Session 3: JavaScript

Introduction

Dear Learners, in this session, you'll discover how JavaScript makes web pages interactive, responsive, and engaging for users!

JavaScript is a powerful programming language used to add interactivity and dynamic behavior to webpages. While HTML provides the structure and CSS handles the design, JavaScript brings webpages to life by enabling animations, form validation, content updates, and user interactions. In this session, you will explore the basics of JavaScript, how to write and run scripts, and how JavaScript interacts with the Document Object Model (DOM) to control and update webpage content. You will also cover events, event handling, and working with forms to collect and validate user input.

Learning Objectives

By the end of this session, learners will be able to:

- Explain JavaScript's role in interactive webpages.
- Write and run basic JavaScript scripts.
- Manipulate the DOM and handle events.
- Use JavaScript for form validation and dynamic content.

Content Outline

3.1 Introduction to Java script

JavaScript is a programming language used to make webpages interactive and dynamic. While HTML provides the structure of a webpage and CSS controls its appearance, JavaScript allows webpages to respond to user actions, update content, create animations, and perform many other interactive tasks.

Why JavaScript is Important

JavaScript adds interactivity and dynamic behavior to websites. For example:

- Showing alerts or messages when a user clicks a button.
- Validating forms before submission.
- Updating webpage content without reloading the page.
- Creating interactive menus, sliders, and animations.

Key Roles of JavaScript in Web Development:

- Interactivity: Makes webpages respond to user actions (buttons, forms, menus).
- Dynamic Updates: Can change content or styles instantly.
- Form Validation: Ensures users enter correct information before submitting.
- Animations & Effects: Adds visual enhancements like sliders and transitions.
- Communication with Servers: Helps webpages send and receive data (e.g., in online forms or apps).

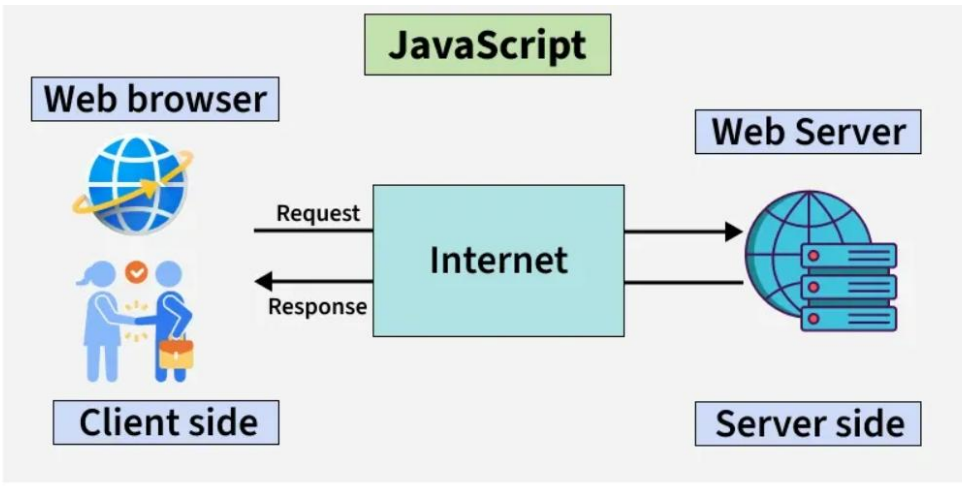


Figure 28 JavaScript interaction

JavaScript is a flexible language that can be used both on the client-side (in the browser) and the server-side (on a server), allowing developers to create complete web applications.

Client-Side:

- Runs in the user's browser.
- Controls the webpage and its DOM (Document Object Model), which is the structure of HTML elements on the page.
- Handles user interactions, such as clicks, typing in forms, or moving the mouse.

Examples: showing alerts, validating forms, updating content without reloading the page.

Server-Side:

- Runs on a server using platforms like Node.js.
- Interacts with databases to store or retrieve data.
- Manipulates files and generates dynamic responses for webpages.

Examples: saving form submissions, fetching product data for an online store, or sending emails.

JavaScript Syntax

- A statement is a single instruction in JavaScript.
- Most statements end with a semicolon **(;)**
- Comments are used to explain code and are ignored by the browser:

```
// This is a single-line comment

/*
This is a
multi-line comment
*/
```

Variables and Constants

- Variables are used to store data that can change, while constants store data that should not change.
- Declare variables using the keywords **var**, **let**, or **const**.
- When naming variables, keep these rules in mind:
    - Names can contain letters, digits, underscores (_), and dollar signs ($).
    - Names must begin with a letter, $ (dollar sign), or _ (underscore).
    - Variable names are case-sensitive, so age and Age are different variables.
    - Reserved words (like var, let, if, for) cannot be used as variable names.

Example

```
let name=”selam”; 'let' declares a variable
const pi=3.14 // 'const' declares a constant
```

Data Types

- JavaScript has different types of data: number, string, Boolean, Null, Undefined, object, Array

Example

```
let age = 25;         // Number
const name = "Betelhem"; // String
var isStudent = true;  // Boolean
let selectedItem = null; // Null
let data;              // Undefined
let user= {name: "Alice", age: 25}  // Object
let num= [1, 2, 3];
```

Operators

JavaScript has several types of operators: Arithmetic, Assignment, logical, comp

Arithmetic Operators: +, -, *, /, %	Assignment Operators: =, +=, -=

Example:
let x = 10;
let y = 5;
document.write (x + y); //15
document.write (x - y); // 5
*document.write (x * y); //50*
document.write (x / y); // 2
	Example:
	let x = 10; // assigns value 10 to x
	x += 3; // same as x = x + 3
	document.write("x=" +x) ; //x=13
	x-= 4; // same as x = x – 4
	document.write("x=" +x) ;//x=9

Logical Operators: &&, ‖,!	Comparison Operators:<, >, ==,>=…

Example:
document.write (true && false); // false
document.write (true || false); // true
document.write (!true);   // false
	Example:
	document.write (10 > 5); // true
	document.write (10 == 5); // false
	document.write (10 != 5); // true

Output in JavaScript

JavaScript provides several ways to display results:

- alert box: alert() function pops up a message on the screen.
  Example: alert("Hello, this is my first alert!");
- document.write() method lets JavaScript write directly into the page.

Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the developer console

  Example: console.log (5 + 6);

Ways to Add JavaScript to HTML

a) Inline JavaScript

JavaScript code is written inside an HTML element's attribute (like onclick):

Example: <button onclick="alert('Button clicked!')">Click Me</button>

b) Internal/ embedded JavaScript:

- JavaScript can be placed in either the <head> or <body> sections of an HTM.
- JavaScript code must be inserted between <script> and </script> tags.

```
<script type="text/javascript"
        ….
</script>
```

- The placement of the tag depends on when you want the code to run:
  - In <head>: runs before the page content loads.
  - In <body>: runs as the page loads and interacts with page elements.
- The browser executes the script as the page loads, combining the dynamic output with the static HTML content.

c) External JavaScript

- JavaScript can be written in a separate .js file and linked to HTML:
- useful when the same code is used across multiple webpages, making maintenance easier
- To use an external script, put the name of the script file in the src (source) attribute of the <script> tag:

```
<script src="firstscript.js" > …... </script>
```

Example:

my.js	Page1.html
<script> *document.write ("Hello JavaScript file!");*	*<!DOCTYPE html>* *<html>* *<head>* *<script src="my.js"></script>*

alert("External JS is working!"); </script>	*</head> <body> <h1>This is a heading</h1> <p>This is a paragraph. </p> </body> </html>*

Control Flow

Control flow determines **the order in which code runs** based on conditions

- Conditional statements: if, else if, else, switch
- Loops: for, while, do…while
- Break & continue statements

*<script type="text/javascript">
if (grade >= 90) {
  document.write ("Excellent");
} else if (grade >= 75) {
  document.write ("Good");
} else {
  document.write (" Improvement");}
</script>*
	*<script type="text/javascript">
	for (i = 1; i <= 6; i++){
	document.write("<h" + i + ">This is heading " + i);
	  document.write("</h" + i + ">");
	}
	</script>*

*<script type="text/javascript">
let i = 1;
while (i <= 5) {
  document.write (i);
 i++;
}
</script>*
	*<script type="text/javascript">
	for (let i = 1; i <= 5; i++) {
	 if (i === 3) continue; // skips 3
	  document.write (i);
	}
	for (let i = 1; i <= 5; i++) {
	 if (i === 3) break; // stops loop at 3
	  document.write (i);
	}</script>*

Array

JavaScript arrays are used to store multiple values in a single variable. They are an ordered collection of values, all referenced by one variable name.

Array Syntax:

- var array-name = new Array(size);// size is optional
  - Size is the number of the element you want in the array
- To assign value to array elements, use
  - array-name[i] = value; Where i is the index of the array item. The 1st item has an index value of 0. Or
  - var array-name = [item1, item2, ...]; Where i is the index of the array item. The 1st item has an index value of 0.

Array declared with values	Array declared without values

<script>
var DayTxt = ["Mo", "Tu", "We", "Th"];
document.write("<h3>"+" Days of the week are:"+"</h3>");
document.write("<ul>");
for (var i = 0; i < DayTxt.length; i++) {
document.write("<li>"+ DayTxt [i] + "</li>");
}
document.write("</ul>");
</script>
	<script>
	var DayTxt= new Array();
	DayTxt [0]="Mo"; DayTxt [1]="Tu"; DayTxt [3]="We"; DayTxt[4]= "Th";
	document.write("<h3>"+" Days of the week are"+"</h3>");
	document.write("<ul>");
	for (var i = 0; i < DayTxt.length; i++) {
	document.write("<li>"+ DayTxt [i] + "</li>");
	}
	document.write("</ul>");
	</script>

6. Function in JavaScript

- A function in JavaScript is a block of code designed to perform a specific task.
- A function has two main parts: the function definition (where you write the code) and the function call (where you execute the function)

Function definition

- function is defined with the function keyword, followed by a name, followed by parentheses ().
- The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets.

Example

```
function myFunction(p1, p2)
             {
         //code to execute
           return p1 * p2;
}
```

Function call

- Function will execute when "something" invokes (calls) the function
- function can be invoked:
    - When an event occurs (when a user clicks a button)
    - When it is invoked (called) from JavaScript code
    - Automatically (self-invoked)

Placing a Function in an HTML File

- Functions should be defined before they are called.
- A common practice is to place all function definitions in the <head> section of your HTML.
- A function is executed only when called by another JavaScript command on the body
- For larger projects, you can store functions in external .js files and link them to HTML pages.

Example1. Function definition without value	Example2 Function definition with value
<html>	<html>
<head>	<head>
<script type="text/javascript">	<script type="text/javascript">
function sayHello() //function definition	function sayHello(name, age)
{	{
alert("Hello there!");	alert(name + " is " + age + " years old.");
}	}
</script>	</script>
</head>	</head>
<body>	<body>
<p>Click the button to call the function</p>	<p>Click the button to call the function</p>
<form>	<form>
<input type="button" onclick="sayHello()"	<input type="button"
value="Say	onclick="sayHello('Amanuel ', 12)" value="Say
Hello"> //sayhello() is function calling	Hello">
</form>	</form>
<p>Use different text in alert box and then	<p>Use different parameters inside the function
try...</p>	and then try...</p>
</body>	</body>
</html>	</html>

7. Pop up Boxes

Popup boxes can be used to raise an alert, or to get confirmation on any input or to have a kind of input from the users.

JavaScript supports three types of popup boxes.

- Alert box
- Confirm box
- Prompt box

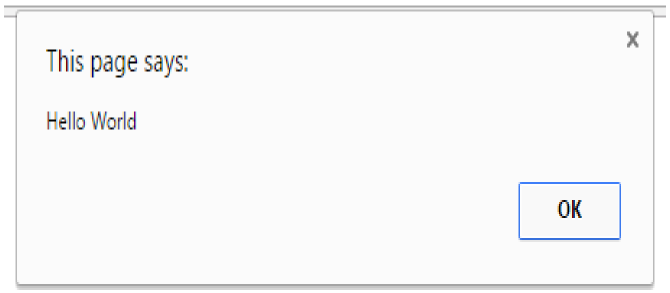
Alert Box

- An **alert box** is used if you want to **make sure** information **comes through** to the **user**.
- When an alert box pops up, the user will **have to click "OK"** to proceed.
- It can be used to display the result of validation.

Example

Code

```
<html>
    <head>
        <title>Alert Box</title>
    </head>
    <body>
        <script>
                alert("Hello World");
        </script>
    </body>
</html>
```



This page says:

Hello World

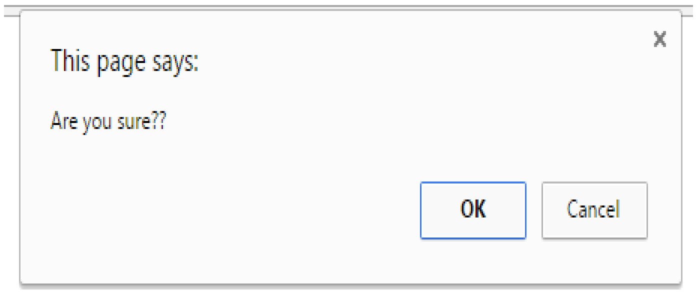
Confirm box

- A **confirm box** is used if you want the user to **accept something**.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed, If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Example

Code

```
<script>
  var a = confirm("Are you sure??");
  if(a==true) {
        alert("User Accepted");
  }
  else  {
        alert("User Cancled");
  }
</script>
```



This page says:

Are you sure??

Prompt box

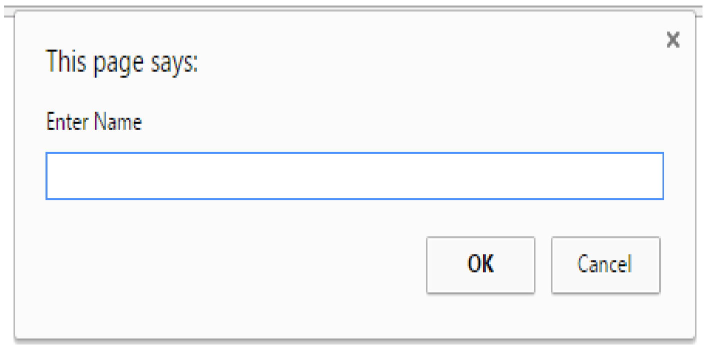
- A prompt box is used if you want the user to input a value.

- When a prompt box pops up, user have to click either "OK" or "Cancel" to proceed, If the user clicks "OK" the box returns the input value, If the user clicks "Cancel" the box returns null.

Example

Code

```
<script>
  var a = prompt("Enter Name");
  alert("User Entered " + a);
</script>
```



This page says:

Enter Name

3.2 Document object model (DOM)

The Document Object Model (DOM) is a way for JavaScript to interact with the structure of a webpage. When a browser loads an HTML document, it creates a tree-like structure of all the elements on the page, such as headings, paragraphs, images, buttons, and links.

Each element in this structure is represented as an object that JavaScript can access and manipulate. You can think of the DOM as a family tree of your webpage: the <html> tag is the root, with <head> and <body> as branches, and every element inside them as child nodes.

DOM is the connection between HTML and JavaScript. Without the DOM, JavaScript would have no way to interact with the page, but with it, you can build dynamic and interactive pages

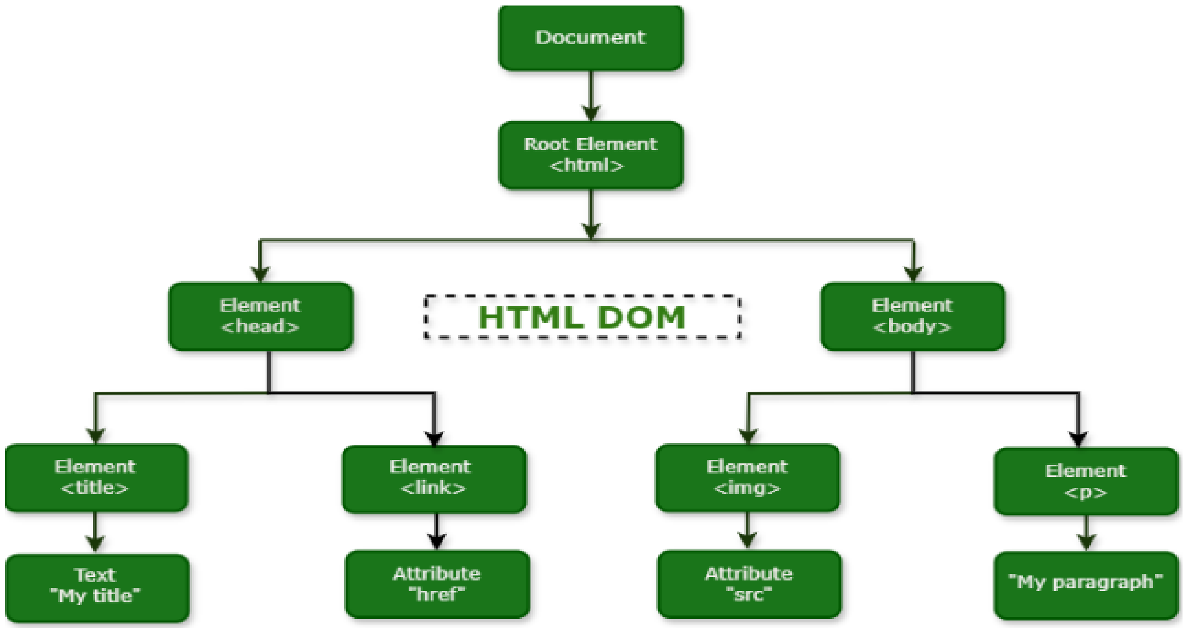


Figure 29: DOM access element structure

To work with the DOM, the first step is learning how to **access elements**. JavaScript provides different methods for access elements.

- **getElementById():** which selects an element based on its unique id attribute

HTML

```
<html>
 <body>
     <input type="text" id="myText">
  </body>
</html>
```

JavaScript

```
<script>
   function myFunction()
   {
  var txt = document.getElementById("myText");
   alert(txt.value);
   }
</script>
```

- **getElementByName():** which selects an element based on its name attribute

HTML

```
<html>
   <body>
  <input type="text" name="myText">
   </body>
</html>
```

JavaScript

```
<script>
function myFunction()
{
  a=document.getElementsByName("myText");
   alert(a.value);
} </script>
```

3.3 Event and Event Handling

Events are things that happen on a webpage, like a user clicking a button, moving their mouse over an element, or submitting a form. JavaScript can be used to "listen" for these events and respond to them, making the page interactive.

Event example

- A **click** event happens when you press a button with the mouse.
- A **mouseover** event happens when you move the mouse pointer over an element.
- A **keypress** event happens when you type something on your keyboard.

To make webpages respond to these events, JavaScript uses event listeners. An event listener uses JavaScript to make a web page interactive by responding to user actions, or events. The most common way to do this is with the addEventListener() method.

Basic syntax:

element.addEventListener('event', functionName);

- element: The HTML element you want to listen to (e.g., a button, a form, or an image).
- 'event': The name of the event you are listening for (e.g., 'click', 'mouseover').

  - functionName: The name of the function to be executed when the event occurs.

Example: A Button Click

Let's say you have a button with the id "myButton" and you want to display a message when a user clicks it.

```
<html>
<head></head>
<body>
<input type="button" id="myButton" value="click me">
<script src="script.js"></script>
</body>
</html>
```

script.js

```
<script>
// 1. Get the button element from the HTML
 const myButton = document.getElementById('myButton');
// 2. Define the function to run when the event occurs function
handleButtonClick()
{
alert('Button was clicked!');
 }
// 3. Attach the event listener to the button
myButton.addEventListener('click', handleButtonClick);

</script >
```

In this example, when the user clicks the button, the event listener runs the function, which shows an alert message.

Event handling allows you to do things like:

- Validate user input in forms before submitting.
- Show or hide extra content when a user interacts.
- Create interactive features like slideshows, games, or animations.

In short, events and event handling are what make webpages feel *alive* they connect the user’s actions with your JavaScript code.

Common Event Types

Event Type	Description
click	Fires when a user clicks on an element. It's the most common event for interactive elements like buttons and links.
mouseover	Occurs when the mouse pointer moves onto an element. It's often used for displaying tooltips or changing an element's appearance.
submit	Fires when a user attempts to submit a form, which is crucial for form validation.
keydown	Occurs when a key is pressed down. It's useful for building features like search bars or keyboard-controlled games.

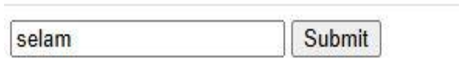
3.4 Working with Forms and User Input

Forms are the way users communicate with a webpage for example, entering their name, email, or password. JavaScript allows us to capture this input, validate it, and give instant feedback to the user.

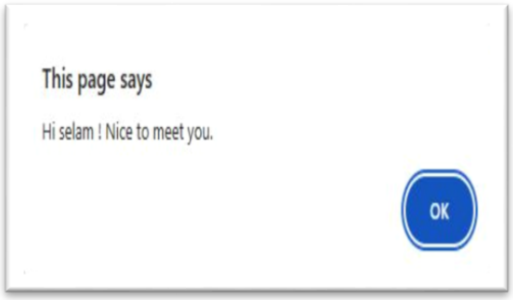
1. **Getting Values from Input Fields**

To read what a user typed into a form, JavaScript uses the **.value** property.

```
<html>
<head>
<script>
 function getValue() {
   let name =
document.getElementById("username").value;
   alert("Hi " + name + "! Nice to meet you.");
  }
</script>
</head>
</body>
<input type="text" id="username" placeholder="Enter
your name">
<button onclick="getValue()">Submit</button>
</body>
</html>
```



When someone types “the name” e.g sealm and clicks submit, they’ll see:
Hi selam ! Nice to meet you.


This page says

Hi selam ! Nice to meet you.

2. **Validating Form Data**

Validation makes sure the data entered is **correct and complete** before it's submitted. Form validation generally performs two functions.

Basic Validation	**Data Format Validation**
• Emptiness • Confirm Password • Length Validation etc.…	the data that is entered must be checked for correct **form** and value • Email Validation • Mobile Number Validation • Enrollment Number Validation etc….

a) **Required Field Validation/Emptiness Validation:** Make sure important fields are not left empty.

HTML	JS

<html>
<head>
<script src="my2.js"> </script>
</head>
  <body>
 <form>
 Name:<input type="text" id="uname">
 FName:<input type="text" id="fname">
 <input type="button" onClick="f()" value="click me">
 </form>
  </body>
</html>
	function f()
	{
	  // Get the form element
	  var a = document.getElementById("uname").value;
	  var b = document.getElementById("fname").value;
	 // checking an input field
	 if(a==""&&b=="")
	  {
	    alert("pleas Enter your info");
	  }
	  else
	  {
	    alert("You entered: " + a+" "+b);
	  }
	}

This example checks: if it's not empty and ask the user to fill the form

b) **Matching Validation:** JavaScript can check if the two inputs match before allowing the form to submit.

passwordForm.html	my2.js
<html> *<head>* *<script src="my2.js"> </script>*	*function f()* *{*

</head>
<body>
<form>
*Password1:<input type="text" id="pass1">
*
Password2:<input type="text" id="pass2">
<input type="button" onClick="f()" value="click-me">
</form>
</body>
</html>
	var p1 = document.getElementById("pass1").value;
	var p2 = document.getElementById("pass2").value;
	
	if(p1==""||p2=="")
	{
	alert("Both fields are required!");
	}
	else if(p1!==p2)
	{
	alert("Passwords do not match!");
	}
	else
	{
	alert("Passwords match!");
	}
	}

This example checks: the password you enter is matched

Matching Validation 2

passwordForm.html	my2.js

<html>
<head>
<script src="my2.js"> </script>
</head>
<body>
<form>
*User name:<input type="text" id="name">
*
password:<input type="text" id="pass">
<input type="button" onClick="f()" value="click-me">
</form>
</body>
</html>
	function f()
	{
	var n = document.getElementById("name").value;
	var p = document.getElementById("pass").value;
	
	if(n==""||p=="")
	{
	alert("Both fields are required!");
	}
	else if(u=="admin" && p=="123")
	{
	alert("valid user and password")
	}
	else
	{
	alert("Invalid user and password");
	}
	}

This example checks: the password and user name you enter is matched with the stored data

c) **Data Validation:** Ensure users enter the right kind of data (e.g., numbers only for age).

age.html

```
<html>
<head>
<script src="age.js"> </script>
</head>
   <body>
     <form>
<label>Age:</label><input type="text"
id="age"><br><br>

       <input type="button" onClick="f()"
value="click-me">
      </form>
   </body>
</html>
```

age.js

```
function f()
{
  var age = document.getElementById("age").value;

   if (age == "") {
     alert("pleas fill the age!");
   } else if (isNaN(age)) {
     alert("Age must be a number");
   } else if(age.length >3){
     alert("age can't be more than 3 characters!");
   }
else {
     alert("your age is"+age);
   }
 }}
```

This example checks: if it’s not empty, if the input is a number, if it’s within a valid range.

Exercise

Create a small registration form for a website that collects a user’s **name, password, and age**. The form should check if the user has filled in all the required fields before submission.

When the user clicks the **Submit** button, your JavaScript code should:

1. Check the inputs if any field is empty, show an alert telling the user to fill it.
2. Validate the password Ensure the length is above 6
3. Validate age Make sure the age is a positive number and above 0.
4. Display a welcome message If everything is filled correctly, show a personalized welcome message like:
   “Welcome, [Name]! Your registration is complete.”

Summary

In this session, we introduced JavaScript, the programming language that makes webpages interactive and dynamic. You learned the basics of JavaScript, including variables, data types, and operators, and explored how to write and run scripts using inline, internal, and external methods.

You also learned about the Document Object Model (DOM**)** and how JavaScript can access and manipulate webpage elements, as well as how to handle events like clicks and keypresses. Finally, we discussed working with forms to collect and validate user input.

With this foundation, you are now equipped to create simple, interactive webpages and are ready to build on these skills to develop more advanced web functionality in the future.

Review Questions

1. Which of the following is the correct way to declare a variable in JavaScript?
   A. variable name = "sara";
   B. let name = "sara";
   C. var name == "sara";
   D. const name;
2. What will the following code output?
   let x = 10;
   x += 5;
   console.log(x);
   A. 5
   B. 10
   C. 15
   D. 20

3. Which of these is used to add interactivity to a webpage?
   A. HTML
   B. CSS
   C. JavaScript
   D. SQL

4. Which event occurs when a user clicks on an HTML element?
   A. Onmouseover
   B. onclick
   C. onkeypress
   D. onchange

5. Which of the following is the correct syntax for a JavaScript function?
   A. function sayHello() { alert("Hello!"); }
   B. func sayHello() { alert("Hello!"); }
   C. function: sayHello() { alert("Hello!"); }
   D. function = sayHello() { alert("Hello!"); }

Session 4: Basic of PHP

Introduction

Dear Learners, in this session, you'll explore the basics of PHP the powerful server-side language that brings dynamic functionality to your websites!

