BASED ON NATIONAL CURRICULUM 2022-23
TEXTBOOK OF

Computer Science

Grade 12

NATIONAL BOOK FOUNDATION
AS
FEDERAL TEXTBOOK BOARD, ISLAMABAD

Based on National Curriculum of Pakistan 2022-23

Textbook of

Computer Science

Grade 12

National Curriculum Council
Ministry of Federal Education and Professional Training

National Book Foundation
as
Federal Textbook Board
Islamabad

PREFACE

This textbook has been developed by NBF according to the National Curriculum of Pakistan 2022- 2023. The aim of this textbook is to enhance learning abilities through inculcation of logical thinking in learners, and to develop higher order thinking processes by systematically building upon the foundation of learning from the previous grades. A key emphasis of the present textbook is on creating real life linkages of the concepts and methods introduced. This approach was devised with the intent of enabling students to solve daily life problems as they go up the learning curve and for them to fully grasp the conceptual basis.

After amalgamation of the efforts of experts and experienced authors, this book was reviewed and finalized after extensive reviews by professional educationists. Efforts were made to make the contents student friendly and to develop the concepts in interesting ways.

The National Book Foundation is always striving for improvement in the quality of its books. The present book features an improved design, better illustration and interesting activities relating to real life to make it attractive for young learners. However, there is always room for improvement and the suggestions and feedback of students, teachers and the community are most welcome for further enriching the subsequent editions of this book.

May Allah guide and help us (Ameen).

Dr. Kamran Jahangir

Managing Director

The Practical Importance of the Textbook in Everyday Life

The following is a unit-by-unit summary of what is presented in each unit with its practical importance in our everyday life.

UNIT 1 Computer Systems: This unit focuses on Human Computer Interaction (HCI). HCI is a field of study about designing and development of interactive systems that are efficient, effective and user-friendly while using digital devices. HCI plays an important role in interaction between the computer system and the user in providing ways through which the user can input data, receive feedback and navigate through the system.

UNIT 2 Computational Thinking & Algorithms: This unit is about developing algorithms which is the first step for writing computer programs. Learning how to develop algorithms holds immense significance for students as it enhances their ability to think critically and provide structured approach to computational problem-solving

UNIT 3 Programming Fundamentals: In this unit students learn programming fundamentals in Python. Python is a high level object-oriented programming language. This unit describes advanced programming constructs such as data structures, file handling and databases and how to implement complex algorithms in Python. In today's modern world, having computer programming skill is essential as it plays an important role in automation of tasks that we perform in our daily life.

UNIT 4 Data and Analysis: This unit focuses on data analysis. It contains material on Machine Learning (ML). ML is a powerful technology that is transforming many industries by making processes efficient and effective. It provides solutions to many problems related to various fields such as marketing and advertising, financial services, healthcare and self-driving cars. Material on data visualization is also explored which is representation of information in the form of chart, graph, maps, diagram, etc.

UNIT 5 Applications of Computer Science: This unit describes the applications of Internet of Things (IoT), Blockchain and Cloud Computing that are applicable in government and private sectors in Pakistan. Applications of these enhances our daily life by improving productivity, efficiency and security. Material on neural networks and deep learning is also presented that is used for development of complex systems. Data sharing and privacy conflicts are explored at the end of the unit to understand issues related with it.

UNIT 6 Impacts of Computing: This unit focuses on issues faced by people when collaborating on digital or online platforms. Today, much of our daily life revolves around the Internet and this exposes us to a wide range of online threats. This unit describes security measures and safe practices such as 2FA, biometric verification and secure ways for transmitting data, to mitigate online threats.

UNIT 7 Digital Literacy: This unit presents material that describes the creation of artefact that answers a research question, communicates results and conclusions through digital resources or tools. It plays an important role in strengthening research, analytical skills and fosters digital literacy.

UNIT 8 Entrepreneurship in Digital Age: This unit explores the creation of a Minimum Viable Product (MVP) for entrepreneurs and businesses. MVP provides low-risk testing ground before making huge investment into a product. It allows entrepreneurs and businesses to validate their business ideas, minimize costs and mitigate risks while launching a new product.

Table of Contents

1 Computer Systems

Learning Outcomes

At the end of this unit students will be able to:

- explain the usability, security and accessibility of devices, the systems they are integrated with.
- explain human interaction with computer systems in terms of:
  - Usability
  - Common problems
  - Methods for improvements
  - Ethical, social, economic, and environmental implications
- identify and explain tradeoffs between the usability and security of computing systems, recommend cybersecurity measures by considering different factors such as efficiency, cost, privacy, and ethics.

Introduction

In today's rapidly evolving digital world, technology plays a central role in our everyday lives. Whether we are using smartphones, accessing online services, or interacting with smart devices at home, the design and functionality of these technologies deeply impact our experiences. As we dive deeper into the world of technology, three key concepts emerge as critical to our interaction with these systems. These are **usability, security, and accessibility.**

- **Usability** ensures that devices and systems are easy to use and understand. Without it, even advanced technology can become frustrating and ineffective.
- **Security** protects our data and interactions in a world of evolving cyber threats. Strong measures are essential to safeguard sensitive information and prevent severe consequences.
- **Accessibility** guarantees that technology is available to everyone, including those with disabilities. As our world becomes increasingly digital, it is important that no one is left behind. Accessible design makes sure that all users, regardless of their physical abilities, can fully interact with and benefit from technology.