PHP is a widely used, open-source, platform-independent scripting language for server-side web development. This is a free and open-source coding language that can run on any operating system. PHP is used for making web servers. Therefore, you will also work with MySQL as the database management system. MySQL is a structured query language that plays a central role.

PHP with MySQL allows you to create dynamic web content, implement real-time applications, and enhance web security through user authentication and two-step verification. In the previous competencies, you learnt about HTML, CSS, and JavaScript, which can greatly enhance the learning process.

In this session, you will learn about the basics of PHP, how to start it, different operators we use at PHP, various control structures that manage conditions and repetitions, functions in PHP, arrays for grouping and connecting to a MySQL database to insert and read records.

Learning Objectives

At the end of this session, learners will be able to:

- Explain PHP's role in server-side web development.
- Create dynamic web content using PHP and MySQL.
- Implement basic user interactions and form handling.

Contents outline

4.1 Introduction to PHP

PHP (Hypertext Preprocessor) is a widely-used server-side scripting language designed to create dynamic and interactive web pages. Unlike HTML, which provides the structure of a page, or CSS, which handles styling, PHP runs on the server to process data, interact with databases, and generate content dynamically before sending it to the user's browser.

With PHP, you can:

- Build dynamic websites that change content based on user actions.
- Handle form submissions and user input.
- Connect to and interact with databases like MySQL.
- Perform calculations, manage sessions, and control access to web resources.

To start coding in PHP, you need a server environment to process scripts, even though PHP works with HTML.

Essential Requirements

1. **Web Server:** Runs PHP scripts and serves webpages (e.g., Apache, Nginx).
2. **PHP Interpreter:** Executes PHP code; must be configured with the server.
3. **Database System** (Optional but Common): Stores and retrieves data (e.g., MySQL, MariaDB).
4. **Code Editor or IDE**: Helps write and manage PHP code (e.g., VS Code, PhpStorm).
5. **Pre-configured Packages** (Optional for Beginners): Bundles PHP, server, and database for easy setup (e.g., XAMPP, WAMP, MAMP).

This is highly recommended for beginners.

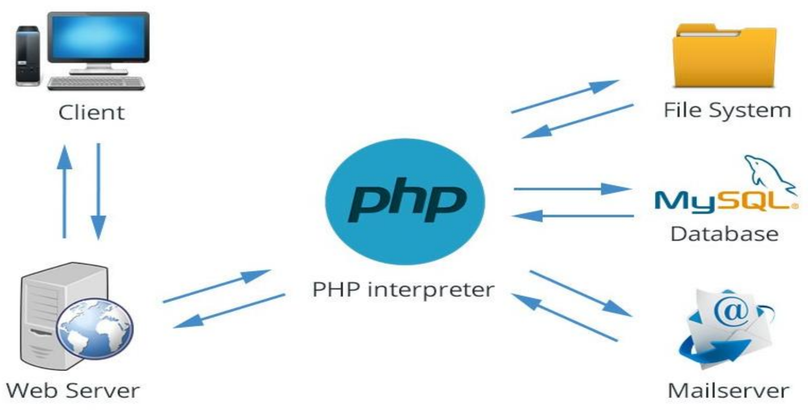


Figure 30: PHP Dynamic content Interaction

PHP Editors

PHP editors help you write, edit, and debug code. They provide syntax highlighting, error checking, and code formatting. You can use:

- **Offline editors**: Visual Studio Code, PhpStorm, Sublime Text, Atom
- **Online editors**: https://www.codechef.com/ide, https://www.codechef.com/ide, https://www.tutorialspoint.com, https://www.ideone.com

Setting Up PHP Environment

Since PHP is a server-side scripting language, which means it needs a web server to process scripts and generate dynamic content. XAMPP provides a ready-to-use environment:

- PHP**:** The scripting language used to create dynamic websites.
- Apache**:** The web server that processes requests and serves web pages.
- MySQL/MariaDB**:** The database server that stores website data.
- phpMyAdmin**:** A web-based interface to manage your databases easily.

By setting up a PHP environment, you create a local server on your computer, enabling you to build, test, and manage PHP-based websites efficiently.

Step 1: Download and Install XAMPP

- Go to https://www.apachefriends.org.
- Download XAMPP for your operating system (Windows, macOS, Linux).
- Run the installer and select the key components: **Apache, MySQL, PHP, phpMyAdmin.**
- Complete the installation.

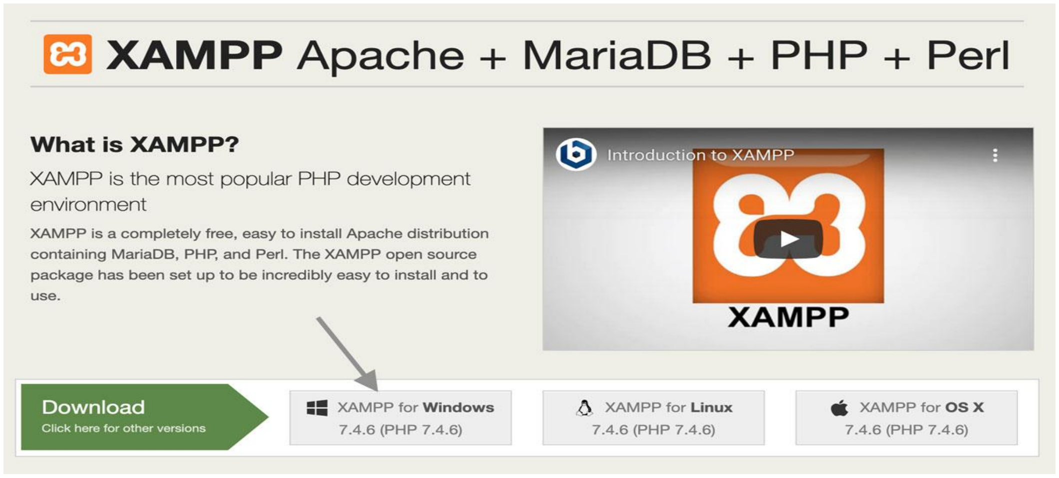
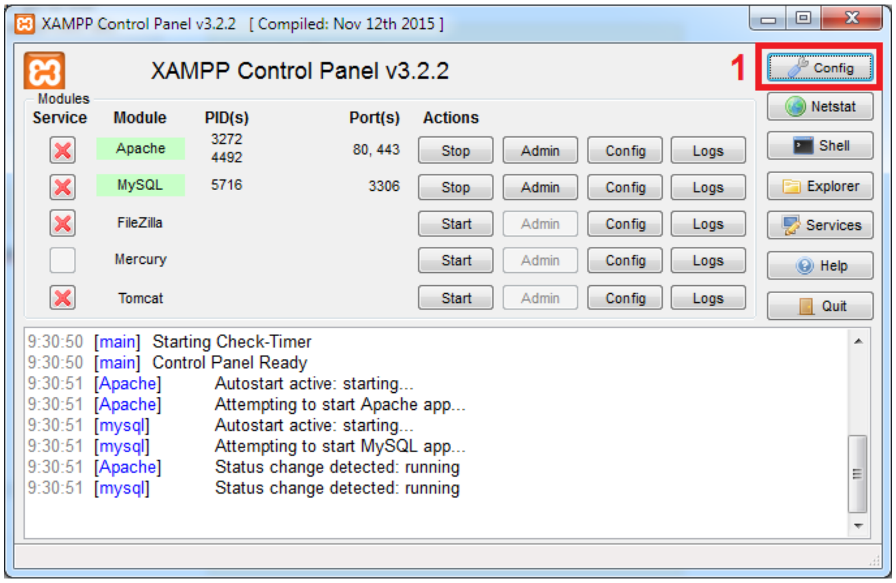

XAMPP Apache + MariaDB + PHP + Perl
What is XAMPP?
XAMPP is the most popular PHP development environment
XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.
Introduction to XAMPP
XAMPP
Download
Click here for other versions
XAMPP for Windows 7.4.6 (PHP 7.4.6)
XAMPP for Linux 7.4.6 (PHP 7.4.6)
XAMPP for OS X 7.4.6 (PHP 7.4.6)

Figure 31: XAMPP download

Step 2: Start Apache and MySQL

- Open the **XAMPP Control Panel**.
- Start **Apache** (web server) and **MySQL** (database server).
- Ensure both services are running (green status).



Module	PID(s)	Port(s)	Actions			
Apache	3272 4492	80, 443	Stop	Admin	Config	Logs
MySQL	5716	3306	Stop	Admin	Config	Logs
FileZilla			Start	Admin	Config	Logs
Mercury			Start	Admin	Config	Logs
Tomcat			Start	Admin	Config	Logs

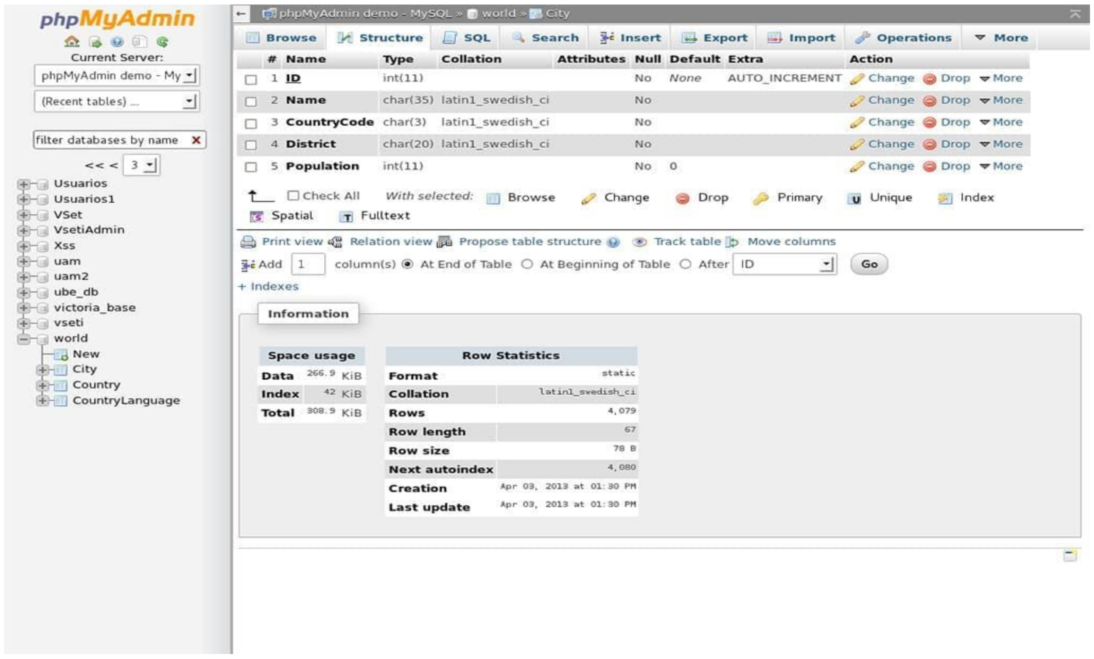
```
9:30:50  [main]    Starting Check-Timer
9:30:50  [main]    Control Panel Ready
9:30:51  [Apache]      Autostart active: starting...
9:30:51  [Apache]      Attempting to start Apache app...
9:30:51  [mysql]       Autostart active: starting...
9:30:51  [mysql]       Attempting to start MySQL app...
9:30:51  [Apache]      Status change detected: running
9:30:51  [mysql]       Status change detected: running
```

Figure 32: Xampp control panal

Step 3: Access phpMyAdmin

- Open a browser and go to `http://localhost/phpmyadmin`.

- Create a new database for your PHP projects or WordPress installation.



#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	ID	int(11)			No	None	AUTO_INCREMENT	Change Drop More
2	Name	char(35)	latin1_swedish_ci		No			Change Drop More
3	CountryCode	char(3)	latin1_swedish_ci		No			Change Drop More
4	District	char(20)	latin1_swedish_ci		No			Change Drop More
5	Population	int(11)			No	0		Change Drop More

Check All *With selected:* Browse Change Drop Primary Unique Index Spatial Fulltext

Print view Relation view Propose table structure Track table Move columns

Add 1 column(s) (●) At End of Table ( ) At Beginning of Table ( ) After ID Go

+ Indexes

Information

Space usage		
Data	266.9	KiB
Index	42	KiB
Total	308.9	KiB

Row Statistics	
Format	static
Collation	latin1_swedish_ci
Rows	4,079
Row length	67
Row size	78 B
Next autoindex	4,080
Creation	Apr 03, 2013 at 01:30 PM
Last update	Apr 03, 2013 at 01:30 PM

Figure 33: Phpmyadmin window

Step 4: Test Your PHP Environment

- Go to the **htdocs** folder in the XAMPP installation directory.
- Create a new folder for your project (e.g., `testsite`).
- Inside it, create `index.php` with:

```
<?php
echo "Hello, World!";
?>
```

- Open `http://localhost/testsite/index.php` in a browser.
- If you see **“Hello, World!”**, your PHP environment is working correctly.

Tips

- Always start **Apache** and **MySQL** before working on your projects.
- Keep your projects organized in separate folders inside `htdocs`.
- Use phpMyAdmin to easily manage databases.

4.2 PHP Basics

PHP Syntax

- Every PHP script starts and ends with **special tags**:
- A PHP script starts with <?php and ends with?>

```
  <?php
 // your PHP code goes here
   ?>
```

- Anything inside these tags will be processed by the server.
- Anything outside will be treated as normal HTML.

PHP Comments

Comments help to learn how to document your code effectively. There are several ways to comment PHP code. Let's have a look at the most useful ones:

Single-line comment:	**Multi-line comment**:
// This is a single-line comment # This is also a single-line comment	/* This is a multi-line comment. It can span multiple lines. */

Basic Output: echo and print

- **echo**: Used to display output (faster, can print multiple values).
- **print**: Similar to echo but slightly slower and can only print one value at a time.

Example:

```
<?php
echo "Hello, PHP!";
print "Hello, PHP!";
 ?>
```

Variables and Data Types

- Variables in PHP start with a **$** sign.
- They store data that can change while your program runs.
- variables are case-sensitive

- Unlike some other languages, you don't need to declare the type of a variable before using it PHP figures it out automatically.
- PHP variables support integer, float, Boolean, string, arrays, object and null

```
<?php
$name = "Betelhem";   // a string
$age = 25;            // an integer
$height = 5.6;        // a float
$isStudent = true;    // a boolean
?>
```

Variables Naming Rules

- Must start with $ followed by a letter or underscore.
- Cannot start with a number.
- Can contain letters, numbers, and underscores.
- Case-sensitive ($name and $Name are different).

Example Valid name: $user_name, $age1 Invalid name : $1age, $user-name

Constants

- Constants are like variables, but their value cannot change during the script.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).
- Created with define define (name, value) or const.

```
<?php
define("Hello ", " Welcome to Amhara Digital!");
const VERSION = "1.0";
echo Hello;   // prints Welcome to Amhara Digital!
echo VERSION;     // prints 1.0
?>
```

var_dump:

- The var_dump() is a built-in function that accepts a variable and displays its type and value.

```
<?php

$name = "Amen";
//To display the information of the $name variable:
var_dump ($name);
?>

Output:
string(4) "Amen"
```

PHP operators

This section covers PHP’s most commonly used operators, including logical, assignment, increment/decrement, and comparison operators.

Arithmetic Operators:

Table 13: php operators

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

Assignment operators: are used with numeric values to write a value to a variable.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

Comparison operators: allow you to compare two values of the same or different types. The PHP comparison operators are used to compare two values (numbers or strings):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Logical Operators: The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP Increment / Decrement Operators: The PHP increment operators are used to increment a variable's value. The PHP decrement operators are used to decrement a variable's value.

Operator	Same as...	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP String Operators:

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

4.3 Control Flow Structure

Conditional statements are used to execute different code based on different conditions.

PHP has three main types of flow control:

1. Conditional Statements (Selection): Choose between options.
2. Looping Statements (Iteration): Repeat code multiple times.
3. Jump Statements: Change the normal flow (like skipping or stopping code).

Conditional Statements (Selection Statements): These are used when you want to check a condition and decide what code to run.

if statement: Runs code **only if** the condition is true.
Example:
`<?php`
`$age = 20;`
`if ($age >= 18) {`
`    echo "You are an adult.";`
`}`
`?>`
	if-else statement: Runs one block if the condition is true, otherwise another block.
	Example:
	`<?php`
	`$age = 15;`
	`if ($age >= 18) {`
	`    echo "You are an adult.";`
	`} else {`
	`    echo "You are under 18.";`
	`}`
	`?>`

if…elseif…else Statement: Checks multiple conditions in order.
Example:
`<?php`
`$num=0;`
`If ($num>0)`
`{`
`Echo (' The number is Positive ');`
`}`
`elseif($num<0)`
`{`
`        echo ("The number is negative");`
`}`
`else`
`{`
`        echo("the number is Zero");`
`}`
`?>`
	switch Statement: Best when you have **many possible values** to check.
	
	Example:
	`<?php`
	`$season="winter";`
	`switch ($season)`
	`{`
	`  case "summer":`
	`    echo " The season is Cold!";`
	`    break;`
	`  case "winter":`
	`    echo "The season is HOT!";`
	`    break;`
	`  default:`
	`    echo "The season is Windy!";`
	`}`
	`?>`

Looping Statements (Iteration Statements): allow for the repeated execution of a block of code. The common loop types (for, while, and do... while).

for Loop: Best used when you know the number of repetitions.
Has 3 parts:
• Initialization: starting value
• Condition: check if loop should continue
• Increment/Decrement: update the value each time
The syntax is:
for (Initialization; Condition; Increment/Decrement)
{
   // Code to be executed
}
	For loop example
	<?php
	
	for ($num =1; $num <=3; $num++) {
	   echo ('Digital Skill ' . $num . "\n");
	}
	?>
	Output
	Digital Skill 1
	Digital Skill 2
	Digital Skill 3

while Loop: used when we want to execute a block of code as long as a test expression continues to be true.
The syntax is:
 	
while (condition)
{
// executable code
}
	While loop example
	<?php
	$i=1;
	while ($i<=3)
	{
	   echo ('Digital Skill ' . $i . "\n");
	   $i++;
	}
	?>
	Output
	
	Digital Skill 1
	Digital Skill 2
	Digital Skill 3

do-while Loop: used when we want to execute a block of code at least once and then as long as a test expression is true.
 	
The syntax is
do
{
//code execution
}
while (condition);
	While do-while loop example
	$i = 0;
	<?php
	$i = 3; // initialization
	
	do {
	  echo " Digital Skill ".$i ."\n";
	  $i--;
	} while ($i >=1);
	?>

Jump Statements: These statements alter the normal flow of execution within loops or other control structures. The two keywords to alter this cycle are Break and continue.

The break statement: in PHP is used to terminate the execution of the current for, foreach, while, do-while, or switch structure.	Break Statement example *<?php* *// Using break in a for loop*

When a break is encountered, the program control immediately jumps to the statement following the terminated structure.
In the previous examples under the switch statement, you learnt the “break” statement terminating each condition.
	for ($i = 1; $i <= 10; $i++) {
	   if ($i == 4) {
	      break; // Exit the loop when $i is 5
	   }
	   echo "digital Skill ". $i . PHP_EOL;
	}
	?>
	Output
	Digital Skill 1
	Digital Skill 2
	Digital Skill 3

The Continue statement: in PHP is used within looping structures (such as for, while, do...while) to skip the rest of the current iteration of the loop and proceed to the next iteration.
 	
In the example given, the continue statement allows continuation of the execution to the next level, skipping while the value i is 2 or the value of I is 3
	Continue Statement example
	
	<?php
	for ($i = 1; $i <= 5; $i++) {
	   if (($i == 2)\|\|($i==3)) {
	      continue; // Skip the rest of this iteration when $i is 2
	   }
	   echo "Digital Skill : " . $i . "\n";
	}
	?>
	Output
	
	Digital Skill: 1
	Digital Skill: 4
	Digital Skill: 5

4.4 PHP Functions

Functions are a type of procedure or routine that gets executed whenever some other code block calls it. PHP has over 1000 built-in functions. These functions help developers avoid redundant work and focus more on the logic rather than routine things. Apart from its own functions, PHP also lets you create your own functions.

Every function needs a name, optionally has one or more arguments, and most importantly, defines some kind of procedure to be followed within the body, that is, code to be executed. There are two primary categories of functions: built-in functions and user-defined functions.

PHP Built-in Functions

Built-in functions are pre-defined functions that come bundled with PHP's core or its extensions. They are readily available for use without requiring any explicit declaration or definition by the developer. These functions cover a wide range of common operations, including:

- String manipulation: strlen(), str_replace(), substr()
- Array handling: count(), array_push(), sort()
- Mathematical operations: round(), rand(), sqrt()
- Date and time formatting: date(), time()
- Input/output: echo(), print_r(), file_get_contents()

Example:

If we want to generate a random number between 5 and 10, the syntax is echo rand(min, max). and want the square root of 64 then:
<?php
echo rand(5,10);
echo "\n";
echo sqrt(64);
?>
	If we want to count an array element
	
	<?php
	$students=array('Hirut', 'Hale', 'Nehemiya', 'Amen', 'Barkot');
	echo count($students);
	
	?>

<?php *//generate year, date, and time* *echo date("y-m-d"."\n");* *//including the hour, minutes, and seconds* *echo date("y-m-d h:m:s " ."\n");* *?>*	If we want to get the length of the string in the phrase 'Abebaw Kebede' *<?php* *echo strlen('Abebaw Kebede');* *?>*

PHP User-Defined Functions are: custom functions created by the developer to encapsulate specific logic or perform tasks not covered by built-in functions.

They allow for code reusability, modularity, and improved organization within a project. User-defined functions are declared using the function keyword, followed by a chosen name, optional parameters, and a block of code enclosed in curly braces.

To execute the code within a user-defined function, you call it by its name, followed by parentheses, and pass any required arguments. The basic syntax of a PHP function is given below:

PHP function Syntax

```
<?php
Function functionName ($argument1, $argument2...) // function definition
{
// code to be executed
}
functionName ($argument1, $argument2...); // function call
?>
```

Function Definition: In PHP, the terms "function definition" and "function declaration" are often used interchangeably. In our example, we create a function named sum. The opening curly brace { indicates the beginning of the function code, and the closing curly brace } indicates the end of the function.

```
function sum($num1, $num2)
{
   return $num1 + $num2;
}
```

Function Call: Calling a Function in PHP Once a function is declared, it can be invoked (called) by simply using the function name followed by parentheses.

```
<?php
  function sum($num1, $num2)
 {
   return $num1 + $num2;
}
echo sum (5, 3);  // Outputs: 8
?>
```

PHP function Examples

```
<?php
function Greeting()  // function definition
{
echo "Hello Haleluya!"; // what this function does
}
Greeting(); // function call
?>
```

On PHP More than one argument can be used whenever needed in the following example, we have three arguments and values passed to these arguments:

```
<?php
function personProfile($name, $city, $job) // function definition with three arguments
{
echo "This ".$name." from ".$city." "; echo "";
echo " & job is ".$job." \n";
}
personProfile("Mahlet", "Bahirdar", "Web Dev");
echo "";
personProfile("Haleluya", "Shimbit", "Graphic Designer"); echo "";
personProfile("Endalew", "Gondar", "System Deigner");
?>
```

Output

```
This Mahlet from Bahirdar  & job is Web Dev
This Haleluya from Shimbit  & job is Graphic Designer
This Endalew from Gondar  & job is System Deigner
```

4.5 PHP Array

An array stores multiple values in a single variable. An array is a special variable that can hold many values under a single name, and you can access the values by referring to an index number or name. The most common are strings and numbers

```
<?php
$languages = array("Python", "PHP", "ASP", "Java");
?>
```

Array elements are accessed like this: $arrayName [positionIndex]. For the above example, we could access "PHP" this way: $languages[1]. Position index is 1 because in programming languages, the first element is always element 0. So, PHP would be 1 in this case.

In PHP, there are three types of arrays these are indexed arrays (Arrays with a numeric index). Associative arrays (Arrays with named keys) and Multidimensional arrays (Arrays containing one or more arrays)

Indexed arrays: is a type of array where elements are stored and accessed using numerical indices. By default, these indices are automatically assigned starting from 0 for the first element, 1 for the second, and so on, incrementing sequentially. We can create these kinds of arrays in two ways shown below:

```
<?php
$names = array("Hale", "Amen", "Diva");
?>
```

```
<?php
//This is a rather manual way of doing it
$names[0] = "Hale";
$names[1] = "Amen";
$names[2] = "Diva";
?>
```

An example where we print values from the array is:

```
<?php
$names = array("Hale", "Amen", "Diva");
echo "My friends are " . $names[0] . ", " . $names[1] . " and " . $names[2];
?>
Output
My friends are Hale, Amen and Diva
```

Associative arrays

Associative arrays are arrays which use named keys that you assign. Again, there are two ways we can create them:

It is a type of array where elements are accessed using named keys rather than numerical indexes. Unlike numerically indexed arrays where elements are ordered and accessed by their position (0, 1, 2, etc.), associative arrays use descriptive strings or integers as keys to identify and retrieve values.

```
<?php
$namesAge = array ("Hale"=>"20",
"Amen"=>"16", "Diva"=>"43");
?>
```

```
<?php
// this is a rather manual way of doing it
$namesAge['Hale'] = "20";
$namesAge['Amen'] = "18";
$namesAge['Diva'] = "43";
?>
```

An example where we print values from the array is:

```
<?php
$namesAge = array("Hale"=>"20", "Amen"=>"16", "Diva"=>"43"); echo "Hale's age is "
. $namesAge['Hale'] . " years old.";
?>
// RESULT
```

Hale's age is 20 years old.

Multidimensional arrays: This is a rather advanced PHP stuff, but for the sake of this tutorial, just understand what a multidimensional array is. Basically, it is an array whose elements are other arrays. For example, a two-dimensional array requires two indices (e.g., array[0][1]),

Summary

At this moment, we believe that you have a basic understanding of PHP. In this module, we focused on the basics of PHP, that discussed in detail about the basics of PHP: which includes the syntax of PHP, how to code with it, and the editors of PHP. The sessions also covered various mathematical operators such as arithmetic, logical, comparison, etc., in addition to their functions, their parts, and how to define and call them were discussed. Additionally, array and their structure as well as how to access them, were covered.

Review Questions

1. Which one of the following shows the surrounding delimiters of PHP server scripts?
   - A. <?script . . script?>
   - B. <style> . . . </style>
   - C. <?php. . . ?>
   - D. <php>. . . </php>
2. What is the correct way to create a function in PHP?
   - A. Create myfunction()
   - B. This_is function()
   - C. function myfunction()
   - D. new_function function();
3. Given the code below what will be the output of the given code?

```
<?php
for ($i = 10; $i >0; $i--)
{
   if($i%2==0)
   {
      if ($i==8 || $i==6)
      {
         continue;
      }
   echo ($i . " \t");
```

What will be the

- A. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
- B. 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
- C. 2, 4, 6, 8, 10
- D. 10, 4, 2

} *}* *?>*	

4. Given the function, what will be the output of the given function code?

<?php *function Array_nums()* *{* *$var = array('Hale','Mahlet','Belete','Endalew');* *$arrayreverse = array_reverse ($var);* *print_r ($arrayreverse);* *}* *Array_nums();* *?>*	A. Array ([0] => Endalew [1] => Belete [2] => Mahlet [3] => Hale) B. Array ([0] => Hale [1] => Mahlet [2] => Belete [3] => Endalew) C. Array (Hale, Mahlet, Belete, Endalew) D. Array (Endalew, Belete, Mahlet, Hale)

Session 5: Basics of Databases

Introduction

In web development, almost every PHP application needs to store and manage data user information, products, comments, etc. To handle this efficiently, we use databases. This session introduces you to the basic concepts of databases, DBMS (like MySQL), and phpMyAdmin a user-friendly tool that helps manage databases without writing SQL manually.

By the end, you will be able to create a simple database, tables, and records using phpMyAdmin, and understand how PHP connects to databases.

Learning objective

By the end of this session, you will be able to:

- Explain what a database is and why it’s needed in PHP development.
- Describe what a DBMS (Database Management System) is, focusing on MySQL.
- Understand the basic structure of a database (tables, rows, columns, keys).
- Use phpMyAdmin to: Create databases and tables, Insert, edit, and delete data. And run basic SQL queries.

Content outline

5.1 Introduction to Database

What is a Database?

A database is an organized collection of data that can be easily accessed, managed, and updated. Instead of storing data in random files, databases store it in a structured way using tables made up of rows and columns.

Example:

A school website might use a database to store:

- Student information
- Teacher details

- Courses and grades

Why Use a Database?

Databases make it easier to:

- Store large amounts of information efficiently
- Quickly search and retrieve data
- Maintain data accuracy and consistency

Basic Database Structure

A **database** contains one or more **tables** and each table stores data about one type of entity.

- **Table**: A collection of related data (e.g., students, courses).
- **Row (Record)**: single entry in the table (e.g., one student's details).
- **Column (Field)**: A specific attribute (e.g., name, email, age).

Example Table: students

id	**name**	**email**	**course**
1	Sara Ali	sara@gmail.com	PHP
2	John Doe	john@gmail.com	Python

  - The table name: students
  - The columns: id, name, email, course
  - Each row represents one student record

What is Database Management System (DBMS)?

A **DBMS** is software that helps users create, store, manage, and interact with databases. It provides tools and commands for storing, modifying, and retrieving data.

Popular DBMS examples:

- **MySQL**: Open-source, popular for web development
- **PostgreSQL**: Advanced, open-source, very stable
- **Oracle Database**: Enterprise-grade, commercial
- **SQLite**: Lightweight, used in mobile apps

Key function of a DBMS

- Manages data storage and retrieval
- Ensures data security and integrity
- Supports multi-user access
- Allows backup and recovery of data

Keys and Relationships

In databases, keys are special fields (columns) used to identify and link records between tables. They help maintain data integrity (no duplicates, no confusion) and establish relationships between tables. There are mainly two types of keys that we commonly use when working with tables:

Primary Key: is a column (or combination of columns) that **uniquely identifies each record** in a table.

- It must contain unique values.
- It cannot be NULL (empty).
- Each table should have only one primary key.

A **Foreign Key** is a column in one table that refers to the **Primary Key** in another table. It's used to **create relationships** between tables.

- It connects two tables together.
- It enforces **referential integrity** (you can't reference something that doesn't exist).
- It allows **relational databases** to reduce data duplication.

Types of Relationships Between Tables

1. **One-to-One (1:1)**
    - One record in Table A relates to one record in Table B.
    - Example: Each student has **one** student ID card.
2. **One-to-Many (1: N)**
    - One record in Table A relates to **many** records in Table B.

   - Example: One course has **many** students.
   - The most common type of relationship.

3. **Many-to-Many (M: N)**
   - Many records in Table A relate to many in Table B.
   - Example: Students can enroll in **many courses**, and each course can have **many students**.
   - Usually implemented with a **third (junction) table**.

5.2 Basic SQL

What is SQL?

SQL (Structured Query Language) is the standard language used to communicate with databases. You use SQL to:

- Create and modify tables
- Insert, update, and delete data
- Retrieve specific information

SQL Categories

1. DDL (Data Definition Language): defines or changes the structure of a database such as creating, altering, or deleting tables.

   Common Commands:

   - CREATE: Create a new database or table
   - ALTER: Change the structure of an existing table (add or remove a column)
   - DROP: Delete a table or database completely

2. DML (Data Manipulation Language): Used to add, edit, or remove the actual data stored inside tables.

   Common Commands:

- INSERT: Add new data (records)
- UPDATE: Modify existing data
- DELETE: Remove data from a table

3. DQL (Data Query Language): Used to retrieve or search for data stored in the database
   Common Command:
   - SELECT: Extracts data from one or more tables
4. DCL (Data Control Language): Used to control access to data and manage user permissions in a database.

   Common Commands:

   - GRANT: Give permission to a user
   - REVOKE: Remove permission from a user

Common SQL Commands

SQL (Structured Query Language) provides a set of commands to interact with the database. These commands allow you to create, modify, store, retrieve, and delete data in a structured way.

1. CREATE: Create a Database or Table

Used to create a new database or a new table inside a database.

Example1: create a Database

```
CREATE DATABASE school;
```

- This SQL command creates a new database named **school**.

Example2: Create a Table

```
CREATE TABLE students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(100),
  course VARCHAR(50)
);
```

- This SQL **command** Creates a table called **students** with four columns: Id: unique ID for each student, Name: student name, email: student email, course: course name

2. INSERT: Add Data into a Table

Used to **insert (add)** new records (rows) into a table.

```
INSERT INTO students (name, email, course)
VALUES ('Sara Ali', 'sara@gmail.com', 'PHP');
```

- This SQL command Adds one student record to the table.

Insert Multiple Records Example:

```
INSERT INTO students (name, email, course)
VALUES
('John Doe', 'john@gmail.com', 'Python'),
('Muna Ibrahim', 'muna@gmail.com', 'JavaScript');
```

3. SELECT: Retrieve Data

Used to retrieve (read) data from one or more tables.

```
SELECT * FROM students;
```

- This SQL command Shows all records and all columns from the table.

Select Specific Columns:

```
SELECT name, course FROM students;
```

- This SQL command Shows only name and course from the table.

Filter Results

```
SELECT * FROM students WHERE course = 'PHP';
```

- This SQL command only show records with PHP course

4. UPDATE: Modify Existing Data

Used to change or update existing records in a table.

```
UPDATE students
SET course = 'Python'
WHERE id = 1;
```

- This SQL command Changes the course of the student whose ID is 1 to *Python*.

5. **DELETE: Remove Data**

   Used to delete records from a table.

```
DELETE FROM students WHERE id = 2;
```

- This SQL command Deletes the record where the student ID is 2.

Warning: If you forget the WHERE clause All records will be deleted!

```
DELETE FROM students;
```

- This SQL command will delete all records

6. **DROP: Delete Table or Database**

Used to **completely remove** a table or database (structure and data).

Delete a Table:

```
DROP TABLE students;
```

Delete a Database:

```
DROP DATABASE school;
```

5.3 PhpMyAdmin

What is phpMyAdmin?

phpMyAdmin is a web-based tool written in PHP that allows you to manage MySQL databases through a graphical user interface (GUI) without writing SQL commands manually. It's included in local server packages like XAMPP, WAMP, or MAMP, which your students will use for PHP development. In simple term phpMyAdmin helps you *see* your databases, tables, and data

Why Use phpMyAdmin?

- Easy to create and manage databases without SQL commands.
- Great for beginners to understand database structure visually.
- Let's you run SQL queries directly for practice.
- Used for **importing/exporting** databases (e.g., for projects).

Steps to PhpMyAdmin

Step 1: Install a Local Server

Before we start **local server, environment** must be installed. Local server includes:

- **Apache** (to run PHP)
- **MySQL** (for databases)
- **phpMyAdmin** (to manage MySQL)

The easiest way: Download and install XAMPP

Here is the Download link: https://www.apachefriends.org

After installation:

1. Open **XAMPP Control Panel**
2. Start **Apache** and **MySQL** services.
3. Wait until both show green “Running” status.

Step 2: Open phpMyAdmin

1. Open your web browser.
2. Type this in the address bar:
3. http://localhost/phpmyadmin
4. The **phpMyAdmin dashboard** will open.

Step 3: Create a New Database

1. On the left sidebar, click **New**.
2. Type a database name e.g., school_db.
3. Click **Create**.

Now you have an empty database name school_db

Step 4: Create a Table

1. Select your database (school_db).
2. Under **Create Table**, enter:
    - Table name: students
    - Number of columns: 5
3. Click **Go**.
4. Define your columns:

Column Name	Type	Length	Attributes
Id	INT	11	Primary Key, Auto Increment
Name	VARCHAR	50	
Email	VARCHAR	100	
Age	INT		
Course	VARCHAR	50	

5. Click **Save**.

Now your table is now created.

Step 5: Insert Data (Records)

1. Click on your table (students).
2. Click **Insert** on the top menu.
3. Fill in the form with sample data:
    - name: Sara tamer, email: sara@gmail.com, age: 20, course: PHP
4. Click **Go**.

Now you've successfully added one record.

Step 6: View Data

1. Click **Browse** to see all your records in a table format.
2. You can **edit**, **delete**, or **add new** data directly from this view.

Step 7: Run SQL Queries

phpMyAdmin also lets you **write and run SQL commands** manually.

1. Click on the **SQL** tab.
2. Type a command like:

```
SELECT * FROM students;
```

3. Click **Go**

- The results will display below the query box.

Summary

In this session, you learned the foundational concepts of databases and their role in PHP applications. You explored how **phpMyAdmin** simplifies the management of MySQL databases and practiced creating databases, tables, and running queries.

Understanding these basics prepares you to Integrate databases into PHP projects, build dynamic, data-driven websites, and Confidently use SQL commands in future PHP lessons.

Practical Exercises

Exercise 1: Create a Database

- Open phpMyAdmin → Click **New** → Create database named school_db.

Exercise 2: Create a Table

- Create a table called students with:

Column Name	**Type**	**Length**	**Attributes**
Id	INT	11	Primary Key, Auto Increment
Name	VARCHAR	50	
Email	VARCHAR	100	
Age	INT		
Course	VARCHAR	50	

Exercise 3: Insert Data

Insert at least **three records** into the students table using the phpMyAdmin **Insert** tab.

Exercise 4: View and Edit Data

- Click **Browse** to see the records.
- Try editing one row (change the name or course).
- Try deleting one record.

Exercise 5: Run SQL Queries

Use the **SQL tab** and try:

```
SELECT * FROM students;

SELECT name, course FROM students WHERE age > 20;

UPDATE students SET course = 'PHP' WHERE name = 'Sara Ali';

DELETE FROM students WHERE id = 2;
```

Session 6: Connecting PHP with a Database (MySQL Integration)

Introduction

In the previous session, you learned how to create and manage databases using phpMyAdmin. Now, it's time to make PHP and MySQL work together. This connection allows PHP applications to store, retrieve, update, and delete data dynamically. Connecting PHP with a MySQL database is a crucial skill for building real world web applications, such as login systems, registration forms, blogs, and e-commerce sites.

In this session, you will learn how to connect PHP scripts to a MySQL database, perform basic SQL operations through PHP code, and handle connection errors effectively.

Learning Objectives

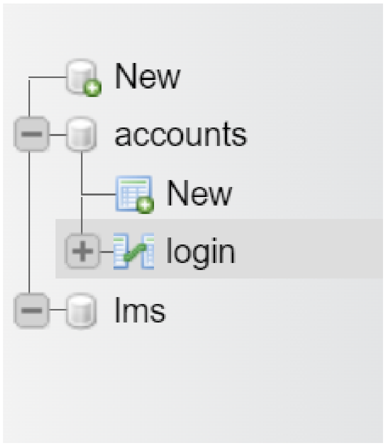
By the end of this session, you will be able to:

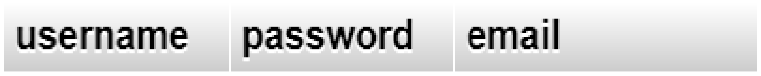
- Explain how PHP interacts with a MySQL database.
- Establish a database connection using the **MySQL** extension.
- Execute SQL queries in PHP (INSERT, SELECT, UPDATE, DELETE).

Content outline

6 .1 Connecting to Database

There are different approaches to get connected to a previously created database. Below, we'll explain how you can use it with the easiest one. First, let's get ready and create a database named "accounts" in MySQL. Under the database, we created a table named "login". Under this table, we created three fields: "username", "password", and "email", all of which are varchar datatypes.





Connecting to MySQL Databases

If we want to connect to the database named "accounts" in our local computer, where the user for MySQL is named "root" and doesn't have a password, the syntax for connecting to a MySQL database would be:

dbconnect.php

```
<?php
$servername = "localhost"; // Or your MySQL server's IP address/hostname
$username = "root"; // Your MySQL username
$password = ""; // Your MySQL password
$dbname = "accounts"; // The name of your database
   // Create connection
   $conn = new mysqli($servername, $username, $password, $dbname);
   // Check connection
   if ($conn->connect_error) {
      die("Connection failed: " . $conn->connect_error);
   }
   echo "Connected successfully";
?>
```

Considering your entered information is correct, you'd be successfully connected to the right database and ready to start writing and testing your queries. Otherwise, the respective error message would appear as defined by the die function.

Inserting a record using an SQL query to your database

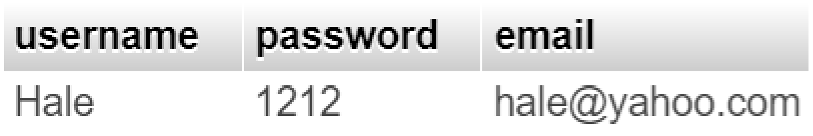
In the above example, we established a connection to a database. When connected, now proceed with the INSERT MySQL query. To add records to the login table in the MySQL database

Here is a full PHP code example that imports the linking code and insert methods:

Insert.php

```
<?php
include_once 'dbconnect.php'; // Adjust path as needed
// Assuming $conn is already established from the connection step
// Prepare and bind
 $sql = "INSERT INTO login (username, password, email) VALUES ('Hale', '1212',
'hale@yahoo.com')";
if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Here, the table looks like this after inserting the record.



username	password	email
Hale	1212	hale@yahoo.com

Reading a record using an SQL query to your database

In the above examples, we established a connection to a database, inserted a record into the database, and now proceed with the SELECT MySQL query. To read records from the login table in the MySQL database. Here is a full PHP code example that imports the linking code and insert methods:

Read.php

```
<?php
```

```
include_once 'dbconnect.php'; // Adjust path as needed
$sql = "SELECT username, password, email FROM login"; // Your SQL SELECT
statement
$result = $conn->query($sql);
if ($result->num_rows > 0) {
   // Output data of each row
   while($row = $result->fetch_assoc()) {
      echo "id: " . $row["username"]. " - Name: " . $row["password"]. " " . $row["email"].
"<br>";
   }
} else {
   echo "0 results";
}
$conn->close(); // Close the database connection
?>
```

Before going to the summary, let’s see how we can insert a record from an HTML form. What we need is the HTML form and PHP code that can push the data to the tables; therefore, the following two consecutive codes perform this action, taking the “dbconnect.php” connection above.

First, let’s look at how we can create an SQL statement for inserting to a database. And save it with the name “link.php”.

link.php

```
<?php
   include_once 'dbconnect.php';

 if(isset($_POST['submit']))
 {
        $username = $_POST ['username'];
        $password = $_POST ['password'];
        $email = $_POST ['email'];
 }
   $sql = "INSERT INTO login (username, password, email)
        VALUES ('$username','$password','$email');";

   mysqli_query($conn, $sql);
   /* If this page runs successfully, URL bar should have success */
```

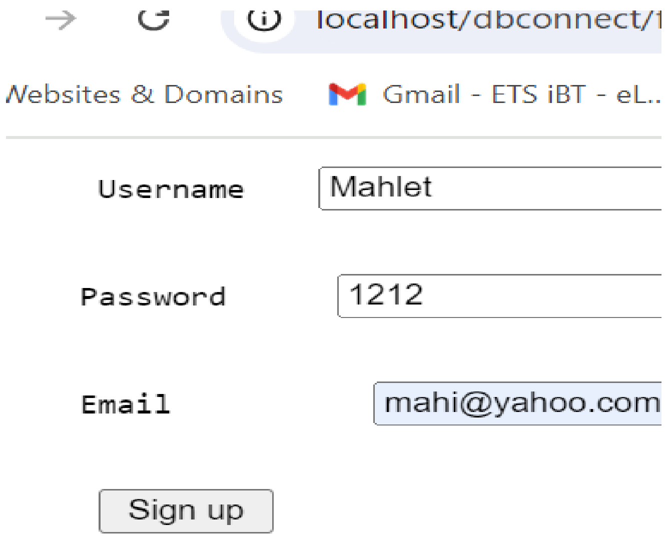
```
    header("location: form.html");
```

Now write the second code to create forms for filling data and save it as “MyForm.html”, then run the form on any browser and see that all the data you have filled on the form is inserted to the database.

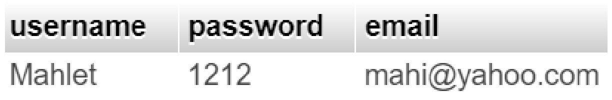
MyForm.html