What will You Learn?

This unit explores the key principles of usability, security, and accessibility, highlighting their role in designing effective and inclusive technology. You will also learn about **Human-Computer Interaction (HCI)** and its focus on user needs. Additionally, the unit examines the **trade-offs between usability and security**, providing insights into how designers and engineers balance these often conflicting requirements to create systems that protect users without compromising their experience.

1.1 System/Device Usability, Security, and Accessibility

When devices and systems are designed, they should be easy to use (usability), safe and secure from threats (security), and accessible to everyone, including people with disabilities (accessibility). Understanding these aspects is crucial because they affect how we interact with technology and how technology affects our lives.

1.1.1 Device/System Usability

Usability is a key factor in the success of any device or system. It refers to how easily and efficiently users can interact with technology, directly affecting their satisfaction and productivity. By applying usability principles, designers can create user-friendly systems that meet user needs and provide a seamless experience.

Fig.1.1: Human-Computer Interaction (HCI)

A well-designed system with high usability is easy to learn, simple to use, and aligned with user expectations. It enables users to achieve their

goals effectively, efficiently, and with minimal effort, ensuring a positive and satisfying experience. Fig.1.1 illustrates such a system.

Key Aspects of Usability:

The following are the key aspects of Usability.

- **Effectiveness:** The degree to which users can successfully complete tasks using the device or system.
- **Efficiency:** The speed and minimal effort with which tasks can be accomplished.
- **Satisfaction:** The user's sense of comfort and positive experience while interacting with the system.

Principles of Usability

Several key principles guide the design of user-friendly systems, ensuring they meet the needs and expectations of their users. These principles include:

- Simplicity
- Consistency
- Feedback
- Error Prevention
- Learnability
- Accessibility

1. Simplicity:

The design should be as straightforward as possible, removing unnecessary complexity and focusing on core functionalities.

Example: The Apple iPod (Fig.1.2), with its simple click-wheel interface, allowed users to navigate music/video quickly and easily, without the clutter of extraneous buttons.

Fig.1.2: Simplicity

Google's homepage (Fig.1.3) is iconic for its simplicity, concentrating on the primary function—searching—without overwhelming users with excessive information or options.

Fig.1.3: Google homepage

2. Consistency:

Interfaces should maintain a consistent design across different parts of the system, making it easier for users to predict how the system will behave and reducing the cognitive load required to use it.

Example: Microsoft Office uses similar menus and toolbars across Word, Excel, and PowerPoint, helping users transition smoothly between applications, as shown in Fig.1.4.

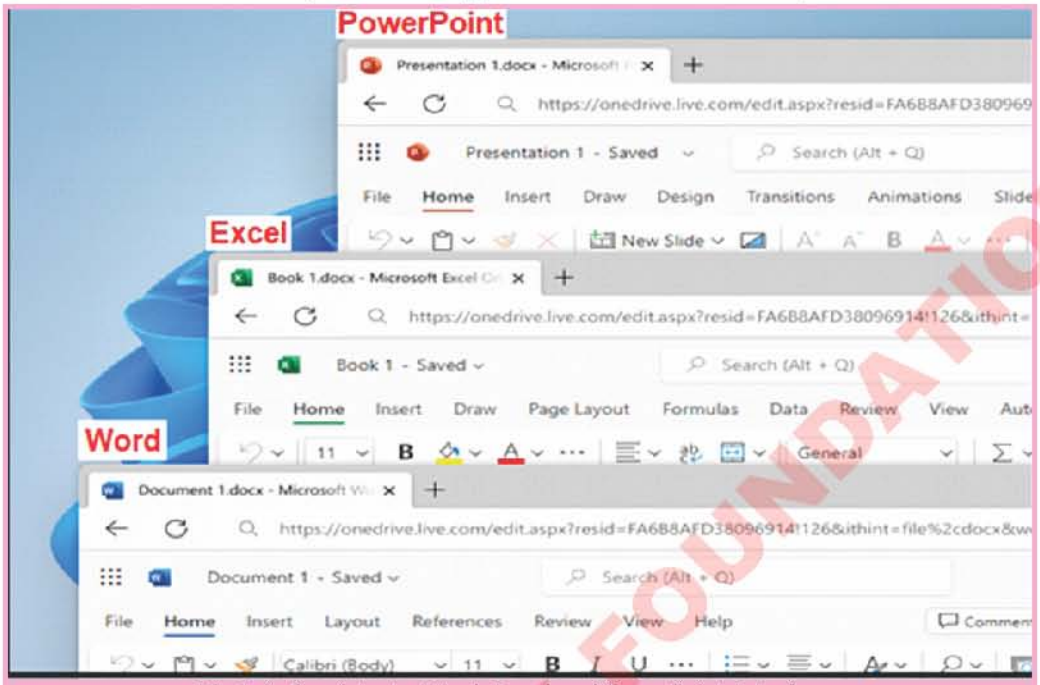


Fig.1.4: Consistent - Word, Excel and PowerPoint Interfaces

3. Feedback:

The system should provide clear, immediate feedback to users regarding their actions, helping them understand if their inputs were successful and what the system is doing in response.