```
<!DOCTYPE html>
    <Header>
    </header>
    <body>
    <form action="link.php" method="POST">
      <label for="username"> Username </label>   <input type="text"
name="username" placeholder="write User Name">
        <br>
  <label for="password">Password </label>     <input type="text"
name="password" placeholder="write Password">
        <br>
       <label for="email">Email </label>  <input type="text" name="email"
placeholder="write Email">
        <br>
        <button type="submit" name="submit"> Sign up </button>
  </pre>
    </form>
    </body>
```

Here is the form and its output on the database



Username	Mahlet
Password	1212
Email	mahi@yahoo.com



username	password	email
Mahlet	1212	mahi@yahoo.com

Summary

In this session, you learned how to connect PHP with a MySQL database and perform basic CRUD (Create, Read, Update, Delete) operations. Now you understand how PHP communicates with MySQL and how to display database data dynamically on a web page.

This skill is the **foundation of all dynamic PHP projects** from login systems to full web applications.

Practical Exercise

1. Create a PHP file named db_connect.php that connects to your school_db database and prints "Connected successfully!"
2. Write a PHP script that inserts a new record into the students table from a simple HTML form.
3. Create a PHP script that fetches all records from the students table and displays them in a styled HTML table.
4. Write a PHP script to update the course of a student where the name is "Selam Belay".
5. Write a PHP script to delete a student by ID.

Session 7: Developing a Website Using WordPress

Introduction

Dear Learners, in this session, you’ll learn the basics of building and managing websites using WordPress.

In today’s digital world, having a website is essential for individuals, businesses, and organizations. WordPress is a widely used platform for building websites because it is user-friendly, flexible, and powerful. Unlike traditional web development, WordPress allows you to create and manage websites without writing code.

In this session, you will learn how to set up a WordPress website, add and manage content like pages and blog posts, customize the website’s appearance using themes and widgets and enhance functionality with plugins.

By the end of the session, you will be able to create a professional website from scratch, even with no prior coding experience.

Learning Objectives

By the end of this session, learners will be able to:

- Explain the purpose and benefits of WordPress.
- Apply skills to set up and manage a WordPress website.
- Create a professional website by customizing themes, widgets, and plugins.

Content Outline

7.1 Understanding WordPress
7.2 Accessing and Navigating the WordPress Dashboard
7.3 Creating Basic Pages and Posts
7.4 Customizing a Website Using Themes and Menus

7.1 Understanding WordPress

What is WordPress?

WordPress is a Content Management System (CMS) - software that helps you create and manage a website without needing to know coding. It is free and open-source, meaning anyone can use and customize it. WordPress powers more than 40% of websites worldwide.

You can use WordPress to build many types of websites, such as:

- Personal Blogs: for writing and sharing stories or articles.
- Business Websites: showcase services, products, and company details.
- Portfolios: display creative work, photos, or designs.
- E-Commerce Stores: sell products online using plugins like WooCommerce.
- Educational or Community Sites: share learning materials or build online forums.

Why Choose WordPress?

- Beginner-friendly: No coding skills needed.
- Flexible & Customizable: Thousands of free and paid themes to change website design.
- Extendable with Plugins: Add features such as contact forms, online shops, SEO tools, and more.
- Large Support Community: Millions of tutorials, forums, and help articles available online.
- Cost-effective: Free to use, with only domain and hosting costs required.

WordPress.com vs. WordPress.org

- WordPress.com → Hosted service, easy setup, limited customization (like renting a space).
- WordPress.org → Self-hosted, full control, requires your own hosting and domain (like owning your home).

Installing WordPress with XAMPP

Step 1: Install XAMPP

1. Go to https://www.apachefriends.org.
2. Download XAMPP for your operating system (Windows).
3. Run the installer and follow the setup wizard.
    - Select components: Apache, MySQL, PHP, phpMyAdmin (others are optional).

- Finish installation.

Step 2: Start XAMPP

1. Open the XAMPP Control Panel**.**
2. Start Apache **(**for the web server).
3. Start MySQL (for the database).
    - Both should turn green when running.

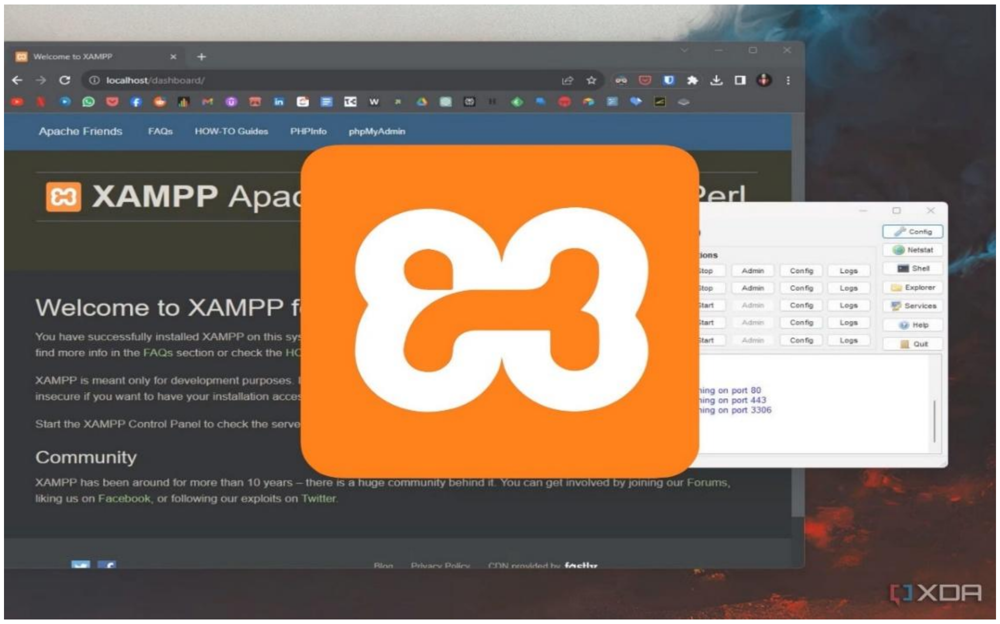
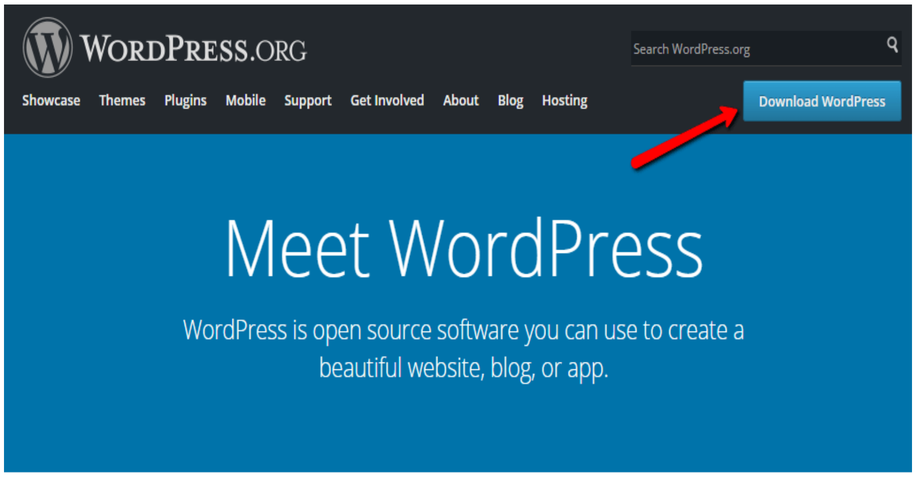


Figure 34: Xampp

Step 3: Download WordPress

1. Visit https://wordpress.org/download.
2. Download the latest WordPress ZIP file**.**
3. Extract the ZIP file.



Meet WordPress

WordPress is open source software you can use to create a beautiful website, blog, or app.

Figure 35: wordpress download page

Step 4: Move WordPress to XAMPP Folder

1. Go to your XAMPP installation folder (usually C:\xampp\htdocs on Windows).
2. Create a new folder (e.g., mywebsite).
3. Copy the extracted WordPress files into this folder.

Step 5: Create a Database

1. In your browser, go to: http://localhost/phpmyadmin.
2. Click Databases → Create a new database.
3. Enter a name (e.g., mywebsite) and click Create.

Step 6: Run WordPress Installation

1. In your browser, type: http://localhost/mywebsite.
2. Select your language and click Continue.
3. Enter database details:
   - Database Name: mywebsite
   - Username**:** root (default for XAMPP)
   - Password**:** (leave blank by default)
   - Database Host**:** localhost
   - Table Prefix**:** wp_ (leave as is)
4. Click Submit → Run Installation**.**

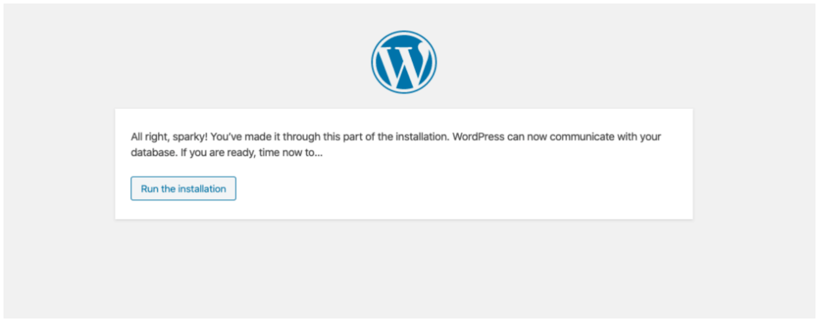
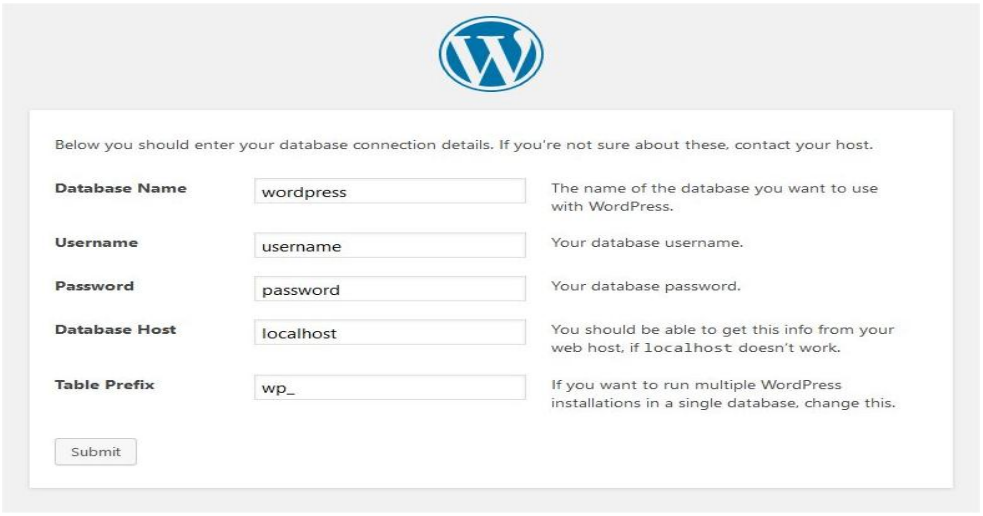

All right, sparky! You've made it through this part of the installation. WordPress can now communicate with your database. If you are ready, time now to…

Figure 36: Wordpress installation

Step 7: Set Up WordPress Site

1. Fill in:
   - Site Title
   - Username (for WordPress admin)
   - Password
   - Email
2. Click Install WordPress**.**


Below you should enter your database connection details. If you're not sure about these, contact your host.

Database Name	wordpress	The name of the database you want to use with WordPress.
Username	username	Your database username.
Password	password	Your database password.
Database Host	localhost	You should be able to get this info from your web host, if `localhost` doesn't work.
Table Prefix	wp_	If you want to run multiple WordPress installations in a single database, change this.

Figure 37: Wordpress site form filling

Step 8: Log In

1. Go to: http://localhost/mywebsite/wp-admin.
2. Enter your username and password.
3. You are now inside the WordPress Dashboard**!**

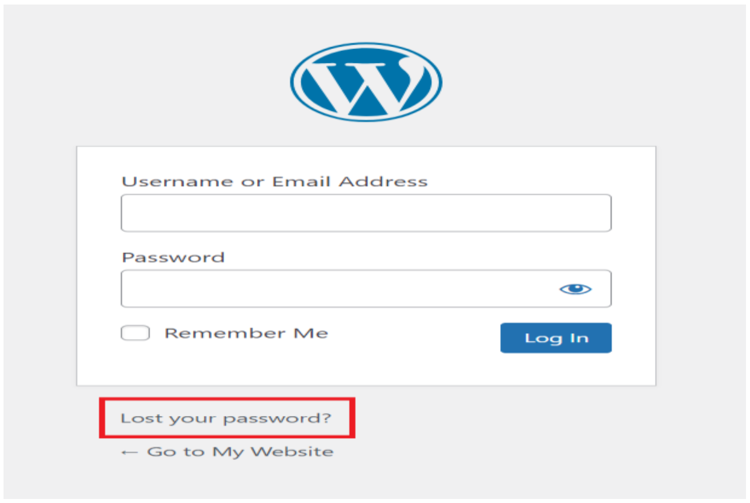


Figure 38:Wordpress login page

7.2 Accessing and Navigating the WordPress Dashboard

The **WordPress Dashboard** is the control center of your website. Once you log in, it is the first screen you see. From here, you can manage everything creating pages and posts, uploading images, customizing the design, installing plugins, and adjusting settings.

Learning how to access and navigate the dashboard is the first step toward confidently managing your WordPress site. Basic Steps for Accessing and Navigating the WordPress Dashboard**:**

Step 1: Access the Dashboard

1. Open your browser.
2. Type your website address followed by /wp-admin.
    - Example:
        - If installed locally → http://localhost/mywebsite/wp-admin
        - If installed online → http://www.mywebsite.com/wp-admin
3. Enter your username and password (set during installation).
4. Click Log In.

   You will now see the WordPress Dashboard (your website's control center).

Step 2: Understand the Dashboard Layout

The dashboard has two main sections:

- Main Menu (left side): All tools and features.

- Workspace (center area): Where you create, edit, and manage content.

Step 3: Key Dashboard Sections for Beginners

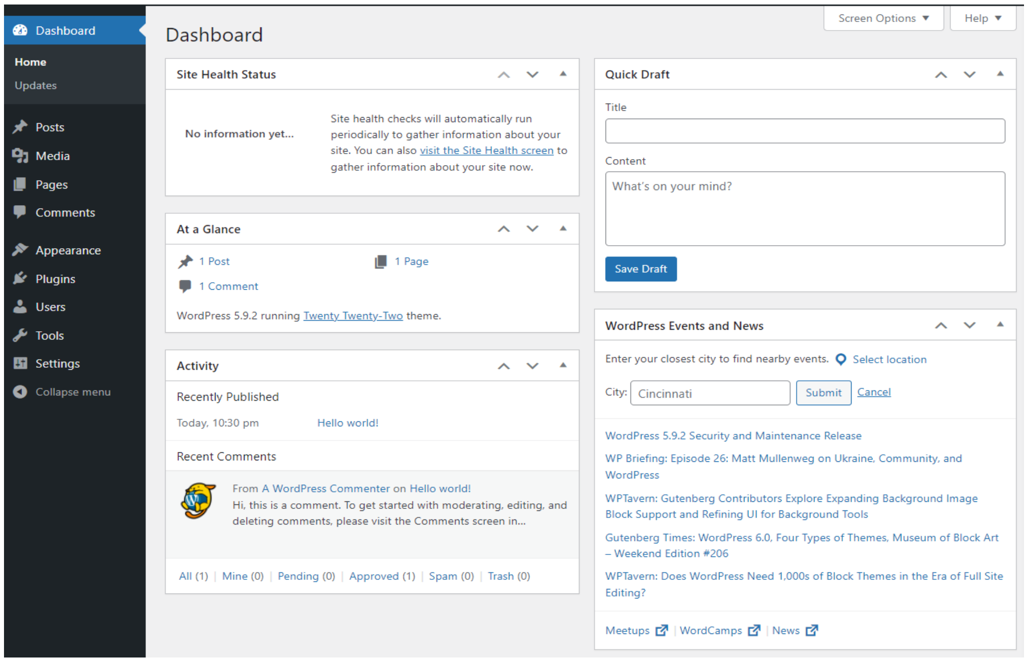
1. Dashboard (Home):
   - Quick overview of your site.
   - Shows updates, drafts, and recent activity.
2. Posts:
   - Create and manage blog articles.
   - Organize posts with **categories** and **tags**.
3. Media:
   - Library for images, videos, and documents.
   - Upload and manage files.
4. Pages**:**
   - Create and edit static pages (e.g., Home, About, Contact).
5. Comments**:**
   - Manage visitor comments (approve, reply, or delete).
6. Appearance**:**
   - Change your site's look with themes, menus, and widgets.
7. Plugins**:**
   - Add new features (e.g., contact forms, SEO tools, and security).
8. Users**:**
   - Manage who can log in (Admin, Editor, Author, and Subscriber).
9. Settings**:**
   - Adjust basic site options (site title, tagline, time zone, etc.).

Step 4: Hands-On Navigation Activity

- Try it yourself:
  A. Log in to your WordPress dashboard.
  B. Hover over each menu item on the left.
  C. Write down what you think it does.
- Discuss: *"Which dashboard feature do you think you'll use most often?"*

Step 5: Tips for Beginners

- Don't worry if the dashboard looks overwhelming at first - you'll mainly use Posts, Pages, Appearance, Plugins, and Settings.
- Use the Help tab (top right corner) for guidance on any page.
- Practice by clicking through each section to see what it does.



No information yet...

Site health checks will automatically run periodically to gather information about your site. You can also visit the Site Health screen to gather information about your site now.

1 Post
1 Page
1 Comment

WordPress 5.9.2 running Twenty Twenty-Two theme.

Recently Published

Today, 10:30 pm Hello world!

Recent Comments

From A WordPress Commenter on Hello world!
Hi, this is a comment. To get started with moderating, editing, and deleting comments, please visit the Comments screen in...

All (1) | Mine (0) | Pending (0) | Approved (1) | Spam (0) | Trash (0)

What's on your mind?

Enter your closest city to find nearby events. Select location

City: Cincinnati

WordPress 5.9.2 Security and Maintenance Release
WP Briefing: Episode 26: Matt Mullenweg on Ukraine, Community, and WordPress
WPTavern: Gutenberg Contributors Explore Expanding Background Image Block Support and Refining UI for Background Tools
Gutenberg Times: WordPress 6.0, Four Types of Themes, Museum of Block Art – Weekend Edition #206
WPTavern: Does WordPress Need 1,000s of Block Themes in the Era of Full Site Editing?

Figure 39:Wordpress dashboard

7.3 Creating Basic Pages and Posts

In WordPress, **Pages** and **Posts** are the two main ways to publish content.

- **Pages** are for static content like *Home, About, Contact*.
- **Posts** are for dynamic, regularly updated content like *blog articles or news updates*.

Understanding how to create and manage them is the foundation of building a WordPress website.

Step 1: Creating a Page

1. From the Dashboard menu, go to Pages → Add New.
2. Type a Page Title (e.g., *Home* or *About*).
3. Use the Block Editor (Gutenberg) to add text, images, or videos.
   - Add blocks using the **+** button.
4. Click **Publish** to make the page live.

Example: Create an *About Us* page with a short introduction and an image

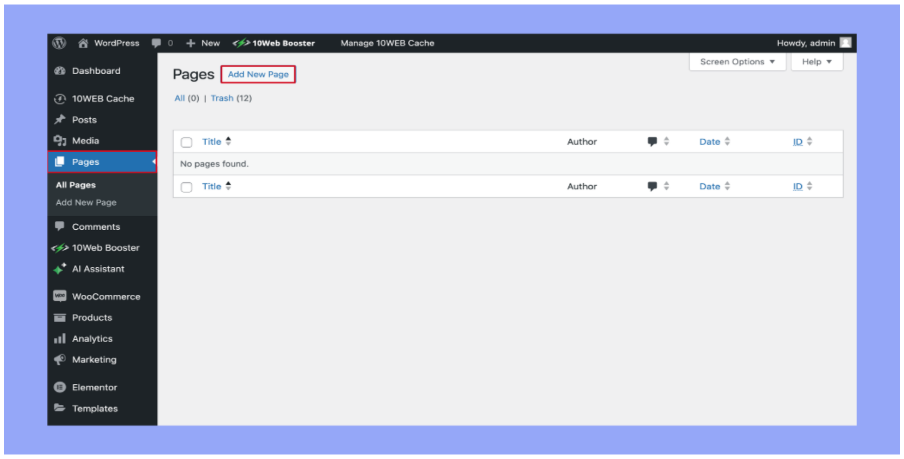


Figure 40: About us page creating

Step 2: Creating a Post

1. From the Dashboard menu, go to Posts → Add New**.**
2. Type a Post Title (e.g., *My First Blog Post*).
3. Write your content in the editor.
4. Add Categories and Tags to organize posts.
5. Insert images or media if needed.
6. Click Publish to make the post live.

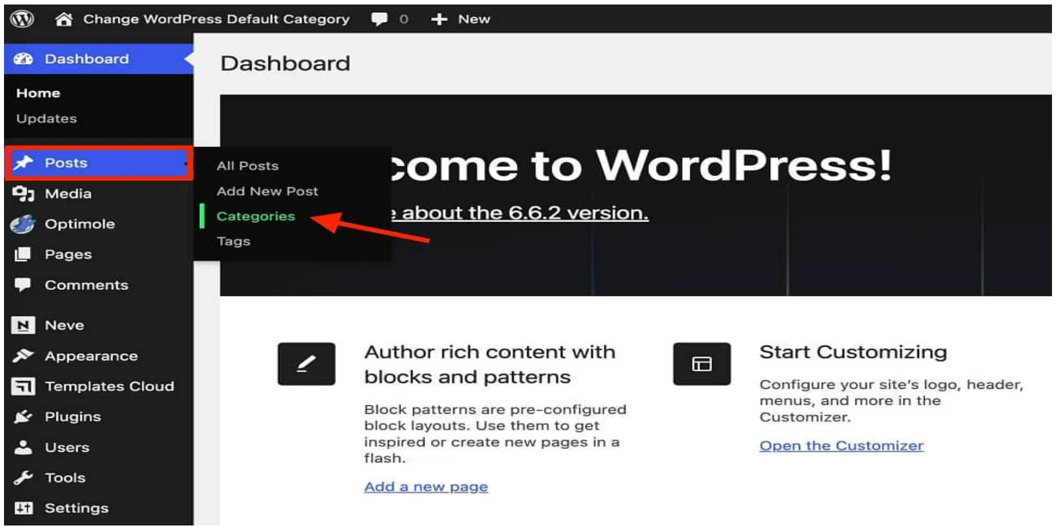
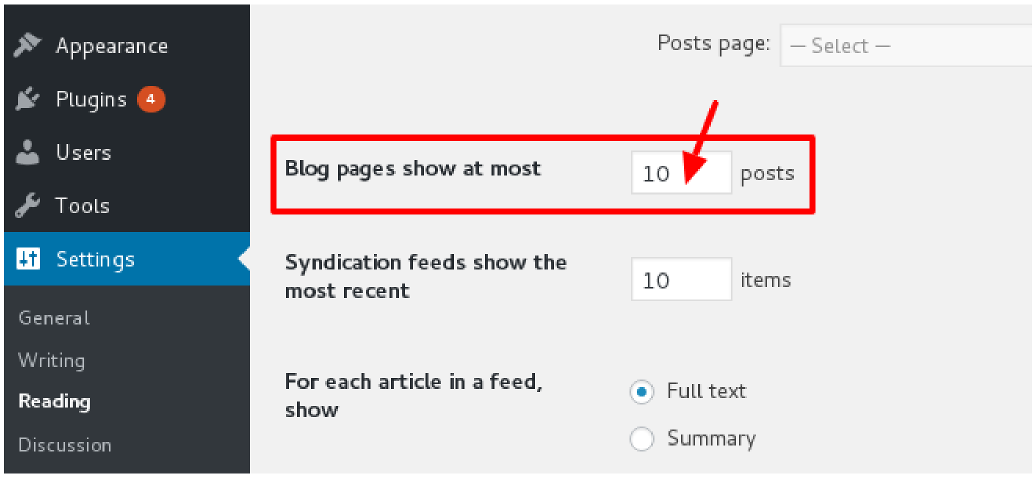

Change WordPress Default Category
0
New
Dashboard
Home
Updates
Posts
All Posts
Add New Post
Categories
Tags
Media
Optimole
Pages
Comments
Neve
Appearance
Templates Cloud
Plugins
Users
Tools
Settings
Dashboard
come to WordPress!
about the 6.6.2 version.
Author rich content with blocks and patterns
Block patterns are pre-configured block layouts. Use them to get inspired or create new pages in a flash.
Add a new page
Start Customizing
Configure your site's logo, header, menus, and more in the Customizer.
Open the Customizer

Figure 41: Add new category

Example: Write a short welcome post for your blog.

Step 3: Viewing Pages and Posts

- To see your content, click View Page **or** View Post after publishing.
- Pages usually appear in the site's navigation menu.
- Posts appear in blog/news sections, with the newest post at the top.



Blog pages show at most 10 posts

Syndication feeds show the most recent 10 items

For each article in a feed, show [✓ Full text] Summary

Figure 42: Blog page setting

Step 4: Managing Content

- Edit: Open the page/post and click Edit to make changes.
- Update: Save changes by clicking Update.
- Delete: Move unwanted content to Trash.

7.4 Customizing a Website Using Themes and Menus

After creating basic pages and posts, the next step is to make your website look professional and easy to navigate. In WordPress, you can do this by:

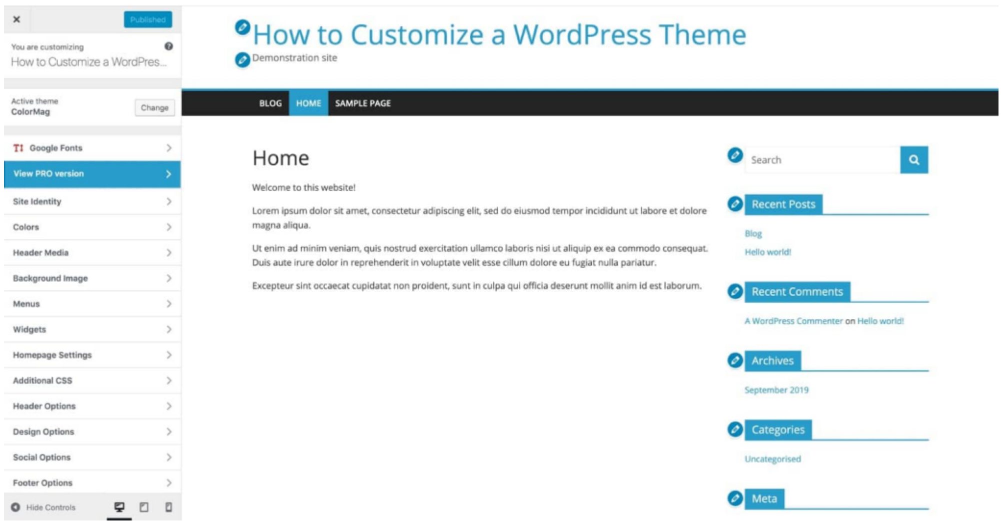
- Changing the **theme** (the design/style of your website).
- Creating **menus** (the navigation links that help visitors move around your site).

Step 1: Customizing with Themes

1. From the Dashboard, go to Appearance → Themes**.**
2. Click Add New to browse free themes from the WordPress library.
3. Preview a theme to see how it looks on your site.

4. Click Install → Activate to apply the theme.
5. Go to Appearance → Customize to adjust:
   - Colors and fonts.
   - Header and background images.
   - Layout of homepage and other sections.

Example: Choose a simple theme and change the site's header image to match your content.



Home

Welcome to this website!

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure 43: Wordpress theme customize

Step 2: Creating and Managing Menus

1. From the Dashboard, go to Appearance → Menus**.**
2. Click Create a New Menu and give it a name (e.g., *Main Menu*).
3. Add items to the menu: select Pages**,** Posts**,** or Custom Links**,** then click Add to Menu**.**
4. Arrange the items by dragging and dropping them in the desired order.
5. Choose where the menu should appear (e.g., *Primary Menu* at the top of the site).
6. Click Save Menu.

Example: Add *Home*, *About*, and *Contact* pages to your main navigation menu.

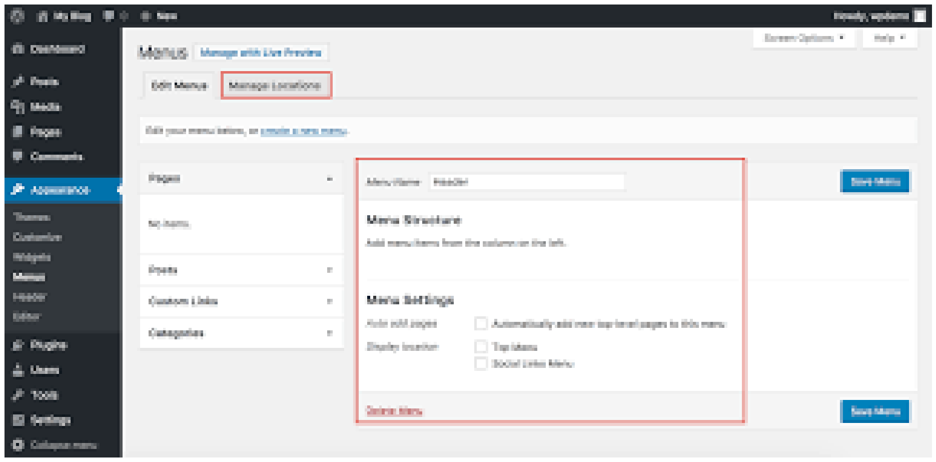


Figure 44: view customization

Step 3: Viewing Customizations

- Open your website in a new tab to check how the new theme and menus look.
- Adjust as needed (e.g., reorder menu items or change theme colors).

Exercise

Building a Basic WordPress Website

Instructions:

Step 1: Access WordPress Dashboard

1. Log in to your WordPress site via http://localhost/mywebsite/wp-admin (local) or http://www.yourwebsite.com/wp-admin (online).
2. Explore the dashboard menu to familiarize yourself with Posts, Pages, Media, Appearance, Plugins, and Settings.

Step 2: Create Pages

1. Go to Pages → Add New.
2. Create a Home Page with a welcome message.
3. Create an About Page with a short introduction about yourself or your website.
4. Click Publish for each page.

Step 3: Create a Post

1. Go to Posts → Add New.
2. Write a short blog post titled “My First Post.”
3. Add some text, and optionally an image.
4. Assign a category and tags, then click Publish.

Step 4: Customize Your Website

1. Go to Appearance → Themes → Add New.
2. Select a free theme and click Activate.
3. Go to Appearance → Customize and modify:
   - Site title
   - Header image
   - Colors and fonts

Step 5: Create a Menu

1. Go to Appearance → Menus → Create New Menu.
2. Add the Home and About pages to the menu.
3. Set it as the Primary Menu and click Save Menu.

Step 6: View Your Website

1. Open your site in a new tab.
2. Check the homepage, About page, blog post, and menu navigation.
3. Adjust as needed in the dashboard.

Summary

In this session, learners were introduced to WordPress, a user-friendly platform for creating websites without coding. They learned how to access and navigate the WordPress Dashboard, the central hub for managing content, settings, and website features. The session covered creating basic pages for static content and posts. Learners also explored how to customize their website's appearance using themes and improve navigation by creating menus. By the end of the session, participants gained practical skills to set up a simple website, add content, and personalize its design, building a foundation for further WordPress development.

Review Questions

1. What is WordPress?
   A. A programming language for building websites
   B. A content management system (CMS) used to create websites
   C. A type of web hosting service
   D. A social media platform

2. Which of the following is used for creating permanent, static content on a WordPress website?
   A. Posts
   B. Pages
   C. Plugins
   D. Widgets
3. Which of the following correctly describes the difference between a Post and a Page in WordPress?
   A. Posts are static content; Pages are for blogs or news updates.
   B. Pages are static content; Posts are for blogs or news updates.
   C. Posts and Pages are exactly the same and used interchangeably.
   D. Pages are only used for plugins.
4. Which WordPress feature allows you to change the design and layout of your website?
   A. Plugins
   B. Menus
   C. Themes
   D. Posts
5. What is the main purpose of a menu in WordPress?
   A. To add images and videos
   B. To navigate visitors easily across different pages
   C. To create blog posts
   D. To install plugins

Capstone Project: Web-Based Personal Portfolio

1. Project Title: "Design and Build Your Own Digital Portfolio Website"
2. Purpose: This capstone project helps trainees apply their digital and creative skills by building a personal portfolio website that showcases their background, skills, and sample work. They will design the website's frontend and set up a simple backend to store content.
3. Learning Objectives: By completing this capstone, trainees will be able to:
   - Create a personal web portfolio with a clean and professional layout.
   - Use HTML, CSS, and simple JavaScript for the frontend.
   - Implement a lightweight backend for storing or serving data.

- Demonstrate safe, ethical, and creative use of digital tools.

4. Project Description: Each trainee will develop a simple personal website containing these sections:
    I. Home / About Me – short introduction and photo.
    II. Skills – key digital and professional competencies.
    III. Projects or Sample Work – short descriptions of 2–3 works (can be PDFs, screenshots, or links).
    IV. Contact – email or LinkedIn.

    Functional Requirements
    - A frontend page (HTML/CSS/JS) displaying all sections.
    - A backend that serves data from a database (e.g., SQLite).
    - The site should load portfolio data dynamically from the backend.
    - Optional: a simple admin page to update text (bonus).
5. Step-by-Step Guidance

Table 14 Capstone Project Guidance

Stage	**Tasks**	**Outputs**
1. Planning	Sketch your site layout; list sections and content.	Draft structure or mock-up
2. Frontend Design	Build pages using HTML, CSS, and JS	Functional webpage
3. Backend Setup	Building the foundational, server-side logic and data management systems	Structured Database, Functional API and Secure Server Environment
4. Integration	Connect frontend with backend (fetch API data).	Connected site
5. Report & Presentation	Write a short report (2 pages) and present your website.	PDF + 5 min demo

Module 4: Advanced Problem-solving

Introduction

In today's world, digital systems are at the heart of nearly every activity. Hospitals, schools, universities, police stations, NGOs, businesses, and government offices all rely on devices, networks, cloud platforms, and collaboration tools to function smoothly. When these systems fail through software glitches, hardware breakdowns, network disruptions, or security threats, the consequences can affect not just individuals but entire groups, institutions, or communities.

Advanced digital literacy goes beyond knowing how to use technology. It requires the ability to identify problems, separate symptoms from root causes, troubleshoot effectively, and apply sustainable solutions. These skills are essential for both professionals and students to reduce disruption, maintain productivity, support peers, and adapt to evolving digital environments.

By the end of this module, you will be equipped to combine critical thinking, creativity, and digital competence to create meaningful solutions in a tech-driven environment.

Session 1: Solving Technical Problems

Introduction

Dear Learners, this session is all about solving technical problems smartly, efficiently, and with real-world impact.

Modern digital systems are highly interconnected. Devices, operating systems, applications, networks, and cloud platforms work together as a chain of dependencies, where a single fault can affect multiple systems and disrupt organizational workflows. Understanding the difference between symptoms and root causes is critical for effective problem-solving. Addressing only the symptom may provide a temporary fix, but resolving the root cause ensures long-term stability and efficiency.

Technical problems can take many forms, including software crashes or incompatible updates, hardware failures, network disruptions, cloud service downtime, and security threats such as malware or unauthorized access, all of which can impact system functionality and data integrity. Developing the ability to systematically identify, diagnose, and resolve these issues is essential for maintaining productivity, reducing disruptions, and supporting the smooth operation of digital environments.

Learning Objectives

By the end of this session, learners will be able to:

- Distinguish between symptoms and root causes
- Identify common digital system issues
- Apply structured troubleshooting methods

Content Outline:

1.1 Introduction to Advanced Problem-Solving

Modern organizations rely heavily on interconnected digital systems, including devices, applications, networks, and cloud platforms. These systems support critical operations in hospitals, schools, businesses, government offices, and NGOs. When a fault occurs, its impact can ripple across multiple processes, affecting productivity, decision-making, and service delivery.

Effective problem-solving in digital environments requires more than just technical know-how. Professionals must be able to distinguish between symptoms and root causes, analyze the scope and impact of a problem, and apply structured methods to resolve it.

Symptoms vs. Root Causes:

- *Symptoms* are the visible signs of a problem, such as slow performance, system crashes, or connectivity issues.
- *Root causes* are the underlying issues triggering these symptoms, which may include outdated software, hardware degradation, misconfigured networks, or security vulnerabilities.

Understanding the difference is critical because addressing only the symptom provides temporary relief, whereas identifying and resolving the root cause ensures long-term stability and efficiency.

This section sets the foundation for the session by emphasizing the importance of systematic problem analysis and the need for structured troubleshooting in maintaining reliable digital systems.

1.2 Common Technical Issues

Before exploring common technical issues in digital systems, take a moment to reflect on the challenges you have encountered in your own experience. Thinking about these problems will help you relate the concepts to real-life situations and prepare you to apply troubleshooting strategies effectively.

Consider the following question:

- What types of technical problems have you experienced in your school, workplace, or personal digital environment?
- Which issues occurred most frequently: software, hardware, network, cloud, or security-related?
- How did these issues affect productivity, workflow, or access to important data?

Exercise:

- List 3–5 technical issues you have personally encountered.
- Briefly describe how each issue impacted your work or learning.
- Reflect on whether the root cause was easy to identify or if it took time to troubleshoot.

This self-reflection helps you connect your experiences to the upcoming section on common technical issues, making it easier to understand, categorize, and address them effectively.

Digital systems can experience failures in a variety of ways, which can significantly disrupt productivity, workflow, and the integrity of organizational data. These issues may arise at different levels, including software malfunctions, hardware failures, network disruptions, cloud service interruptions, and security threats, each affecting the smooth operation of processes in distinct ways.

Software Issues

Software problems are among the most common technical challenges in digital environments. They arise when applications fail to function as intended, preventing users from completing tasks efficiently. Some of the common software issues include:

- Application crashes, freezes, or unexpected errors
- Incompatible updates or missing components
- Corrupted or inaccessible files

Hardware Failures

Hardware failures occur when physical components of a digital system malfunction or degrade over time, affecting overall system stability and performance. Common hardware issues include:

- Overheating components such as CPUs or GPUs
- Degraded or failing hard drives, memory modules, or storage devices
- Malfunctioning peripheral devices, including printers, scanners, or input devices

Network Issues

Network issues arise when connectivity between devices, servers, or external networks is disrupted, misconfigured, or slowed. These problems can block access to critical systems, hinder communication, and affect multiple users simultaneously. Common network issues include:

- Connectivity problems, including Wi-Fi, LAN, or ISP outages
- Misconfigured routers, switches, or firewalls
- Slow network speeds, dropped connections, or intermittent access

Cloud Service Problems

Cloud service problems occur when online platforms or storage solutions fail to operate as expected, disrupting access to files, applications, or collaborative tools. These issues can affect multiple users simultaneously, especially in organizations that rely on cloud services for remote work or data sharing. Common cloud service problems include:

- Downtime or unavailability of cloud applications

- Synchronization failures between devices or users
- Permission errors restricting access to shared resources

Security Threats

Security threats pose risks to both the functionality of digital systems and the integrity of organizational data. These issues can arise from malicious attacks or unauthorized access, and they often require immediate attention to prevent operational disruption or data breaches. Common security threats include:

- Malware infections, viruses, ransomware, or spyware
- Phishing attacks and social engineering attempts
- Unauthorized access attempts, brute-force attacks, or weak authentication

Understanding the different categories of technical issues helps professionals prioritize responses effectively, ensuring that the most critical problems are addressed first. It also enables the use of structured troubleshooting methods and appropriate diagnostic tools, allowing for accurate identification of root causes and efficient problem resolution.

1.3 Troubleshooting Methodology

Troubleshooting is a structured process used to identify, diagnose, and resolve technical problems effectively. A systematic approach ensures that issues are addressed at their root cause rather than just treating symptoms, minimizing disruption and preventing recurrence.

A. Observe and Collect Data

The first step in troubleshooting is to carefully observe the issue and gather all relevant information before attempting any fixes. This step includes the following tasks:

- Carefully examine the problem and gather relevant evidence.
- Collect error messages, system logs, screenshots, and user reports.
- Avoid jumping to conclusions; focus on understanding the symptom fully.

Example: Note when the problem occurs, which devices or users are affected, and any patterns in the issue.

B. Diagnose the Root Cause

Once sufficient data is collected, the next step is to analyse it to determine the underlying cause of the problem. This step includes the following tasks:

- Analyse the collected data to identify the root cause.
- Use structured frameworks such as the 5 Whys or Root Cause Analysis (RCA).
- Separate symptoms from actual faults to ensure that solutions address the real issue.

5 Whys

- It is a simple, quick method: ask "Why?" Up to five times until the root cause is revealed.
- Best for straightforward problems.
- Example:

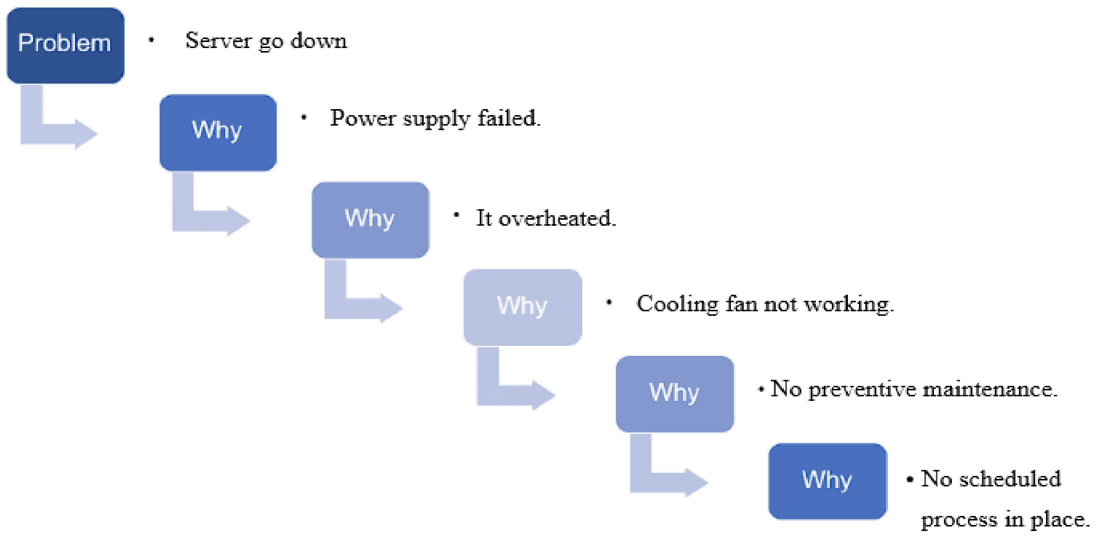

Problem
• Server go down
Why
• Power supply failed.
Why
• It overheated.
Why
• Cooling fan not working.
Why
• No preventive maintenance.
Why
• No scheduled process in place.

Fig: 5 Whys" technique

Root Cause Analysis (RCA)

- A broader, structured approach that may use tools like fishbone diagrams, fault tree analysis, or Pareto charts.

- Goes beyond just asking “Why?” by mapping multiple possible causes (technical, human, process, organizational).
- Best for complex or recurring issues**.**

C. Develop and Test Solutions

After identifying the root cause, develop multiple potential solutions and test them systematically. This step includes the following tasks:

- Identify multiple possible solutions based on the diagnosed cause.
- Test options in a controlled, least-disruptive manner before full implementation.
- Verify that the chosen solution resolves the problem without introducing new issues.

Example: Restart a service, update software, or adjust configuration settings to confirm effectiveness.

D. Implement, Monitor, and Document

The final step is to implement the chosen solution, monitor its effectiveness, and document the process for future reference. This step includes the following tasks:

- Apply the selected solution fully and monitor the system to ensure stability.
- Document the problem, root cause, and resolution for organizational learning.
- Share documentation to reduce repeated troubleshooting and support team knowledge.

Example: Record in an IT logbook or knowledge base how the problem was identified and resolved.

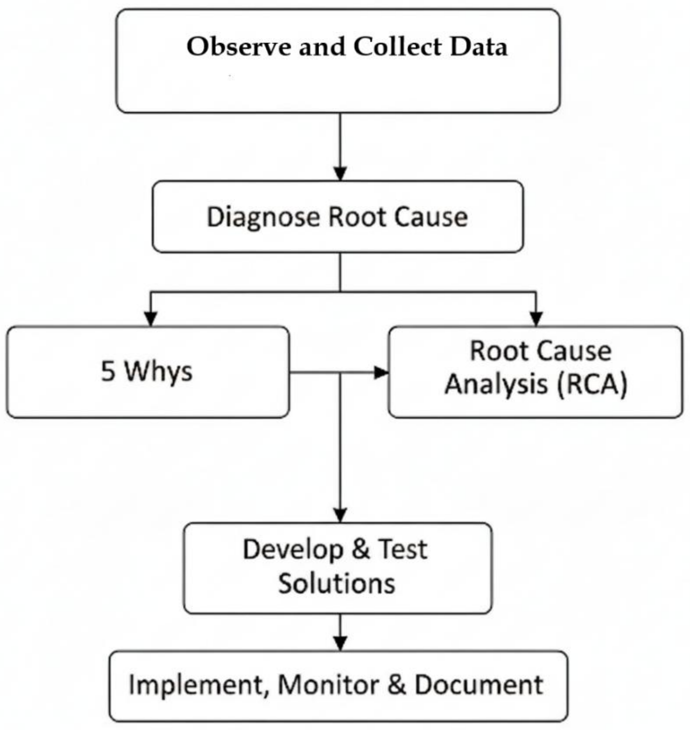


Figure 45: Troubleshooting methodology flowchart

Effective troubleshooting relies on following systematic principles rather than relying on trial and error. The focus should always be on identifying the root cause of a problem instead of only treating visible symptoms, as this ensures that solutions are lasting and effective. Using appropriate tools and structured frameworks supports accurate diagnosis and prevents missteps during the process. Finally, documenting and communicating solutions not only strengthens individual learning but also contributes to organizational knowledge, helping teams respond more efficiently to future issues.

1.4 Diagnostic Tools and Monitoring Systems

Diagnostic and monitoring tools are essential for identifying issues, understanding system behaviour, and maintaining stability. They allow IT professionals to detect problems early, troubleshoot effectively, and prevent small issues from escalating into major failures. Here's the key Categories of Diagnostic Tools and Monitoring Systems:

A. System-Level Tools

These tools provide insights into how devices and applications are performing, helping to detect issues that may affect overall system stability. The most common advantages of using system-level tools include:

- Monitoring hardware and software performance.
- Detecting application crashes, CPU spikes, memory issues, and process failures.
- Offering built-in utilities that make it easier to identify and resolve performance bottlenecks.

Examples:

- Windows: Task Manager, Performance Monitor, Event Viewer
- Linux/macOS: top, dmesg, Activity Monitor

B. Network Diagnostic Tools

These tools help in testing connectivity and analysing the flow of data across networks, making it easier to detect performance or configuration issues. The most common advantages include:

- Testing network connections and communication paths.
- Identifying latency, packet loss, and misconfigurations.
- Supporting detailed traffic analysis for troubleshooting complex issues.

Examples:

- Basic tools: Ping, Traceroute
- Advanced tools: Wireshark

C. Security Tools

Security tools are essential for safeguarding systems against threats that can compromise functionality or data integrity. The most common advantages include:

- Detecting malware, unauthorized access attempts, and suspicious activities.
- Protecting data confidentiality, availability, and integrity.

- Supporting compliance with organizational or regulatory security standards.

Examples:

- Basic tools: Antivirus and anti-malware programs
- Advanced tools: SIEM platforms such as Splunk, QRadar

D. Monitoring Systems

Monitoring systems provide real-time visibility into the health of IT infrastructure, helping prevent problems before they escalate. The most common advantages include:

- Tracking system, network, and application performance continuously.
- Sending alerts when critical thresholds are exceeded to minimize downtime.
- Providing reports and analytics for long-term performance optimization.

Examples:
Nagios, Zabbix, PRTG

Summary

This session introduced the importance of solving technical problems systematically by distinguishing symptoms, such as slow performance or system crashes, from root causes like outdated software or hardware degradation. Learners explored common technical issues across software, hardware, networks, cloud platforms, and security, recognizing how each can disrupt productivity and compromise data integrity. The troubleshooting methodology emphasized four structured steps: observing and collecting data, diagnosing the root cause, developing and testing potential solutions, and implementing while monitoring and documenting results for future learning.

The session also covered key diagnostic and monitoring tools, including system utilities like Task Manager, network tools such as Ping and Wireshark, and security solutions from antivirus programs to SIEM platforms, while monitoring systems like Nagios and Zabbix provide real-time alerts and performance tracking. Together, these tools support structured troubleshooting, documentation, and overall system reliability.

Review Questions

1. Why is it important to distinguish between symptoms and root causes?
   A. Because symptoms are always easy to fix
   B. To ensure long-term stability and avoid temporary solutions
   C. So, you can ignore system errors
   D. To focus only on software updates
2. What is the first step in structured troubleshooting?
   A. Implementing the solution
   B. Observing and collecting data
   C. Running antivirus scans
   D. Restarting all devices
3. Which tool is primarily used for monitoring system-level performance on Windows?
   A. Wireshark
   B. Event Viewer
   C. Nagios
   D. Splunk
4. What is the main purpose of monitoring systems like Nagios or Zabbix?
   A. Writing code for applications
   B. Providing real-time visibility, alerts, and performance tracking
   C. Replacing IT staff
   D. Installing software updates automatically
5. Which principle is essential for effective troubleshooting?
   A. Trial and error only
   B. Focus on symptoms without analysis
   C. Systematic problem-solving and documenting solutions
   D. Ignoring documentation for faster fixes

Session 2: Identifying Needs and Technological Responses

Introduction

Dear Learners, in this session, you'll learn how to spot challenges and match them with smart technological solutions!

In this session, we will focus on developing the skills needed to recognize digital and organizational needs and respond with the right technological solutions. Together, we'll explore how to analyse workflows using structured methods, identify gaps in efficiency, communication, data management, and security, and then match those needs with appropriate digital tools.

The session goes beyond simply choosing technology, it emphasizes how to tailor and optimize tools so they fit real organizational contexts. We'll look at practical examples of cloud services, collaboration platforms, automation tools, and accessibility technologies, giving you hands-on strategies, you can apply directly to workplace challenges.

Learning Objectives

By the end of this session, learners will be able to:

- Analyse workflows and identify gaps
- Evaluate and select appropriate digital tools
- Apply customization and optimize technological solutions

Content Outline:

2.1. Analysing Workflows and Identifying Gaps
2.2. Evaluating and Selecting Digital Tools
2.3. Customizing, Optimizing, and Monitoring Solutions

2.1 Analysing Workflows and Identifying Gaps

Analyzing workflows helps organizations see how tasks, people, and tools interact in daily operations. By mapping processes step by step, it becomes easier to spot bottlenecks, redundancies, or inefficiencies that affect performance. This analysis lays the groundwork for understanding needs analysis and highlights common organizational gaps that must be addressed to improve productivity and outcomes.

Understanding Needs Analysis

A needs analysis is the process of systematically identifying the gaps, challenges, and requirements within an organization or workflow. Its purpose is to ensure that any technological solution implemented addresses real problems rather than creating additional complexity or inefficiency.

Why Needs Analysis Matters:

- Prevents investing in tools that are unnecessary or poorly suited.
- Helps prioritize areas where technology will have the greatest impact.
- Supports informed decision-making for digital transformation projects.
- Improves alignment between technology solutions and organizational goals.

Structured Frameworks for Needs Analysis:

A. Workflow Mapping:
   - Visual representation of tasks, processes, and interactions.
   - Helps identify bottlenecks, redundancies, and inefficiencies.

B. Process Charts / Flowcharts:
   - Step-by-step diagrams showing inputs, actions, and outputs in a process.
   - Useful for understanding task dependencies and identifying areas for automation.

C. SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats):
   - Evaluates internal and external factors affecting workflows or technology use.
   - Strengths and weaknesses focus on internal processes; opportunities and threats focus on external environment and risks.
   - Example: assessing an existing project management system to determine strengths (easy task assignment), weaknesses (poor reporting), opportunities (integration with communication tools), and threats (security vulnerabilities).

Figure 46: SWOT Analysis

Common Organizational Gaps

In any organization, workflows and digital processes can be hindered by gaps that reduce productivity, compromise data integrity, or create security risks. Recognizing these gaps is critical to implementing effective technological solutions. Common organizational gaps include:

A. Efficiency Gaps

Efficiency gaps arise when workflows are slow, redundant, or unnecessarily complex. These gaps can reduce productivity, create delays, and cause frustration among team members. Identifying and addressing efficiency gaps is essential for improving workflow speed, minimizing duplication, and enhancing overall team performance.

Examples:

- Repetitive manual tasks that could be automated, such as data entry or report generation.
- Delays caused by unnecessary approval steps or overly long review processes.
- Overlapping responsibilities where multiple employees perform the same tasks, causing confusion or duplication.

Impact:

- Lower overall productivity and slower project completion.
- Increased employee frustration and reduced morale.
- Higher risk of errors due to repetitive or manual processes.

Solution Approach:

- Introduce workflow automation to handle repetitive tasks efficiently.
- Use task management systems to assign responsibilities clearly and track progress.
- Apply process optimization tools to streamline workflows and eliminate bottlenecks.

B. Communication Gaps

Communication gaps occur when the flow of information within a team or organization is interrupted, unclear, or inconsistent. These gaps hinder collaboration, misunderstandings, and can result in mistakes or missed opportunities.

Examples:

- Missed emails or messages due to lack of a central communication platform.
- Limited collaboration channels that prevent effective teamwork.
- Unclear reporting lines causing misunderstandings about responsibilities.

Impact: Misaligned priorities, errors, missed deadlines, decreased team cohesion.

Solution Approach: Adopt collaboration platforms, instant messaging tools, and project management dashboards.

C. Data Management Gaps

Data management gaps occur when an organization handles data inefficiently, inconsistently, or insecurely. These gaps can affect the accuracy, accessibility, and reliability of information, making it harder to make informed decisions, maintain operational efficiency, and comply with regulatory requirements.

Examples:

- Inconsistent data entry practices leading to errors.
- Missing or outdated backups, risking data loss.
- Poor file organization and retrieval systems causing delays in finding information.

Impact: Reduced decision-making accuracy, operational inefficiencies, and potential compliance issues.

Solution Approach: Use centralized databases, automated data validation, cloud storage solutions, and backup scheduling.

D. Security Gaps

Security gaps are weaknesses in an organization’s systems, processes, or practices that leave sensitive information and digital assets vulnerable to unauthorized access, data breaches, or cyberattacks. These gaps can compromise confidentiality, integrity, and compliance with legal or regulatory requirements.

Examples:

- Weak access controls allowing unauthorized users to access sensitive data.
- Unencrypted data, increasing vulnerability to breaches.
- Non-compliance with regulatory requirements (e.g., GDPR, HIPAA).

Impact: Data breaches, financial loss, legal penalties, reputational damage.

Solution Approach: Implement robust access controls, encryption, multi-factor authentication, and regular compliance audits.

2.2 Evaluating and Selecting Digital Tools

Key Factors in Selecting Digital Tools

When selecting a digital solution for your organization, it is essential to evaluate key criteria to ensure the tool meets immediate needs while supporting long-term efficiency, security, seamless integration, and accessibility. The following critical factors should be considered when choosing a digital solution:

A. Functionality: Does the tool effectively address the specific problem or workflow gap? Example: A project management tool should track tasks, deadlines, and responsibilities clearly.

B. Scalability: Can the tool grow with the organization's evolving needs without requiring a full replacement? Example: Cloud storage that can expand as the company accumulates more data.
C. Security: Does the tool comply with data protection standards and regulatory requirements? Example: Encrypted storage for sensitive customer information.
D. Interoperability: Can the tool integrate seamlessly with existing systems and platforms? Example: A CRM system that works with the organization's email and accounting software.
E. Accessibility: Can it accommodate diverse users, including individuals with disabilities? Example: Tools that support screen readers, keyboard shortcuts, and voice commands.

Types of Digital Tools

Digital tools are software and platforms designed to help organizations address specific needs, streamline workflows, and improve productivity. Understanding the different types of tools available allows you to select solutions that best fit your organizational goals, enhance collaboration, automate processes, and ensure accessibility for all users. The main categories of digital tools include:

A. Cloud Solutions: Provide centralized storage, remote access, automated backups, and support for distributed teams. Examples: Amazon Cloud, Google Cloud, Microsoft Cloud, and others.
B. Collaboration Platforms: Facilitate team communication, project tracking, and coordination. Examples: Microsoft Teams, Slack, Trello, Asana
C. Automation Tools: Reduce repetitive tasks, streamline approvals, send notifications, and automate data processing. Examples: Zapier, Power Automate, IFTTT
D. Accessibility Tools: Ensure inclusive use for all employees, including those with visual, auditory, or mobility impairments. Examples: Screen readers, high-contrast modes, voice commands, keyboard shortcuts.

Table 15: Example on a Comparison of Digital Collaboration Tools

Key Factors	**Microsoft Teams**	**Slack**	**Asana**
Primary Purpose	A comprehensive collaboration hub for communication and productivity, deeply integrated with Microsoft 365.	A communication-first platform built for real-time messaging, file sharing, and team chat.	A dedicated project management tool focused on organizing and tracking tasks, projects, and workflows.
Functionality	Offers a wide range of features including chat, video conferencing, file sharing, and real-time co-authoring of documents.	Excels in instant messaging and channel-based communication. It supports voice/video calls.	Provides robust project management features such as task lists, Gantt charts, and customizable workflows.
Scalability	Designed for and excels in large enterprises, with extensive administrative controls and security features that support large-scale deployment.	Highly scalable for teams of all sizes, from small startups to large enterprises, with custom plans and features for enterprise-level use.	Easily scalable from small teams to large organizations, with tiered pricing that unlocks advanced features as your needs grow.
Security	Backed by Microsoft's robust security infrastructure, offering enterprise-grade features and is compliance-ready.	Offers strong security features, including data encryption and compliance certifications.	Provides a secure platform with encryption for data at rest and in transit.
Interoperability	Deeply and seamlessly integrated with the Microsoft 365 ecosystem.	An extensive app directory with over 2,400 integrations.	Integrates with over 200 popular applications.

Aligning Digital Tools with Organizational Needs

When implementing a digital tool, consider both its immediate benefits and long-term impacts, including effects on efficiency, security, scalability, and overall organizational performance.

A. Immediate Effectiveness: The tool should directly address the identified workflow gap or problem, delivering measurable improvements in productivity and communication.
B. Future Growth: Tools should be scalable to accommodate expansion, new users, or additional features as organizational needs evolve.
C. Security and Compliance: Solutions must protect sensitive data, comply with regulations, and minimize the risk of breaches.
D. Interoperability: Ensure the tool integrates smoothly with existing systems to avoid disruptions and additional configuration.
E. Accessibility: Promote an inclusive digital environment by selecting tools that all employees can use comfortably and effectively.

2.3 Customization and Optimization of Digital Environments

Implementing a digital tool is only the first step. True value comes when organizations adapt, personalize, and refine these tools so they align with unique workflows, employee needs, and organizational goals.

Digital tools are rarely perfect straight out of the box. While most offer strong default features, they are designed for a broad audience rather than the specific needs of a particular organization. Without customization, teams often find themselves forced to adapt their workflows to fit the tool, which can create inefficiencies or resistance to adoption.

Customization ensures that technology is adapted to the organization’s workflows, culture, and goals. It aligns tools with the way people actually work rather than imposing rigid processes. Optimization goes a step further. It focuses on continuous improvement, regularly refining and adjusting digital tools so they remain effective, efficient, and relevant as the organization evolves.

Key Benefits of Customization and Optimization

A. Higher Efficiency and Productivity: Tools designed around actual workflows eliminate wasted steps and speed up processes.
B. Greater User Satisfaction and Adoption: When tools feel intuitive and tailored, employees are more likely to embrace and fully utilize them.
C. Reduced Redundancy and Bottlenecks: Custom workflows prevent duplication of tasks and minimize delays caused by misaligned processes.
D. Scalability and Flexibility for Future Growth: Well-customized systems can expand as the organization grows, avoiding the need for complete overhauls.

Principles of Customization

Customization is not just about changing settings; it is about shaping digital environments so they fit seamlessly into organizational workflows, user needs, and long-term strategies. The following principles guide effective customization:

A. Tailoring to Workflows: Digital tools should reflect the way an organization actually operates. By aligning tools with established processes, teams can reduce friction and improve efficiency.
B. User-entered Design: Employees are more likely to adopt and fully utilize tools when they feel intuitive and relevant to their roles. A user-entered approach ensures customization is guided by staff needs.
C. Integration with Existing Systems: Digital tools should not operate in isolation. Seamless integration reduces duplication, strengthens data accuracy, and ensures smoother workflows.
D. Accessibility Adjustments: An inclusive digital environment allows every employee to work effectively, regardless of ability or circumstance. Accessibility should be a core part of customization.
E. Scalability and Flexibility: Customization should not lock systems into rigid configurations. Instead, it should prepare tools to evolve with organizational needs.

Strategies for Optimization

Once digital tools are implemented and customized, organizations must ensure they remain effective, efficient, and relevant over time. Optimization is a continuous process that requires monitoring, refinement, and user support. The following strategies help organizations maximize the value of their digital environments:

A. Performance Monitoring: Optimization begins with evidence. Monitoring how tools are used provides insight into adoption rates, user behaviours, and areas for improvement.
B. Regular Reviews and Updates: Technology evolves rapidly, and so do organizational needs. Regular reviews ensure digital environments remain aligned with strategic goals.
C. Automation Enhancements: Optimization often means reducing manual effort and repetitive tasks through automation.
D. Training and Support: No tool can reach its full potential without confident users. Continuous training and support ensure staff maximize the benefits of customized features.

Summary

In this session, learners will develop skills to recognize organizational and digital needs and respond with appropriate technological solutions. The session emphasizes analysing workflows to identify gaps in efficiency, communication, data management, and security, and selecting digital tools such as cloud solutions, collaboration platforms, automation tools, and accessibility technologies that address these gaps. Learners will also explore customizing and optimizing tools to align with real workflows, enhance productivity, ensure security, support accessibility, and allow scalability. Continuous monitoring, updates, and user support are highlighted to maintain the effectiveness of implemented solutions.

Review Questions

1. What is the main purpose of a need's analysis in an organization?
   A. To buy the latest technology
   B. To identify gaps and challenges for effective solutions
   C. To reduce employee training
   D. To replace all existing workflows
2. Which of the following is NOT considered a common organizational gap?
   A. Efficiency
   B. Communication
   C. Marketing strategy
   D. Security
3. Which tool helps visualize tasks, processes, and interactions in a workflow?

   A. SWOT analysis
   B. Workflow mapping
   C. Accessibility tools
   D. Cloud storage

4. When selecting digital tools, which factor ensures the tool grows with organizational needs?
   A. Security
   B. Scalability
   C. Accessibility
   D. Interoperability
5. Which type of digital tool helps reduce repetitive tasks and streamline approvals?
   A. Cloud solutions
   B. Collaboration platforms
   C. Automation tools
   D. Accessibility tools
6. Why is customization of digital tools important?
   A. To make tools look visually appealing
   B. To fit tools to workflows, user needs, and organizational goals
   C. To reduce software costs
   D. To avoid using automation
7. Which strategy is part of optimizing digital tools after implementation?
   A. Ignoring user feedback
   B. Continuous monitoring, updates, and training
   C. Only installing new tools
   D. Reducing workflow documentation

Session 3: Digital Creativity and Competence Mapping

Introduction

Dear Learners, in this session, you'll explore Digital Creativity and learn how to map your skills and competencies to bring your ideas to life!
In this session, we will bring together two essential dimensions of digital growth: creativity in using technology and awareness of our own digital strengths and gaps. While technology provides powerful tools, the true value comes from how we apply them in innovative, flexible, and meaningful ways to solve problems, improve processes, and unlock new opportunities.

At the same time, digital creativity is only sustainable if we continually reflect on our own skills and competencies. By identifying where our digital knowledge is strong and where gaps exist, we can target learning, adopt the right strategies, and ensure that our creative use of technology is both effective and future-ready.

This session emphasizes a practical balance: experimenting with digital tools in creative ways, while also engaging in structured self-assessment to map personal and organizational digital competence. Together, these skills will help learners become not only innovators but also adaptive professionals who know how to grow and thrive in a rapidly evolving digital environment.

Learning Objectives

By the end of this session, learners will be able to:

- Use digital technologies creatively
- Identify digital competence strengths and gaps
- Evaluate digital skills and plan growth

Content Outline:

3.1 Digital Creativity
3.2 Identifying digital competence strengths and gaps
3.3 Enhancing digital skills and planning growth

3.1 Digital Creativity

What is Digital Creativity?

Digital creativity means thinking differently about how you use technology. It’s not just knowing how tools work-it’s about applying them in new, flexible, and effective ways to solve problems, improve processes, and enhance collaboration.

Digital creativity is important because technology reaches its full potential when it is adapted to meet real workflow challenges rather than just being used theoretically. Creative approaches can streamline repetitive tasks, saving time and increasing efficiency. By applying tools in innovative ways, teams can improve collaboration, communication, and coordination. Moreover, imaginative use of digital technologies fosters innovation, opening opportunities to discover new solutions and enhance overall organizational performance.

Activity:

Think about one task you perform daily (e.g., reporting, scheduling, or communication).

- How do you currently complete it?
- Could digital tools help you do it differently or more efficiently?
- Note your ideas in a learning journal or discussion board to revisit later.

Digital creativity is about combining imagination with technology to work smarter, not harder, while opening doors to new ways of solving problems.

Principles of Creative Technology Use

To use digital tools creatively, learners need to understand several core concepts that guide innovative and flexible application:

- Exploration and Experimentation - Creativity begins by testing features, integrations, and new ways of using digital tools. Experimenting helps you discover capabilities you might not have considered.

- Problem-Solving with Creativity - Identify workflow or collaboration challenges and think about how technology could solve them in innovative ways. Avoid sticking to standard procedures; explore alternative approaches.
- Flexible Thinking - Instead of forcing workflows to fit tools, adapt tools to fit your processes. Flexibility allows tools to enhance productivity without adding unnecessary complexity.
- Collaboration Enhancement - Technology is most effective when it improves teamwork. Creative use of digital platforms can strengthen communication, coordinate tasks, and support shared problem-solving.

Exercise

- Select a tool you use regularly. Brainstorm three new ways it could help solve a task or improve team collaboration.
- Share your ideas with peers or note them in your learning journal for reflection.

Understanding and applying these concepts helps learners not just use digital tools-but use them creatively and effectively to improve workflows, collaboration, and outcomes.

Creative Applications of Digital Tools

Applying digital tools creatively means looking beyond standard usage and experimenting with features, integrations, and alternative approaches that improve efficiency and team coordination.

For example:

- Project management apps like Trello or Asana can automate task reminders and notifications, reducing missed deadlines and freeing time for strategic work.
- Cloud storage tools combined with automation apps like Zapier can minimize repetitive data-entry, improve accuracy, and save time.
- Interactive dashboards created with Power BI or Google Data Studio make reports easier to understand and share. Collaboration platforms such as Teams, Slack, or Trello can also be used creatively to support brainstorming, idea-sharing, and team coordination.

Exercise

- Identify a workflow or task in your daily work that feels inefficient or repetitive.
- Brainstorm two or three ways a digital tool could improve or simplify the task. Consider questions such as:
    - Could automation reduce manual effort?
    - Could visualization tools make reporting or decision-making easier?
    - Could collaboration platforms improve communication or coordination?
- Share your ideas with peers, or document them in a learning journal to reflect on your solutions.

Reflection and Building Digital Creativity

Reflection is essential for sustaining digital creativity. It helps you assess how currently use technology, recognize their strengths, identify gaps, and plan for continuous improvement. By taking time to review experiences, you can understand what worked well, what could be improved, and how to apply creative solutions more effectively in the future.

To start, conduct a self-assessment of your digital creativity skills. Consider which tools or workflows you used creatively, and evaluate the impact of your solutions on efficiency, collaboration, or problem-solving. Next, perform competence mapping: identify your strongest areas and note where additional learning or practice could enhance your skills. Finally, create an action plan to experiment with new tools, try alternative approaches, or integrate creative techniques into daily tasks.

Tips for Sustaining Digital Creativity:

- Experiment regularly with new features or tools.
- Share ideas and solutions with colleagues to inspire and learn from others.
- Monitor results, refine approaches, and adopt lessons learned.
- Stay informed about emerging technologies and innovative practices.

Sustained digital creativity comes from a cycle of practice, reflection, and continuous learning, empowering learners to innovate confidently and adapt effectively in any digital environment.

3.2 Identifying digital competence strengths and gaps

Understanding Digital Competence

Digital competence is more than just knowing how to operate a device or use a software application-it is the ability to leverage digital technologies effectively, efficiently, and safely to perform tasks, solve problems, and communicate. It combines technical proficiency, cognitive skills, and adaptive attitudes, enabling individuals to respond to evolving digital environments, integrate new tools, and optimize workflows.

Identifying your digital competence strengths and gaps allows you to target learning efforts where they will have the greatest impact. It ensures that creative applications of technology are grounded in solid skills and that solutions are implemented effectively. Organizations also benefit from competence mapping by identifying skill gaps across teams, deploying training strategically, and aligning tasks with individuals' capabilities. Awareness of gaps reduces inefficiencies, mitigates risk, and promotes safe, responsible technology use. By systematically understanding competencies, learners and organizations can make informed decisions about technology adoption, process optimization, and capacity-building initiatives.

Key Domains of Digital Competence

Digital competence can be organized into several interrelated domains:

- Technical Skills: Proficiency in software, hardware, cloud services, automation tools, and emerging digital platforms.
- Information Management: The ability to locate, evaluate, organize, and use information effectively for decision-making or workflow improvement.
- Collaboration and Communication: Skills in using digital platforms to coordinate tasks, share information, and communicate clearly within teams.
- Problem-Solving and Creativity: Applying digital tools innovatively to address challenges, streamline processes, and develop new solutions.
- Security and Compliance Awareness: Knowledge of data protection, cybersecurity best practices, and regulatory compliance to maintain digital safety.

Self-Assessment and Competence Mapping

A practical approach to identifying strengths and gaps is through structured self-assessment. Begin by listing the digital tasks you perform regularly and evaluate your confidence or proficiency in completing each task. Consider how these abilities align with emerging technologies, industry trends, or organizational expectations. For instance, a learner might be highly skilled in creating spreadsheets but have limited experience automating workflows or developing interactive dashboards. Identifying these gaps informs learning priorities and highlights opportunities to apply digital creativity more effectively.

Exercise:

- List five core digital tasks you perform in your daily work.
- For each task, rate your proficiency as High, Medium, or Low.
- Identify one or two areas where improving your skills could most enhance productivity, collaboration, or innovation.
- Reflect on your findings in a learning journal or discuss them with a peer: What specific steps, training, or practice could help you close these gaps?

Mapping digital competence provides a clear picture of both strengths and areas for improvement. It enables learners to focus skill development strategically, adopt technology solutions confidently, and support creative problem-solving in their work. Continuous self-assessment fosters adaptability in an evolving digital landscape, ensuring that individuals remain capable of leveraging technology effectively while maintaining efficiency, security, and innovation. By combining self-awareness with structured skill development, learners can grow into digitally competent, creative, and adaptive professionals.

3.3 Enhancing digital skills and planning growth

Developing digital competence is an ongoing process that extends beyond identifying strengths and gaps. Enhancing digital skills involves deliberate practice, continuous learning, and strategic planning to ensure that you can adapt to new tools, technologies, and workflows. In today’s dynamic digital environment, individuals who proactively grow their skills are better equipped to apply technology creatively, improve efficiency, and contribute meaningfully to organizational goals. Skill enhancement also empowers learners to embrace innovative approaches, reduce reliance on trial-and-error, and maintain confidence in problem-solving within digital contexts.

Continuous growth in digital competence is critical because technology and workplace needs are constantly evolving. A skill set that is sufficient today may become outdated tomorrow, and new tools or platforms may introduce opportunities that require different competencies. By actively enhancing skills, you can maintain relevance, expand their capacity for creative problem-solving, and strengthen their contribution to team and organizational performance. Planned skill development also mitigates risks associated with inefficient technology use, security gaps, or incomplete adoption of new systems.

Strategies for Enhancing Digital Skills

Enhancing digital competence involves multiple strategies that combine practice, structured learning, and experiential application:

- Targeted Training and Learning: Focus on courses, tutorials, workshops, or certifications that address specific gaps in your digital skills, such as data visualization, automation, or collaboration platforms.
- Practical Application: Apply new skills directly to workplace tasks or personal projects. Experimenting with real-world scenarios ensures that knowledge is internalized and transferable.
- Peer Learning and Mentorship: Collaborate with colleagues, share experiences, and learn from those with complementary skills. Mentorship and peer support accelerate learning and provide practical guidance for applying tools effectively.
- Structured Practice and Experimentation: Regularly experiment with new software features, integrations, or alternative workflows to explore creative applications. Iterative practice fosters confidence and adaptability.
- Reflection and Feedback: Continuously evaluate the impact of new skills on efficiency, problem-solving, and collaboration. Feedback from peers, supervisors, or self-reflection helps refine techniques and optimize outcomes.

Planning for Growth

Skill enhancement is most effective when paired with a structured growth plan. You should define clear goals, prioritize areas for improvement, and set realistic timelines for achieving milestones. Growth planning involves mapping current competence against desired outcomes, identifying learning resources, scheduling practice opportunities, and regularly reviewing progress. For example, if you want to automate reporting workflows, the growth plan might include learning a workflow tool, experimenting with scripts on non-critical tasks, and gradually implementing solutions in regular processes.

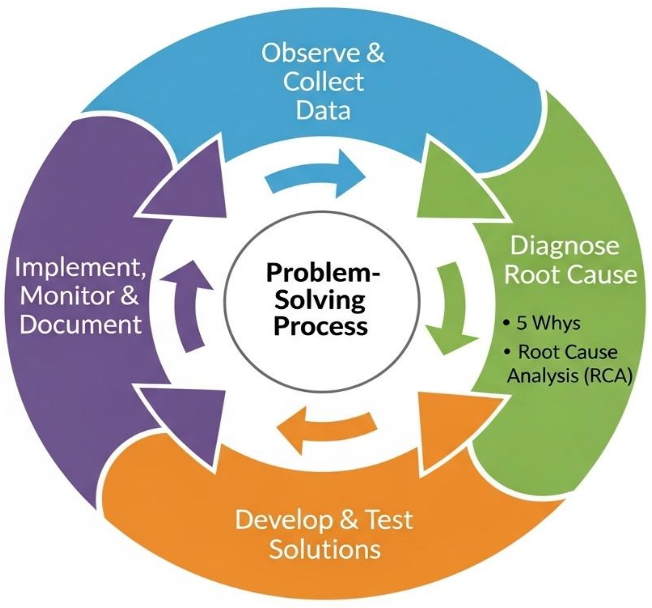


Figure 47: Problem solving process

Exercise:

- Review the digital competence gaps you identified before.
- Select one or two priority areas to focus on for improvement.
- Create a simple growth plan, including:
  - Specific skills to develop
  - Learning resources or training opportunities
  - Opportunities to practice or apply the skills
  - Timeline for achieving progress

- Record your reflections and progress, revising the plan as needed based on experience and feedback.

Enhancing digital skills and planning for growth ensures that learners can use technology creatively, confidently, and effectively. By combining targeted learning, practical application, collaboration, and reflection, individuals strengthen their competence and adaptability. Structured growth planning transforms skill development from a reactive task into a proactive, strategic process. Over time, this approach cultivates resilient professionals who are capable of navigating evolving digital landscapes, applying innovative solutions, and supporting organizational transformation.

Summary

This session explored two key aspects of digital growth: using technology creatively and understanding personal and organizational digital competence. Digital creativity involves applying tools in flexible, innovative ways to solve problems, improve processes, and enhance collaboration. Learners examined examples such as automating tasks, creating interactive dashboards, and leveraging collaboration platforms for brainstorming and coordination.

The session also focused on mapping digital competence across technical skills, information management, collaboration, problem-solving, and security awareness. Self-assessment exercises helped you identify strengths, recognize gaps, and align skills with evolving technology and organizational needs. Strategies for enhancing skills emphasized deliberate practice, targeted learning, peer collaboration, and structured growth planning. Overall, combining creativity with continuous competence development empowers you to innovate confidently and adapt effectively in a rapidly changing digital environment.

Review Questions

1. What does digital creativity involve?
   A. Using technology exactly as instructed
   B. Applying tools flexibly and innovatively to solve problems
   C. Memorizing all software features
   D. Avoiding experimentation

2. Why is mapping digital competence important?

   A. To identify strengths and gaps for skill development

   B. To replace existing workflows

   C. To reduce collaboration

   D. To learn one tool only

3. Which of the following is NOT a key domain of digital competence?

   A. Technical skills

   B. Information management

   C. Problem-solving and creativity

   D. Fashion design

4. Give an example of applying digital tools creatively in the workplace.

   A. Sending the same email repeatedly

   B. Automating task reminders in project management apps

   C. Ignoring collaboration platforms

   D. Writing reports manually without visualization

5. What is a key strategy for enhancing digital skills?

   A. Practicing only once

   B. Targeted learning, practical application, and reflection

   C. Avoiding new tools

   D. Using one method for all tasks

6. How can self-assessment help learners in digital growth?

   A. By identifying gaps and focusing learning on high-impact areas

   B. By reducing workflow complexity automatically

   C. By eliminating the need for creativity

   D. By memorizing software shortcuts

Capstone Project: Solving a Real Digital Problem

1. Purpose: This final project helps learners apply all the skills gained in Module 4. Learners will identify a real digital or technical problem in their school, organization, or community and design a practical solution using appropriate digital tools.
2. Objectives
   - Apply advanced problem-solving methods in a real situation.
   - Develop a creative, technology-based solution.
   - Demonstrate collaboration, communication, and reflection skills.
3. Project Steps

Table 16 Capstone Project Guidance

Stage	**Description**	**Expected Output**
1.Identify Problem	Choose a real digital issue (e.g., slow data system, poor online communication).	1-page problem statement
2. Analyze & Plan	Use methods like SWOT or root-cause analysis to explore causes and plan solutions.	Short analysis report or diagram
3. Design Solution	Create or simulate a digital tool (website, spreadsheet, workflow, or dashboard).	Screenshots or prototype
4. Implement / Test	Test the solution or present a simulation; gather feedback.	Implementation notes
5. Reflect & Present	Summarize lessons and present results to the class.	Final report + slides

4. Reflection Questions

I. What problem did you address and why?

II. Which tools or strategies were most effective?

III. What challenges did you face and how did you solve them?

IV. How can this skill help you in your work or studies?

Module 5: Python for Data Analysis

Introduction

This module provides learners with the essential skills to use Python for programming and data analysis. It begins by introducing the fundamentals of Python, including syntax, variables, data types, operators, user input, and control flow. Learners will gradually advance to core programming concepts such as functions, built-in data structures, file input/output operations, and error handling, enabling them to write efficient and robust programs.

Building on this foundation, the module introduces Python libraries widely used in data analysis, with a particular focus on NumPy. Learners will explore how to create and manipulate arrays, perform mathematical operations, and handle datasets effectively. Finally, the module concludes with the complete data analysis workflow, guiding learners through the phases of transforming raw data into meaningful insights using Python libraries and practical examples with CSV datasets.

By the end of this module, learners will be able to write Python programs, manage and manipulate data, apply numerical computing techniques with NumPy, and follow a structured workflow for data analysis tasks.

Session 1: Python Fundamental I

Introduction

Dear learners, welcome to this session where you will explore the fundamental concept of python. This session introduces learners to the fundamentals of Python programming. Learners will explore Python syntax, variables, data types, operators, user input, and control flow. Through hands-on exercises, they will practice writing Python programs, performing calculations, making decisions, and repeating tasks using loops. By the end of this session, learners will be able to write basic Python programs and control the flow of execution effectively.

Learning Objectives

By the end of this session, you will be able to:

- Explain Python’s purpose, applications, and an IDE.
- Apply variables, operators, decision-making and looping, and control flow.

Content outline

1.1 Python basics

1.2 Python operators

1.3 Control flow

1.1 Python Basics

Python is a high-level, interpreted programming language that is simple, powerful, and easy to learn. Its clean and readable syntax makes it a favorite among beginners and professionals alike.

Python is widely used in many fields, such as:

- 🌐 Web Development: building websites and web apps.
- 📊 Data Science: analyzing and visualizing data.
- 🤖 Artificial Intelligence & Machine Learning: powering smart systems.
- ⚙️ Automation: automating repetitive tasks.

An IDE (Integrated Development Environment) is a tool that helps you write, test, and debug code more efficiently. Some popular IDEs for Python are:

- PyCharm: A full-featured IDE known for its robust debugging, code analysis, and web development capabilities.
- VS Code (Visual Studio Code): A highly customizable and extensible code editor with excellent Python support through extensions.
- Spyder: An open-source IDE optimized for data science workflows, integrating well with Python data science libraries.
- IDLE: Python's default integrated development environment, bundled with the standard Python distribution.

- Google Collab: A cloud-based environment that runs Python directly in your browse

In this session, learners will utilize the PyCharm Integrated Development Environment (IDE). PyCharm Environment:

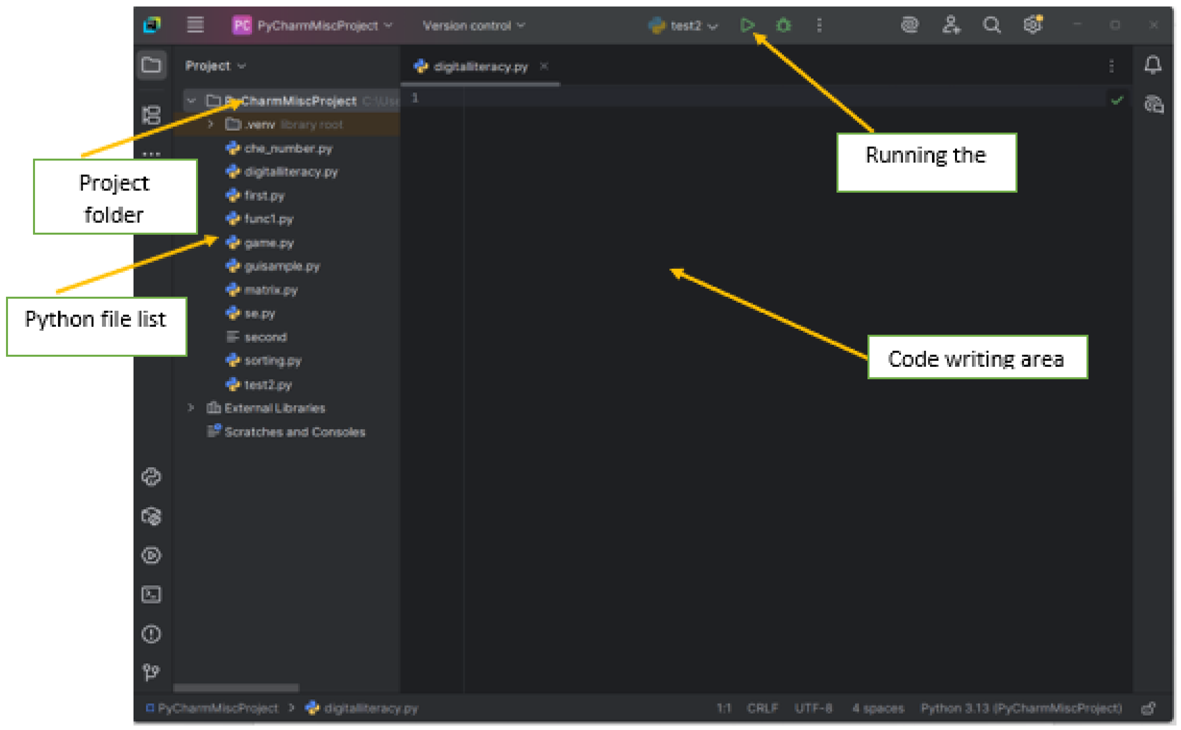


Figure 48: PyCharm Environment

Python basic syntax

Syntax refers to the rules that define the correct structure of code. Python is known for having a clear, readable syntax, which makes it a top choice for beginners. Unlike many other programming languages, Python doesn’t use curly braces {} or semicolons; to define blocks of code. Instead, it uses indentation, making the code look clean and easy to follow.

Indentation in Python: Indentation isn't just for readability - it's required. It uses indentation to group statements together.

Indentation tells Python where blocks of code begin and end. This is especially important in control flow structures like if, for, while, and function definitions. If our indentation is incorrect, Python will raise an Indentation Error.

Example:

```
if score > 80:
  print("You passed!")
```

- print() function: is a function that tells python to display output

  Text inside quotes " " is called a string.

  Output shown in the console will be: you passed

You can also print more than one item by separating them with commas:

```
name = "Alex"
age = 17
print("Name:", name, "Age:", age)
```

The output of this code will be: Name: Alex Age: 17

Comments

In coding, comments are notes written in the code. They are ignored by the computer but help programmers explain, organize, and understand the code. Here are the types of comments in Python**:**

- Single-line comments: Use the # symbol to write a comment on a single line:
  Example: # This is a single-line comment

```
name = "Alex" # You can also comment beside the code
```

- Multi-line comments: While Python doesn't have a specific multi-line comment syntax, you can use a triple-quoted string (''' or """) that isn't assigned to any variable.

Variable and data types

In programming, variables are fundamental building blocks that allow us to store, manipulate, and reuse data in our code. A variable is like a labeled box where you can store data. Imagine you have a box labeled "age" and you put the number 25 in it.

```
Example:  age = 25
          print(age)
```

Output: 25

Rules for Python Variable Names

When naming variables in Python, follow these rules:

1. Use only alphabets, numbers, and underscores (A-Z, a-z, 0-9, and _).
   Example: user_age, total_count2
2. Start with a letter or underscore, not a number. Valid: _count, name1 Invalid: 1name
3. No spaces allowed. Valid: first_name Invalid: first name
4. Case-sensitive: age, Age, and AGE are three different variables.
5. Avoid using Python keywords (like if, for, while) as variable names.

Data Types

It represents the kind of value that tells what operations can be performed on a particular data. Python has several built-in data types:

a) Integers (int): Whole numbers, positive or negative. E.g. *age = 25*, *count = -10*
b) Floating-point numbers (float): Numbers with decimal points. E.g. *pi = 3.14159*
c) String Type: store text and are created using single or double quotes. Eg. *name = "Alice"*
d) Boolean Type: represent True or False values.

Example:

```
is_student = True
   if is_student:
               print("You are a student")
   else:
               print("You are not a student")
```

e) Sequence Types

- Lists: are Ordered, mutable collections that can contain items of different types.
  E.g.

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

- Tuples: are Ordered, immutable collections.
  E.g.

```
coordinates = (10, 20)
print(coordinates)
```

Type Conversion

Sometimes we need to change a value from one data type to another. Python allows conversion between different data types. There are two types of type conversion:

- Implicit Type Conversion: Python automatically converts types in some cases.
  Example.

```
x = 7 # int
y = x / 2
print(y) # Output: 3.5 (float)
```

- Explicit Type Conversion: You can manually convert types using built-in functions.
  Example.

```
# String to int
num_str = "12" # int
num_int = int(num_str)
result = num_int + 10  # Result: 22
# Int to float
x = 5
y = float(x)
print(y)  # Output: 5.0
# Float to int (truncates decimal part)
z = int(3.14)
print(z)  # Output: 3
```

Checking Variable Types

Use type() or isinstance() to check variable types:

```
a = 5
print(type(a))  # Output: <class 'int'>
```

Input Types

A. The input() function

   The input() function allows us to receive information from the user through the keyboard and it always returns the input as a string.

   Example:

```
name = input("Enter your name: ")

print("Hello, " + name + "!")
```

   If the you type selam, the output will be: Hello, selam!

B. Handling Multiple User Inputs

   When working with user input, it’s common to require multiple values from the user. Python provides a convenient way to handle this using the split() method, which divides a string into a list of substrings based on a specified separator. In the case of user input, we can use split() to separate multiple values entered by the user.

   Example:

```
x, y = input("Enter two numbers separated by space: ").split()
print("First number:", x)
print("Second number:", y)
```

Remember input always return string so if you need to perform numerical operations on these inputs, you’ll need to convert them to numerical types. This can be achieved using the map() function, which applies a given function to each item of an iterable (in this case, the list of strings returned by split()).

- map(function, iterable) → applies a function to each item in a list.

Example:

```
x, y = map(int, input("Enter two numbers: ").split())
print("Sum:", x + y)
```

if you type: 10 7   the output will be 17

1.2 Python Operators

Python operators are special symbols or keywords used to perform operations on one or more operands. Operands can be variables, values, or expressions.

Arithmetic Operators: Used for mathematical calculations.

+ (Addition)	% (Modulo - returns the remainder of a division)
- (Subtraction)	** (Exponentiation - raises the left operand to the power of the right)
* (Multiplication)	// (Floor Division - returns the integer part of the quotient
/ (Division)	

Example:

```
#Arithmetic Operators
a=10
b=3
print(f"Arithmetic Operators:")
print(f"Addition: {a+b}")#13
print(f"Multiplication: {a*b}") #30
print(f"Division: {a/b}")  #3.333...
print(f"Floor Division: {a//b}")#3
print(f"Modulus: {a%b}")  #1
print(f"Exponentiation: {a**b}")#1000
```

Comparison (Relational) Operators: Used to compare two values, returning True or False.

== (Equal to)	!= (Not equal to)
> (Greater than)	< (Less than)
>= (Greater than or equal to)	<= (Less than or equal to)

Example:

```
#Comparison Operators
x=5
y=8
print(f"\nComparison Operators:")
print(f"Equal to: {x==y}") #False
print(f"Not equal to: {x!=y}") #True
print(f"Greater than: {x>y}") #False
print(f"Less than or equal to: {x<=y}")#True
```

Assignment Operators: Used to assign values to variables.

= (Assignment)

+=, -=, *=, /=, %=, **=, //= (Compound assignment operators, combining an arithmetic operation with assignment)

Example:

```
#Assignment Operators
c=15
print(f"\nAssignment Operators:")
c+=5 #c=c+5
print(f"Add and assign: {c}")#20
c-=3  #c=c-3
print(f"Subtract and assign: {c}") #17
c*=2  #c=c*2
print(f"Multiply and assign: {c}") #34
c/=4  #c=c/4
print(f"Divide and assign: {c}")   #8.5
```

Logical Operators: Used to combine conditional statements.

and (Returns True if both statements are true)

or (Returns True if at least one statement is true)

not (Reverses the logical state of its operand)

Example:

```
#Logical Operators
p=True
q=False
print(f"\nLogical Operators:")
print(f"AND: {p and q}") #False
print(f"OR: {p or q}")   #True
print(f"NOT: {not p}")   #False
```

Exercise:

Write a Python program that asks the user to input two numbers and operation they want (+, -, *, /), then performs the operation and prints the result

1.3 Control Flow

Control flow in Python determines the order in which instructions are executed in a program. It helps us make decisions, repeat tasks, and control how our program behaves.

Conditional Statements (Decision Making):

Conditional statements allow the program to choose different paths depending on conditions.

- if statement: Executes a block of code if a given condition is true.

Example:

```
x = 15
if x > 10
  print("x is greater than 10")
```

- if-else statement: Executes one block of code if the condition is true, and another block if it is false.

Example:

```
x = 10
if x > 15:
  print("x is greater than 15")
else:
  print("x is less than 15")
```

- if-elif-else statement: Allows for multiple conditions to be checked sequentially.

Example:

```
number = -10
if number > 0:
    print('Positive number')
elif number < 0:
    print('Negative number')
else:
    print('Zero')
print('This statement is always executed')
```

Looping Statements (Iteration):

Loops allow us to repeat tasks without writing the same code multiple times.

- for loop: Iterates over a sequence (e.g., list, tuple, string, range) and executes a block of code for each item.

  Example:

```
fruits = ["apple", "banana", "mango"]
for fruit in fruits:
    print(fruit)
```

```
x={1,2,3,4,5,6,7}
for i in x:
    print(i)
```

- while loop: Repeatedly executes a block of code as long as a given condition remains true.

  Example:

```
count = 1
while count < 10:
    print(count)
    count += 1
```

Summary

In this session, learners were introduced to the fundamentals of Python programming. They explored Python syntax, variables, and data types, and practiced writing and running programs using PyCharm. Learners explored user input, type conversion, and using operators for calculations and comparisons. They also applied control flow concepts through decision-making statements (if, if-else, if-elif-else) and loops (for and while) to control program execution. By the end of the session, learners are able to write simple Python programs, perform basic operations, and manage program flow effectively on their own.

Review Questions

1. What is the correct way to output "Hello, World!" in Python?
   A. echo("Hello, World!")
   B. print("Hello, World!")
   C. printf("Hello, World!")
   D. cout << "Hello, World!"
2. What does Python use to define blocks of code?
   A. Curly braces {}
   B. Semicolons;
   C. Indentation
   D. Parentheses ()
3. Which of the following is a valid variable name in Python?
   A. 2value
   B. value_2
   C. value-2
   D. value 2
4. How do you write multi-line comments in Python?
   A. /* comment */
   B. Triple quotes '''comment'''
   C. // comment
   D. # comment
5. What will be the output of the following code?

```
x = 5
y = 2
print(x // y)
```

   A. 2.5
   B. 2
   C. 3
   D. Error

6. Write the Output (Assume Input)

```
input: Hayat
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

7. What does the following code do?

```
count = 0
while count < 3:
print("Hi")
count += 1
```

   A. Infinite loop
   B. Prints "Hi" once
   C. Prints "Hi" three times
   D. Syntax error

8. What will be the output of this code?

```
x = 7
if x > 10:
    print("A")
elif x > 5:
    print("B")
else:
    print("C")
```

   A. A
   B. B
   C. C
   D. No output

Session 2: Python Fundamental II

Introduction

Dear learners, welcome to this session where you will explore the function, data structure, and file handling concept of python.

This session introduces learners to core Python programming concepts essential for writing practical and efficient programs. Learners will gain hands on experience with functions, built-in data structures, file input/output operations, and error handling. By the end of the session, learners will be able to create reusable code, manage and manipulate data effectively, read from and write to files, and handle errors gracefully to ensure robust programs.

Learning Objectives

By the end of this session, learners will be able to:

- Apply and use python functions and built-in data structures.
- Apply file I/O operations.
- Apply errors handling in Python programs.

Content Outline

2.1 Python Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Benefits of Using Functions

- Code Reuse
- Reduced code length
- Increased readability of code

Creating a Function

In Python a function is defined using the def keyword:

def is a keyword used to define or create a function. It marks the beginning of a function block and allows you to group a set of statements so they can be reused when the function is called.

Syntax

```
def function_name(parameters):
# indented code block (function body)
# code to execute
```

Explanation:

- *def* → starts the function definition.
- *function_name()* → name of the function.
- *parameters* → inputs for the function, optional.
- *:* → marks the start of the function body.
- *Indented code* → statements that run when the function is called.

Example:

```
def my_function():
  print("Hello from a function")
```

Calling a Function:

It uses to execute a function. To call a function, use the function name followed by parenthesis:

Example:

```
def my_function():
  print("Hello from a function")
my_function()
```

Parameters and Arguments:

Parameters: These are placeholders defined in the function signature that receive values when the function is called.

Arguments: These are the actual values passed to the function when it is invoked.

Example:

```
def my_function(fname):
        print(fname + " Alem")
my_function("Erku")
my_function("Ethiopia")
```

Output

```
Erku Alem
Ethiopia Alem
```

```
# function with two arguments
def add_numbers(num1, num2):
    sum = num1 + num2
    print("Sum: ", sum)
# function call with two values
add_numbers(5,4)
```

Output

```
Sum:9
```

Return Values:

Functions can optionally return a value using the return keyword. If no return statement is present, the function implicitly returns None.

Example:

```
def add(a, b):
    return a + b
result = add(5,3) #result will be 8
print(result)
```

Output

```
8
```

```
def sub():
    a=int(input("enter any number:"))
    b=int(input("enter second number:"))
    return (a-b)
res=sub()
print(res)
```

Output

```
enter any
number:20
enter second
number:10
10
```

2.2 Built-in Data Structures

Python provides several built-in data structures to organize and store collections of data efficiently. The most commonly used are Lists, Tuples, Dictionaries, and Sets. Each has unique characteristics that make it suitable for different tasks.

List [ ]: Ordered, mutable sequences of elements.

- Can contain elements of different data types (heterogeneous).
- Support operations like indexing, slicing, appending, inserting, and removing elements.

Example:

```
#Creating a list of fruits
fruits = ["mango ", "banana", "apple"]
#Accessing elements by index (indexing starts at 0)
first_fruit = fruits[0]  #"mango"
second_fruit = fruits[1] #"banana"
third_fruit = fruits[2]  #"apple"
```

```
name=["abe","hai","dae","lim"]
for i in name:
   print (i)
```

Output

```
abe
hai
dae
lim
```

Tuples ( ): Ordered, immutable sequences of elements.

- Similar to lists but cannot be modified after creation.
- Often used for representing fixed collections of related items.

Example:

```
my_tuple = (10, 20, 30)
print(my_tuple)
```

Output

```
(10, 20, 30)
```

```
person_info = ("Alex", 30, True, 18.5)
print(person_info)
```

Output

```
('Alex', 30, True, 18.5)
```

Sets { }: Unordered collections of unique elements.

- Mutable, allowing addition and removal of elements.
- Useful for operations involving uniqueness, such as finding common elements or differences between collections.

Example:

```
#create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)
```

Output

```
Student ID: {112, 114, 115, 116, 118}
```

Dictionaries { }: Unordered collections of key-value pairs.

- Mutable, allowing addition, modification, and deletion of key-value pairs.
- Keys must be unique and immutable (e.g., strings, numbers, tuples).
- Values can be of any data type.
- Defined using curly braces {} with key: value pairs.
- Efficient for retrieving values based on their associated keys.

Example:

```
#Creating a dictionary
#Using curly braces {}
student_info = {
    "name": "Alex",
    "age": 20,
    "major": "Cyber Security"
}
print("Student Info", student_info)
print("Student name:", student_info["name"])
```

Output

```
Student Info: {'name': 'Alex', 'age': 20, 'major': 'Cyber Security'}
Student name:Alex
```

Adding Elements to the list

Adding the elements in the list can be achieved using the append(), extend() and insert() functions.

- The append() function adds all the elements passed to it as a single element.
- The extend() function adds the elements one-by-one into the list.

- The insert() function adds the element passed to the index value and increase the size of the list too.

Example:

```
my_list = [1, 2, 3]
print(my_list)
my_list.append([10, 11]) #add as a single element
print(my_list)
my_list.extend([100, 'more_list']) #add as different elements
print(my_list)
my_list.insert(1, 'insert_list') #add element i
print(my_list)
```

Output

```
[1, 2, 3]
[1, 2, 3, [10, 11]]
[1, 2, 3, [10, 11], 100, 'more_list']
[1, 'insert_list', 2, 3, [10, 11], 100, 'more_list']
```

Deleting Elements

- To delete elements, use the *del* keyword which is built-in into Python but this does not return anything back to us.
- If you want the element back, you use the pop() function which takes the index value.
- To remove an element by its value, you use the remove() function.

Example:

```
my_list = [1, 2, 3, 'list', 3.132, 10, 30]
del my_list[5] #delete element at index 5
print(my_list)
my_list.remove('list') #remove element with value
print(my_list)
a = my_list.pop(1) #pop element from list
print('Popped list: ', a, ' List remaining: ', my_list)
my_list.clear() #empty the list
print(my_list)
```

Output

```
[1, 2, 3, 'list', 3.132, 30]
[1, 2, 3, 3.132, 30]
Popped list: 2 List remaining:  [1, 3, 3.132, 30]
[]
```

2.3 File IO in Python

File IO in Python revolves around two main operations: reading from files and writing to files. Python offers built-in methods and functions to make file handling easier, allowing developers to complete these tasks more quickly.

Opening a File in Python

Before reading or writing, a file must be opened using the open() function. The open() function requires two arguments:

1. The **filename**.
2. The **mode** in which the file is opened:
   - 'r' → read (default)
   - 'w' → write (creates or overwrites)
   - 'a' → append (adds to the end)
   - 'b' → binary mode

   Example:

```
file = open('example.txt', 'r')
```

In this example, example.txt is the name of the file, and 'r' is the mode for reading.

Writing to a File

To write data to a file, you use the write() method. But first you need to open the file in write mode ('w') or append mode ('a').

```
file = open('example.txt', 'w')
file.write('Digital literacy')
file.close()
```

Writing Multiple Lines

If you need to write multiple lines, you can use the writelines() method.

Example:

- A list of strings, each ending with a newline character (\n)

```
lines = ["First line\n", "Second line\n", "Third line\n"]
```

- Open the file in write mode ('w') , If the file does not exist, it will be created and If it already exists, its contents will be overwritten.

```
with open('example1.txt', 'w') as file:
```

- Write all the lines from the list into the file

```
file.writelines(lines)
```

Reading the Entire File

The simplest way to read a file is to read its entire content at once.

```
file = open('example1.txt', 'r')
content = file.read()
print(content)
file.close()
```

Reading multiple lines: Let us see the following two examples.

```
file=open('example1.txt','r')
for line in file:
    print(line.strip())
```

```
file = open('example1.txt', 'r')
line = file.readline()
while line:
    print(line, end='')
    line = file.readline()
file.close()
```

CSV file I/O operations

File I/O operations with CSV files in Python are handled using the built-in **csv** module. This module allows you to easily read from and write to CSV files.

Steps to Write to a CSV File

1. Import the csv module

```
import csv
```

2. Open the CSV file
   - Use *with open()* to safely open the file.
   - Open in write mode (*'w'*) to create/overwrite, or append mode (*'a'*) to add rows.
   - Always include *newline=' '* to prevent blank lines on Windows.

```
with open('output.csv', 'w', newline='') as file:
```

3. Create a csv.writer object

```
Writer = csv.writer(file)
```

4. Write data to the file
   - Single row: use

```
writer.writerow()
```

   - Multiple rows: use

```
writer.writerows()
```

Example: Suppose we have a csv file named employee.csv with the following entries.

```
Name,   Age, Profession
Haile,   23,  Doctor
Bishaw, 22,  Engineer
```

Writing the data to csv file

```
import csv
with open('employee.csv','w',newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Name','Age','Profession'])
    writer.writerow(['haile','23','Doctor'])
    writer.writerow(['Bishaw','22','Engineer'])
```

Reading the data

```
import csv
with open('employee.csv', 'r') as file:
   reader = csv.reader(file)
   for row in reader:
       print(row)
```

Exercise:

Create a file called students.csv with the following structure:

Name, Age, Grade

Alex, 20, A

selam, 22, B

Chalie, 21, C

And print the information

2.4 Python Error Handling

When writing programs, errors are very common. Some errors stop the program completely, while others can be anticipated and handled. Error handling in Python allows you to manage these situations gracefully instead of crashing your program.

Python uses exceptions to indicate errors. To manage them, we use the keywords try, except, and finally.

try block

- Contains the code that might cause an error.
- If an error occurs, Python jumps to the matching except block.

except block

- Defines how to respond to a specific error (e.g., ZeroDivisionError, ValueError).
- Prevents the program from crashing.

finally block

- Code inside finally always runs, no matter what happens (error or no error).
- Useful for cleanup tasks like closing a file or releasing resources

Example:

```
try:
    # code that may raise an exception
    x = 10 / 0
except ZeroDivisionError:
    # code to handle the exception
    print("You can't divide by zero!")
```

```
try:
    x = 5 / 1
except ZeroDivisionError:
```

```
try:
    x = int(input("Enter first number: "))
    y = int(input("Enter second number: "))
    result = x / y
    print("Result:", result)
except ZeroDivisionError:
    print("Error: You cannot divide by zero!")
except ValueError:
    print("Error: Please enter only numbers.")
```

- If the user enters 0 as the second number → ZeroDivisionError is caught.
- If the user enters text → ValueError is caught.

Summary

In this session, learners explored the fundamentals of Python programming that go beyond basic syntax. They learned to organize code into functions, work efficiently with built-in data structures for storing and manipulating data, and perform file operations to read and save information. Additionally, learners understood how to handle errors and exceptions, ensuring that programs run reliably even when unexpected issues occur. Through hands-on examples and exercises, learners are now prepared to write structured, modular, and error-resistant Python programs.

Review Questions

1. What is the output of this code?
```
fruits = ["mango", "banana", "juice"]
fruits.append("date")
print(fruits[-1])
```
   A. "juice"
   B. "mango"
   C. "date"
   D. Error
2. Which mode should you use to open a file for writing and remove its existing content?
   A. 'r'
   B. 'a'
   C. 'w'
   D. 'x'
3. What is the correct way to define a function in Python?
   A. function myFunc():
   B. def myFunc():
   C. func myFunc():
   D. define myFunc():
4. What is the output of the following code?
```
my_list = [1, 2, 3,6]
print(len(my_list))
```
   A. 2

B. 3

C. 4

D. Error

5. Which function is used to write a line to a file?

A. writefile()

B. print()

C. write()

D. writeline()

6. What will be the output?

```
def greet(name):
    return f"Hi {name}"
print(greet("Sam"))
```

A. Hi

B. Hi name

C. Hi Sam

D. greet("Sam")

7. What is the output of this code?

Assume a file example.txt contains: Hello digital literacy

```
with open("example.txt", "r") as file:
print(file.read())
```

A. Hello

B. Hello World

C. sample.txt

D. Error

Session 3: Essential Data Analysis Libraries

Introduction

Dear learners, welcome to this session where you will explore numpy library in python.
In this session, learners will be introduced to Python libraries that are widely used for data analysis. The focus will be on NumPy, the foundation library for numerical computing in Python. Learners will explore how NumPy arrays differ from Python lists, how to create arrays, perform indexing, slicing, and carry out common mathematical operations. By the end of this session, learners will be able to use NumPy arrays effectively for data manipulation and basic analysis tasks.

Learning Objectives

By the end of this session, learners will be able to:

- Explain the purpose of Python data analysis libraries.
- Apply one-dimensional and multi-dimensional arrays.
- Apply array indexing, slicing, and basic mathematical operations.

Content outline

3.1 Introduction to Python Libraries

Data analysis libraries are pre-written collections of functions, tools, and classes in Python that make it easier to work with, clean, manipulate, and analyze data. Instead of writing code from scratch, you can use these libraries to save time and ensure accuracy. They are especially important in data science, machine learning, and statistics, since they provide optimized, efficient, and reliable methods for handling large datasets.
Most Widely Used Data Analysis Libraries:

- **NumPy**
  - Foundation for numerical computing in Python.
  - Provides fast n-dimensional arrays (ndarray) and mathematical functions.
  - Used for: linear algebra, numerical computations, element-wise operations.
- **Pandas**
  - Built on top of NumPy.

- Provides DataFrame and Series objects for handling tabular data.
- Used for: data cleaning, manipulation, joining datasets, handling missing values, time series analysis.

- **Matplotlib**
  - A popular plotting library.
  - Used for: creating static, interactive, and animated plots (bar charts, line graphs, scatter plots).

In this session, we will focus on NumPy, the foundation library for numerical data analysis.

3.2 NumPy: Numerical Python

NumPy, short for Numerical Python, is one of the most important foundational packages for numerical computing in Python. Most scientific libraries in Python such as Pandas, rely on NumPy arrays for data storage and computation.

Why NumPy?

- Faster than Python lists (due to optimized C implementation).
- Supports vectorized operations (no need for explicit loops).
- Provides tools for linear algebra, random number generation, statistics, and reshaping data.

Create Arrays

NumPy arrays can be created in several ways:

```
import numpy as np
# From Python list
arr = np.array([1, 2, 3, 4])
# Predefined arrays
zeros = np.zeros((2,3))   # 2x3 array of zeros
ones = np.ones((3,3))     # 3x3 array of ones
range_arr = np.arange(0, 10, 2)  # Even numbers from 0 to 8
```

Types of Arrays

Two types of arrays are as follows:

- **One Dimensional Array:**

  A one-dimensional array is a type of linear array.

5	6	7	8	2

```
import numpy as np
a = [5, 6, 7, 8, 2]
arr = np.array(a)
print("List in python : ", a)
print("Numpy Array in python :", arr)
```

Output

```
List in python :  [5, 6, 7, 8, 2]
Numpy Array in python : [5 6 7 8 2]
```

- **Multidimensional Array in NumPy**

A Multi-Dimensional Array is an array that can store data in more than one dimension such as rows and columns (like a matrix). One of the key features of NumPy is its **N-dimensional array object** (ndarray), which is a fast, flexible container for large datasets in Python. Arrays enable you to perform mathematical operations on whole blocks of data using similar syntax to the equivalent operations between scalar elements.

5	6	7	8
1	9	3	10
2	11	12	13

```
import numpy as np
list_1 = [5, 6, 7, 8]
list_2 = [1, 9, 3, 10]
list_3 = [2, 11, 12, 13]
sample_array = np.array([list_1,
                    list_2,
                    list_3])
print("Numpy multi dimensional array\n",
sample_array)
```

Output

```
Numpy multidimenstional array
[[ 5  6  7  8]
 [ 1  9  3 10]
 [ 2 11 12 13]]
```

Array Indexing & Slicing

Array Indexing:

The method for accessing and manipulating data elements or subsets inside a NumPy array is called indexing.

One dimensional array indexing: Elements in one-dimension array are accessed using their index number in square brackets. Indices are 0-based, meaning the first element is at index 0.

```
import numpy as np
arr = np.array([40, 30, 20, 60, 50])
print(arr[2]) # Output: 20
```

Negative Indexing: Negative indices count from the end of the array, where -1 is the last element.

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[-1]) # Output: 50
```

Multi-dimensional Indexing: For multi-dimensional arrays, indices are provided for each dimension, separated by commas.

```
array_samp = np.array([[5, 2, 6], [9, 3, 7]])
print(array_samp [0, 1]) # Access element at row 0, column 1 (output: 2)
```

Array Slicing:

Slicing allows selecting a range of elements using the syntax start:stop:step.

- start: The starting index (inclusive).
- stop: The stopping index (exclusive).
- step: The step size (defaults to 1).

Example:

```
import numpy as np
arr = np.array([40, 50, 60, 80, 100])
print(arr[1:4])
print(arr[::2]) #(every other element)
```

Output

```
[50, 60, 80]
[40, 60, 100]
```

Array Operations

NumPy allows **element-wise operations** on arrays, meaning operations are applied to each corresponding element in arrays. Arithmetic operations like addition, subtraction, multiplication, and division.

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

Output

```
[5 7 9]
[-3 -3 -3]
[4 10 18]
[0.25 0.4 0.5]
```

Common Mathematical Functions

NumPy includes many functions for **mathematical computations**:

Aggregate Functions:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(np.sum(arr))
print(np.mean(arr))
print(np.min(arr))
print(np.max(arr))
print(np.std(arr))
```

Output

```
10
2.5
1
4
1.1180
```

Element-wise Functions:

```
import numpy as np
arr= np.array([1,2,3,4])
print(np.sqrt(arr)) # square root of each element
print(np.exp(arr))  # Exponential of each element
print(np.log(arr))  # Natural logarithm
print(np.sin(arr))  # Sine of each element
```

Output

```
[1.          1.41421356   1.73205081    2]
[ 2.71828183  7.3890561    20.08553692   54.59815003]
[0.          0.69314718   1.09861229    1.38629436]
[0.84147098   0.90929743   0.14112001   -0.7568025]
```

Summary

This session introduced learners to Python libraries for data analysis, with a focus on NumPy. Learners explored how to create both one-dimensional and multi-dimensional arrays, access elements using indexing, and select subsets using slicing. They practiced basic array operations such as addition, subtraction, multiplication, and division, as well as using aggregate and element-wise mathematical functions like sum, mean, standard deviation, square root, exponential, logarithm, and sine. With these skills, learners can efficiently store, manipulate, and analyze numerical data in Python, forming a strong foundation for future work with advanced data analysis libraries.

Review Questions

1. What is NumPy primarily used for in Python?
   A. Text processing
   B. Image editing
   C. Numerical and scientific computing
   D. Web development
2. What will be the output of the following code?
   import numpy as np
   arr = np.array([40, 30, 20, 60, 50])

```
print(arr[2])
```

A. 40
B. 30
C. 20
D. 60

3. What is the output of the code below?

```
import numpy as np
array_samp = np.array([[5, 2, 6], [9, 3, 7]])
print(array_samp[0, 1])
```

A. 5
B. 6
C. 2
D. 3

4. What will be printed?

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(np.sum(arr))
```

A. 10
B. 24
C. 6
D. 12

5. What is the output of this code?

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1:4:2])
```

A. [2, 3, 4]
B. [2, 4]
C. [1, 2]
D. [1, 3]

6. What is the output of the following code?

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4])
print(np.max(arr))
```

A. 1
B. 2
C. 4
D. 0

7. What is the output of the following?

```
import numpy as np
arr = np.array([1, 2, 3])
print(np.square(arr))
```

A. [1, 4, 9]
B. [1, 2, 3]
C. [1.0, 1.41, 1.73]
D. [1, 8, 27]

8. What will be printed by this code?

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(np.mean(arr))
```

A. 2.5
B. 10
C. 4
D. 2

Session 4: Data Analysis Workflow

Introduction

Dear learners, welcome to this session where you will explore the fundamental workflow of data analysis in python.

Data analysis is a systematic process of transforming raw data into meaningful insights for decision-making. In this session, learners will explore the key stages of the workflow using Python libraries and Practical examples using CSV datasets will illustrate how each step is implemented in Python.

Learning Objectives

By the end of this lesson, learners will be able to:

- Explain the key phases of the data analysis workflow.
- Perform data analysis using python.

Content Outline

4.1 Data analysis phases

The data analysis workflow is a methodical process for drawing conclusions from data. It guarantees that unprocessed data is methodically converted into useful information for making decisions. The main stages are listed below, with a focus on important technical subtopics

Library Setup

It's crucial to import pertinent libraries and modules that offer strong tools for data handling, analysis, and visualization before beginning any data analysis task.

Common Python Libraries:

- NumPy: Numerical operations and array processing.
- Pandas: Data manipulation and analysis.
- Matplotlib & Seaborn: Data visualization.
- Scikit-learn: Machine learning and preprocessing.
- Statsmodels: Statistical analysis.

- Openpyxl, CSV, Requests: Data acquisition and file handling.

Data Acquisition

It is the phase you will do finding relevant data sources, gathering information manually or automated methods, digitizing physical phenomena, and then storing and arranging the information in a structured way for later analysis, visualization, and decision-making. The goal is to import the data efficiently and correctly.

Common data sources:

- CSV/Excel files: via pandas.read_csv() or read_excel()
- Databases: using connectors like SQLAlchemy, sqlite3, or pyodbc
- Web APIs: using requests, json
- Online datasets: via URLs or scraping tools
- Cloud storage: e.g., Google Drive, AWS S3

Eample Importing CSV File in Python: Sample students**.csv** File Content is above the python code

```
ID,Name,Age,Grade
1,Abebe,20,A
2,Almaz,21,B
3,Henok,19,A
4,Selam,22,C
5,Meles,20,B
import pandas as pd
dl= pd.read_csv('students.csv')  # Assuming the
CSV file is named students.csv and located in the
 same directory as script:
print(df.head())     # View the first few rows
```

Output

```
ID   Name    Age   Grade
 1   Abebe    20     A
 2   Almaz    21     B
 3   Henok    19     A
 4   Selam    22     C
 5   Meles    20     B
```

Data Cleaning and Preprocessing

Data cleaning and preprocessing are essential steps in preparing raw data for analysis. They help ensure data quality, consistency, and suitability for machine learning or statistical modeling.

A. Data Cleaning Tasks

- **Handling Missing Values:** Datasets often contain missing values (NaN). These must be handled before analysis. Common strategies:
    - **Imputation** → Fill missing values with mean, median, or mode.
    - **Removal** → Drop rows or columns with too many missing values.

```
import pandas as pd
dl = pd.DataFrame({
   'A': [1, 2, None, 4],
   'B': [5, None, 7, 8]
})
# Fill missing values with column mean
dl_filled = dl.fillna(dl.mean())
print(dl_filled)
```

- **Removing Duplicates:** Duplicate rows can cause biased results**.** Identifying and eliminating duplicate rows to ensure data uniqueness.

```
dl_no_duplicates = dl.drop_duplicates()
```

- **Correcting Erroneous Values:** Addressing inconsistencies or errors in data entries, often through data validation rules or manual inspection.
- **Standardizing Data Formats:** Ensuring consistent data types and formats across columns. Example: Convert all dates to YYYY-MM-DD.

```
dl['Date'] = pd.to_datetime(dl['Date'], errors='coerce')
```

- **Dealing with Outliers:** Identifying and handling extreme values that might skew analysis, through methods like capping, transformation, or removal. Outliers can skew results**.**

Approaches:

- **Capping** extreme values
- **Transformation** (e.g., log, square root)
- **Removal** if they are errors

B. Data Preprocessing Tasks

- **Feature Scaling:** Scaling numerical features to a similar range to prevent features with larger values from dominating algorithms. Scaling ensures features contribute equally to models. Common techniques include:
  - Normalization (Min-Max Scaling): Scales data to a specific range, usually between 0 and 1.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(dl[['Numerical_Feature']])
```

  - Standardization (Z-score Normalization): Rescales data to have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
standardized_data=scaler.fit_transform(dl[['Numerical_Feature']])
```

- Label Encoding: Convert categorical values into numeric form for ML models.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
dl['Encoded_Column'] = le.fit_transform(dl['Categorical_Column'])
```

Data Visualization

Data visualization is the graphical representation of information and data. It helps you explore the datasets, understand patterns, trends, and outliers, and communicate results effectively.

Common Python Libraries for Visualization:

- **Matplotlib**: Basic plots (line, bar, scatter, histogram)
- **Pandas**: Quick plotting from DataFrames
- **Seaborn**: Advanced statistical plots (heatmaps, boxplots, pairplots)

Example: Visualizing Data from a CSV. Sample Dataset (students.csv)

```
ID,Name,Age,Grade
1,Abebe,20,A
2,Almaz,21,B
3,Henok,19,A
4,Selam,22,C
5,Meles,20,B
6,Biruk,21,C
7,Dagmawi,22,A
```

Example Visualizations:

- **Bar Chart - Grade Distribution**

  Here is the Python code to visualize the given dataset as a Bar Chart - Grade Distribution.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the data
dl = pd.read_csv('students.csv')
# Count students per grade
grade_counts = dl['Grade'].value_counts()
# Bar plot
plt.figure(figsize=(6,4))
sns.barplot(x=grade_counts.index, y=grade_counts.values)
plt.title('Grade Distribution')
plt.xlabel('Grade')
plt.ylabel('Number of Students')
plt.show()
```

The following figures show the output of the visualization code provided above.

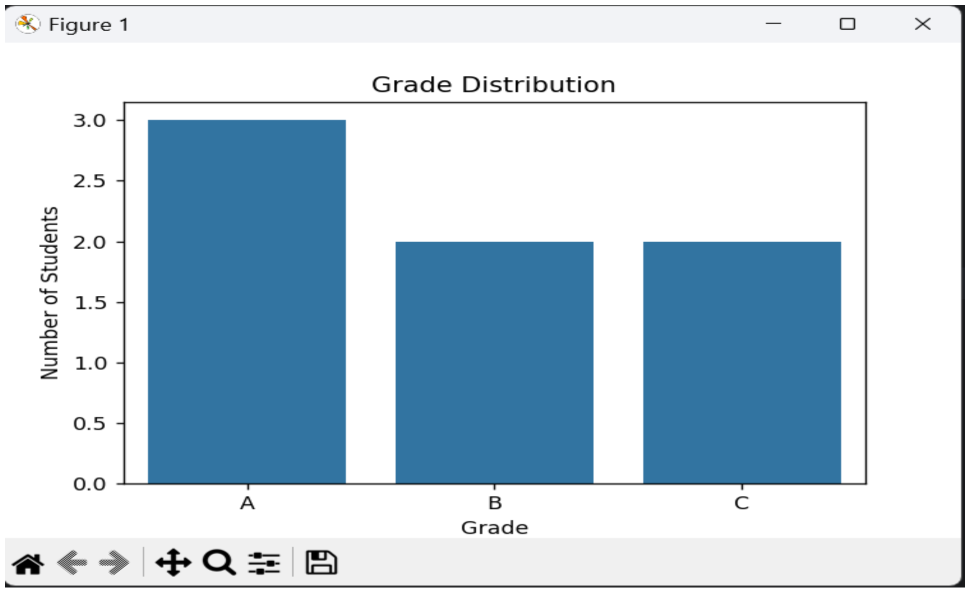


Figure 49: A bar chart showing how many students got each grade (A, B, C).

- **Histogram - Age Distribution**

  Here is the Python code to visualize the given dataset as a Histogram - Age Distribution.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the dataset
dl = pd.read_csv('students.csv')
# Plot Histogram – Age Distribution
plt.figure(figsize=(6, 4))
sns.histplot(dl['Age'], bins=4, kde=True)
plt.title('Student Age Distribution')
plt.xlabel('Age')
plt.ylabel('Number of Students')
plt.show()
```

The following figure shows the output of the Histogram - Age Distribution visualization code.

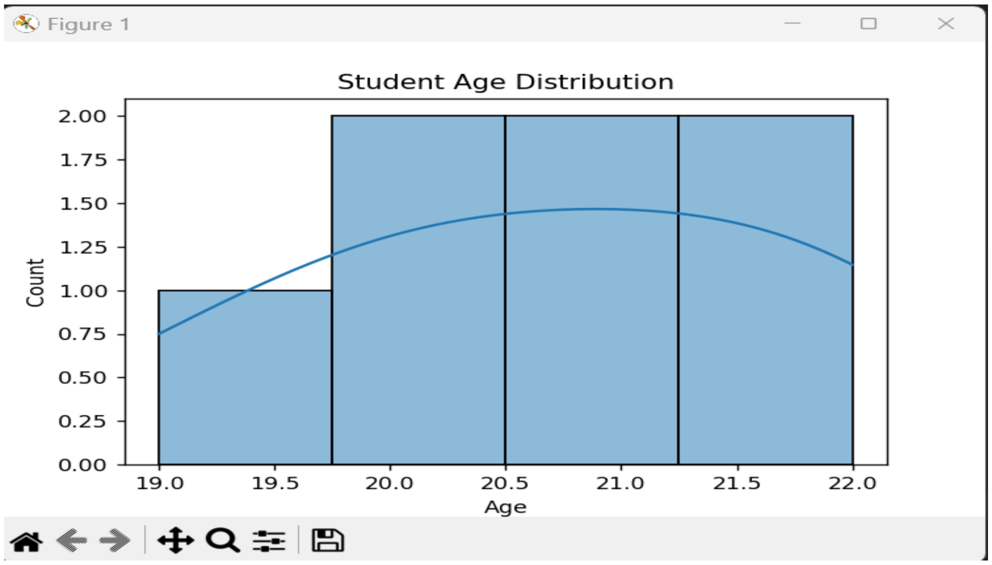


Figure 50: Histogram - Age Distribution

- **Scatter Plot - Age vs. Grade**

  Here is the Python code to visualize the given dataset as a Scatter Plot - Age vs. Grade.

```
import pandas as pd
import matplotlib.pyplot as plt
# Load the dataset
dl = pd.read_csv('students.csv')
# Map grades to numeric values for plotting
grade_map = {'A': 3, 'B': 2, 'C': 1}
dl['Grade_Num'] = dl['Grade'].map(grade_map)
# Plot Scatter Plot – Age vs. Grade
plt.figure(figsize=(6, 4))
plt.scatter(dl['Age'], dl['Grade_Num'], color='blue', alpha=0.7)
plt.title('Age vs. Grade')
plt.xlabel('Age')
plt.ylabel('Grade (numeric)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

The following figure shows the output of the Scatter Plot - Age vs. Grade visualization code.

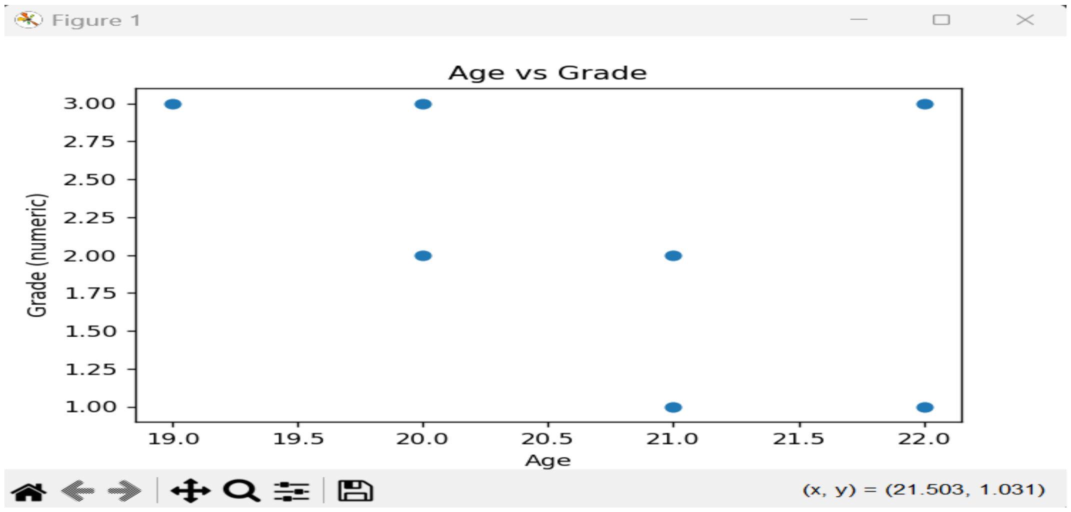


Figure 51: Scatter Plot - Age vs. Grade

Summary

In this lesson, learners explored the phases of data analysis, starting from library setup and data acquisition to cleaning, preprocessing, and visualization. They learned how to import Python libraries, gather and organize datasets, handle missing or inconsistent data, and create meaningful visualizations to analyze patterns and trends. By mastering these phases, learners are prepared to perform structured, effective, and reproducible data analysis in Python.

Practical Application and Projects

Project title: Analyzing Digital Literacy and Technology Usage

Having the dataset given bellow analyze the data for:

1. Skill Level Distribution
2. Internet Access by Age
3. Online Payment Usage vs Social Media Usage

Table 17: Digital Literacy and Technology Usage dataset

ID	Age	Gender	Internet Access	Device Usage	Email Usage	Social Media Usage	Online Payment Usage	Digital Skill Level
1	22	M	Yes	Smartphone	Yes	Yes	No	Intermediate
2	28	F	Yes	Laptop	Yes	Yes	Yes	Advanced
3	19	M	No	Tablet	No	Yes	No	Beginner
4	35	F	Yes	Smartphone	Yes	No	Yes	Intermediate
5	40	M	Yes	Laptop	Yes	Yes	Yes	Advanced
6	24	F	Yes	Smartphone	Yes	Yes	No	Intermediate
7	30	M	No	Tablet	No	No	No	Beginner
8	27	F	Yes	Laptop	Yes	Yes	Yes	Advanced
9	21	M	Yes	Smartphone	Yes	Yes	No	Intermediate
10	33	F	Yes	Laptop	Yes	Yes	Yes	Advanced
11	23	M	Yes	Smartphone	Yes	Yes	No	Intermediate
12	29	F	Yes	Laptop	Yes	No	Yes	Advanced
13	20	M	No	Tablet	No	Yes	No	Beginner
14	36	F	Yes	Smartphone	Yes	Yes	Yes	Advanced
15	38	M	Yes	Laptop	Yes	Yes	Yes	Advanced
16	25	F	Yes	Smartphone	Yes	Yes	No	Intermediate
17	33	M	No	Tablet	No	No	No	Beginner
18	26	F	Yes	Laptop	Yes	Yes	Yes	Advanced
19	22	M	Yes	Smartphone	Yes	Yes	No	Intermediate
20	34	F	Yes	Laptop	Yes	Yes	Yes	Advanced
21	23	M	Yes	Smartphone	Yes	Yes	No	Intermediate
22	28	F	Yes	Laptop	Yes	No	Yes	Advanced
23	19	M	No	Tablet	No	Yes	No	Beginner
24	35	F	Yes	Smartphone	Yes	Yes	Yes	Beginner
25	40	M	Yes	Laptop	Yes	Yes	Yes	Advanced

Capstone Project: Python for Data Analysis

1. Introduction:

   This Capstone Project concludes Python for Data Analysis. It allows learners to apply Python programming skills to real or simulated datasets, demonstrating their ability to clean, analyze, and visualize data to solve real-life problems. The project emphasizes local relevance using data from sectors such as education, agriculture, health, or small businesses in Ethiopia.

2. Purpose: To help learners use Python libraries (NumPy, Pandas, Matplotlib) to analyze real or locally simulated data and produce meaningful insights that support decision-making.

3. Learning Objectives: By completing this project, learners will be able to:
   - Collect, clean, and prepare real or sample Ethiopian data for analysis.
   - Use Python for summarizing, visualizing, and interpreting data.
   - Communicate findings that inform social or economic issues in Ethiopia.
   - Demonstrate responsible and ethical data handling.

4. Project Tasks: Learners will complete the capstone in five structured stages:

Stage	**Description**	**Expected Output**
1. Choose a Theme and Dataset	Select a dataset related to Ethiopian development or local issues (e.g., student performance, crop yield, hospital visits, mobile banking use, or unemployment). Use open data sources such as CSA Ethiopia, the Ministry of Education, or simulated community data.	Dataset file (.csv or .xlsx) and short description
2. Clean and Prepare Data	Use Python (Pandas/NumPy) to handle missing values, rename columns, and convert data types.	Cleaned dataset and short code snippet
3. Analyze the Data	Perform basic statistical and exploratory analysis (mean, median, correlation, group-by summaries).	Printed output or analysis report

4. Visualize Findings	Create at least two visualizations (bar chart, histogram, or scatter plot) using Matplotlib or Seaborn.	Image files or screenshots of graphs
5. Interpret and Report Results	Summarize insights: What do the numbers show? What decisions can be made from the analysis?	Short written summary

5. Reflection Questions
   - I. What local issue or dataset did you analyze, and why is it important?
   - II. Which Python functions or libraries were most useful in your project?
   - III. What challenges did you face in cleaning or analyzing the data?
   - IV. How can data analysis support better decisions in your community or workplace?

Glossary

Term	Definition
API	Application Programming Interface. A set of protocols, routines, and tools that allows two software applications to communicate with each other.
ATS	Applicant Tracking System. Software used by companies to manage recruitment and hiring. It often scans CVs and resumes for keywords before a human recruiter sees them.
Crowdfunding	A method of raising capital for a project or venture by obtaining small amounts of money from a large number of people, typically via the Internet.
Crypto-currencies	Digital or virtual currencies that are secured by cryptography, making them impossible to counterfeit. They operate independently of a central bank.
DOM	Document Object Model. A programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.
Fintech	Financial Technology. Companies offering innovative financial services and products by leveraging modern technology, often replacing or supplementing traditional banking methods.
Gig Economy	A labor market characterized by the prevalence of short-term contracts or freelance work, as opposed to permanent jobs.
Mitigation Strategies	Specific actions or planned steps taken to reduce the likelihood of a risk occurring or minimize its negative impact if it does occur.
NumPy	Numerical Python. A fundamental Python library that provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
SEO	Search Engine Optimization. The practice of increasing the quantity and quality of traffic to a website through organic search engine results.
USSD	Unstructured Supplementary Service Data. A protocol used by GSM cellular phones to communicate with the service provider's computers, often used for mobile money and banking services without needing internet access.
Workflow	The series of steps, tasks, or activities that must be executed to complete an organizational process or goal.

Agile Methodology	An approach to project management, particularly software development, where requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer/end-user.
Big Data	Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations.
Blockchain	A decentralized, distributed ledger that records transactions across many computers so that the record cannot be altered retroactively without the alteration of all subsequent blocks and the consensus of the network.
Data Visualization	The presentation of data in a pictorial or graphical format (like charts, graphs, and maps) to make complex data understandable and accessible.
Digital Footprint	The unique set of traceable data activities, contributions, and communications (likes, comments, post) that a person leaves while navigating the internet.
Firewall	A network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. It acts as a barrier between a trusted internal network and untrusted external networks.
Metadata	Data about data. It provides information about a piece of content, such as the author, date created, file size, location data, and copyright information.
Pandas	A fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool built on top of the Python programming language.
Two-Factor Authentication (2FA)	A security process in which a user provides two different authentication factors to verify themselves, typically a password (something they know) and a code sent to a mobile device (something they have).
VPN	Virtual Private Network. A service that encrypts your internet traffic and disguises your online identity by rerouting your connection through a remote server.
GSM	Global System for Mobile Communications. It is a crucial standard developed to describe the protocols for second-generation (2G) digital cellular networks used by mobile phones.

References

1. Alliance for Financial Inclusion. (2020). *Digital financial services basic terminology*. https://www.afi-global.org
2. GSMA. (2021). *State of the Industry Report on Mobile Money 2021*. GSM Association. https://www.gsma.com/mobilemoney
3. National Bank of Ethiopia. (2022). *Digital financial services directive and strategy*. https://www.nbe.gov.et
4. Ethio Telecom. (n.d.). *Telebirr: Mobile money service*. https://www.ethiotelecom.et/telebirr/
5. Commercial Bank of Ethiopia. (n.d.). *CBE Birr – Mobile Money Service*. https://www.combanketh.et
6. Safaricom Ethiopia. (n.d.). *M-Pesa Ethiopia: Mobile financial services*. https://www.safaricom.et
7. Dashen Bank. (n.d.). *Dashen Bank SuperApp*. https://www.dashenbanksc.com
8. World Bank. (2020). *Digital financial inclusion*. https://www.worldbank.org/en/topic/financialinclusion/brief/digital-financial-inclusion
9. United Nations Capital Development Fund (UNCDF). (2019). *Digital finance for financial inclusion*. https://www.uncdf.org
10. Amole. (n.d.). *Amole Digital Payment Services*. https://www.amolepay.com
11. Canva. (n.d.). Design a resume. https://www.canva.com/resumes/
12. NovoResume. (n.d.). Online resume builder. https://www.novoresume.com/
13. Zety. (n.d.). Resume builder. https://zety.com/resume-builder
14. Upwork. (n.d.). Hire freelancers & find freelance jobs online. https://www.upwork.com/
15. Fiverr. (n.d.). Freelance services marketplace. https://www.fiverr.com/
16. PeoplePerHour. (n.d.). Hire freelancers online & find freelance jobs. https://www.peopleperhour.com/
17. Toptal. (n.d.). Hire the top 3% of freelance talent. https://www.toptal.com/
18. LinkedIn. (n.d.). Build your professional profile. https://www.linkedin.com/
19. Wix. (n.d.). Create a free professional website. https://www.wix.com/
20. Google Sites. (n.d.). Create and collaborate on websites. https://sites.google.com/
21. Behance. (n.d.). Showcase and discover creative work. https://www.behance.net/
22. GitHub. (n.d.). Build software better, together. https://github.com/
23. World Economic Forum. (2020). The rise of the gig economy. https://www.weforum.org
24. International Labour Organization (ILO). (2021). Digital labour platforms and the future of work. https://www.ilo.org
25. LinkedIn Learning. (n.d.). Career development and digital identity courses. https://www.linkedin.com/learning/
26. PayPal. (n.d.). Send, receive, and manage your payments. https://www.paypal.com/
27. Payoneer. (n.d.). Global payment solutions for freelancers and businesses. https://www.payoneer.com/
28. Telebirr. (n.d.). Digital financial services by Ethio Telecom. https://www.ethiotelecom.et/telebirr/
29. M-Pesa Ethiopia. (n.d.). Mobile money for the Ethiopian market. https://www.safaricom.et/m-pesa/
30. Etsy. (n.d.). Buy and sell handmade, vintage items, and craft supplies. https://www.etsy.com/
31. Shopify. (n.d.). Start your e-commerce store. https://www.shopify.com/
32. Amazon. (n.d.). Online retail and marketplace. https://www.amazon.com/

33. Gumroad. (n.d.). Sell digital products and memberships. https://www.gumroad.com/
34. YouTube. (n.d.). About YouTube monetization. https://support.google.com/youtube/answer/72857
35. Google AdSense. (n.d.). Monetize your content with ads. https://www.google.com/adsense/
36. Patreon. (n.d.). Membership platform for creators. https://www.patreon.com/
37. TikTok. (n.d.). TikTok Creator Marketplace & Monetization. https://www.tiktok.com/business/en
38. Teachable. (n.d.). Create and sell online courses. https://teachable.com/
39. Udemy. (n.d.). Teaching and learning marketplace. https://www.udemy.com/
40. Mozilla Developer Network (MDN). (n.d.). CSS: Cascading Style Sheets. https://developer.mozilla.org/en-US/docs/Web/CSS
41. W3Schools. (n.d.). HTML Tutorial. https://www.w3schools.com/html/
42. W3Schools. (n.d.). CSS Tutorial. https://www.w3schools.com/css/
43. GeeksforGeeks. (n.d.). Introduction to HTML. https://www.geeksforgeeks.org/html-introduction/
44. GeeksforGeeks. (n.d.). CSS Basics. https://www.geeksforgeeks.org/css-introduction/
45. World Wide Web Consortium (W3C). (n.d.). HTML & CSS standards. https://www.w3.org/
46. CSS-Tricks. (n.d.). A complete guide to the CSS box model. https://css-tricks.com/the-css-box-model/
47. freeCodeCamp. (n.d.). Learn HTML and CSS for free. https://www.freecodecamp.org/
48. PHP.net. (n.d.). PHP Manual. https://www.php.net/manual/en/
49. Apache Friends. (n.d.). XAMPP: Apache + MariaDB + PHP + Perl. https://www.apachefriends.org/
50. W3Schools. (n.d.). PHP Tutorial. https://www.w3schools.com/php/
51. GeeksforGeeks. (n.d.). Introduction to PHP. https://www.geeksforgeeks.org/php-introduction/
52. TutorialsPoint. (n.d.). PHP Basics Tutorial. https://www.tutorialspoint.com/php/
53. phpMyAdmin. (n.d.). phpMyAdmin - A web interface for MySQL and MariaDB. https://www.phpmyadmin.net/
54. CodeChef IDE. (n.d.). Online IDE for programming. https://www.codechef.com/ide
55. Ideone. (n.d.). Online compiler and debugging tool. https://www.ideone.com/
56. JetBrains. (n.d.). PhpStorm - PHP IDE. https://www.jetbrains.com/phpstorm/
57. Microsoft. (n.d.). Visual Studio Code. https://code.visualstudio.com/
58. WordPress.org. (n.d.). WordPress – Create a website or blog. https://wordpress.org/
59. WordPress.com. (n.d.). WordPress.com: Build a site with ease. https://wordpress.com/
60. Apache Friends. (n.d.). XAMPP: Easy Apache, MySQL, PHP, Perl. https://www.apachefriends.org/
61. phpMyAdmin. (n.d.). phpMyAdmin – Database management tool. https://www.phpmyadmin.net/
62. W3Schools. (n.d.). WordPress Tutorial. https://www.w3schools.com/wordpress/
63. Kinsta. (2024). What is WordPress? Explained for beginners. https://kinsta.com/knowledgebase/what-is-wordpress/
64. Jetpack by WordPress.com. (n.d.). Overview of the WordPress Dashboard. https://jetpack.com/support/wordpress-dashboard-overview/
65. ThemeIsle. (2023). Beginner's guide to WordPress themes and customization. https://themeisle.com/blog/how-to-use-wordpress-themes/
66. Hostinger. (2024). How to install WordPress locally using XAMPP. https://www.hostinger.com/tutorials/how-to-install-wordpress-on-xampp
67. Elementor. (n.d.). How to create WordPress pages and posts. https://elementor.com/help/creating-pages-posts/

68. Microsoft. (n.d.). Task Manager overview. https://support.microsoft.com/en-us/windows/task-manager-overview-1b3451c4-6c5b-4e0a-bcbb-93770ec2e3de
69. Cisco. (n.d.). Introduction to network troubleshooting tools. https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/13788-3.html
70. Wireshark Foundation. (n.d.). Wireshark user guide. https://www.wireshark.org/docs/wsug_html_chunked/
71. Nagios Enterprises. (n.d.). Nagios monitoring software. https://www.nagios.com/products/nagios-xi/
72. Zabbix. (n.d.). Zabbix monitoring solutions. https://www.zabbix.com/documentation/current/manual
73. Splunk Inc. (n.d.). Splunk platform overview. https://www.splunk.com/en_us/products/splunk-platform.html
74. Atlassian. (n.d.). Workflow analysis and process mapping. https://www.atlassian.com/work-management/workflows
75. Harvard Business Review. (2016). How to conduct a needs analysis. https://hbr.org/2016/09/how-to-conduct-a-needs-analysis
76. Gartner. (2022). Evaluating and selecting digital tools: Best practices. https://www.gartner.com/en/documents
77. Zapier. (n.d.). Automation tools for workflow optimization. https://zapier.com/blog/automation-tools/
78. Microsoft Teams. (n.d.). Collaboration and communication platform. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
79. Slack Technologies. (n.d.). Slack features and integrations. https://slack.com/features
80. Asana. (n.d.). Project management software guide. https://asana.com/guide
81. U.S. Department of Health & Human Services. (n.d.). Security best practices for organizations. https://www.hhs.gov/hipaa/for-professionals/security/index.html
82. McKinney, W. (2018). Python for data analysis: Data wrangling with Pandas, NumPy, and IPython (2nd ed.). O’Reilly Media.
83. Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. CreateSpace.
84. NumPy Developers. (2023). NumPy documentation. https://numpy.org/doc/stable/
85. Python Software Foundation. (2023). Python documentation. https://docs.python.org/3/
86. Lutz, M. (2013). Learning Python (5th ed.). O’Reilly Media.
87. McKinney, W. (2023). Python for data analysis: Tips and techniques for working with structured data. O’Reilly Media.
88. VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. O’Reilly Media.
89. McKinney, W., & et al. (2020). Practical Python for data analysis. O’Reilly Media.
90. Python Software Foundation. (2023). Python CSV file reading and writing. https://docs.python.org/3/library/csv.html

Content Developers Profile

SN	Full name	Qualification	Specialization	Institution	Role	Email
1.	Abebaw Mulat	MSc	IT	BDPTC	Content Developer	abebawmulat@gmail.com
2.	Alemayehu Abera	MSc	IT M	BDPTC		aberaalemayehu19@gmail.com
3.	Belete Mersha	MSc	CS	BDU-BiT		bele.2001@gmail.com
4.	Endalew Alemu	MSc	IT	BDPTC		endomark8@gmail.com
5.	Haleluya Kiflu	MSc	IS	ANRSEB		haleluyaluya@gmail.com
6.	Mahlet Woreta	MSc	CS	BDU-BiT		mahletworeta@gmail.com
7.	Mihretu Fitsum	BSc	ISE	ANRC-ITB		fitsumsha@gmail.com
8.	Netsrework Berhanu	MSc	CS	BDU-BiT		nassweet143@gmail.com
9.	Gashaw Kindie	MEd	Education and Curriculum Studies	ANRSEB	Coordinator	gashawkindie@gmail.com

Reviewers Profile

SN	Full name	Qualification	Specialization	Institution	Role
1	Tesfa Tegegne Asfaw (PhD)	PhD	CS	BDU-BiT	Module reviewer
2	Yibeltal Tafere	MSc	CS	BDU-BiT	
3	Tamir Anteneh Alemu	MSc (Ass't prof.)	IS	BDU-BiT	
4	Dejenie Aynalem Ayenew	MSc	CS	BDPTC	
5	Temesgen Melaku Kassa (PhD)	PhD	EDM	BDU	Coordinator