Fig.1.5: Application installation progress bar - feedback

Example: E-commerce sites like Amazon provide immediate feedback when an item is added to the shopping cart, often with a pop-up or visual confirmation, reassuring the user that their action was successful. Another example is the progress bar in software installations, as shown in Fig.1.5, which indicates the ongoing process and how much time is remaining.

4. Error Prevention:

The design should prevent errors before they occur and offer easy ways to correct them if they do.

Example: When you try to close a document without saving, a word processor like Microsoft Word prompts you to save your work, preventing accidental data loss. As shown in Fig.1.6.

Fig.1.6: Error Prevention

5. Learnability:

The system should be easy for new users to learn, allowing them to become proficient quickly, with minimal training or guidance.

Example: Mobile apps often include a tutorial or walkthrough for first-time users, helping them get started quickly.

6. Accessibility:

The system should be designed so that it can be easily used by everyone, regardless of their abilities. This includes making sure that people with disabilities can also use it comfortably. The goal is to ensure that the system is inclusive, meaning it works well for as many people as possible, no matter their physical or cognitive abilities.

Example1: Voice Over feature on Phones (Fig.1.7)

Voice Over feature allows visually impaired users to interact with their Phones by reading out text on the screen. Users can navigate through apps, read messages, and even use the camera with voice guidance, making the device accessible to those who are blind or have low vision.

Fig.1.7: Voice Over feature on Phones

Example2: TV Remote Controls (Device Accessibility)

Some TV remotes are designed with large buttons and high-contrast colors to assist users with vision or dexterity issues. These remotes are easier to see and press, making them more accessible to elderly users, as shown in Fig.1.8.

Fig.1.8: TV Remote Controls

1.1.2 Usability Testing

Usability testing is a key method used to evaluate how user-friendly and effective a system or device is. There are several types of usability testing, each with a specific focus and approach.

The following are some common types of usability testing methods.

1. Moderated Usability Testing

In this type of testing, a facilitator or moderator is present to guide users through tasks, ask questions, and observe behaviors in real-time. The moderator can also interact with users to clarify instructions or probe deeper into specific issues. Fig. 1.9 shows the whole process.

Example: A company developing a new software application conducts moderated usability testing by inviting participants to a lab environment. The moderator asks users to perform specific tasks, such as navigating through the app's features or completing a transaction. The moderator observes how easily users can accomplish these tasks and asks questions about their experience, helping to identify any usability issues.

Fig.1.9: Moderated Usability Testing

2. Unmoderated Usability Testing

In unmoderated usability testing, users complete tasks on their own without a moderator's direct involvement. This type of testing is often conducted remotely, allowing participants to use the system or device in their natural environment. The sessions are typically recorded for later analysis, as shown in Fig.1.10.


Fig.1.10: Unmoderated Usability Testing

Example: An e-commerce website wants to test how easily customers can find and purchase products. The company conducts unmoderated usability testing by sending out tasks for users to complete at home, such as searching for a product, adding it to the cart, and completing the checkout process. The user interactions are recorded, and the company reviews the footage to identify any pain points or areas of confusion.

3. Qualitative Usability Testing

This approach focuses on understanding the user's experience, thoughts, and feelings while interacting with the system or device. The goal is to gather in-depth insights into user behavior, including what they find intuitive or challenging. As shown in Fig.1.11.


Fig.1.11: Qualitative Usability Testing

Example: A mobile app developer conducts qualitative usability testing by interviewing users after they use the app. Users are asked open-ended questions about their experience, such as what they liked or disliked and any challenges they encountered. This feedback helps the developer understand the user's perspective and make improvements to the app's design.

4. Quantitative Usability Testing

Fig.1.12: Quantitative Usability Testing

Quantitative usability testing collects numerical data to measure usability aspects like task completion time, error rates, and success rates. This method is useful for comparing different versions of a system or device or setting benchmarks for usability.

Example: A company tests two versions of a website's navigation menu. In a quantitative usability test, users are timed as they attempt to find specific information using each menu version. The company collects data on how long it takes users to complete the tasks and how often they make errors. This data is then analyzed to determine which menu is more efficient and user-friendly. As shown in Fig.1.12.

5. Remote Usability Testing

Fig.1.13: Remote Usability Testing

Remote usability testing allows users to test a system or device from their own location, using their own devices. This type of testing can be either moderated or unmoderated, and it is particularly useful for reaching a wider audience or observing how users interact with a product in their natural environment.

Example: A global software company wants to test a new feature with users in different countries. Remote usability testing is conducted, with participants using the software on their own computers while being observed through screen-sharing technology. The company gains insights into how the feature performs in different regions and environments without needing to bring users into a lab. As shown in Fig.1.13.

6. A/B Testing

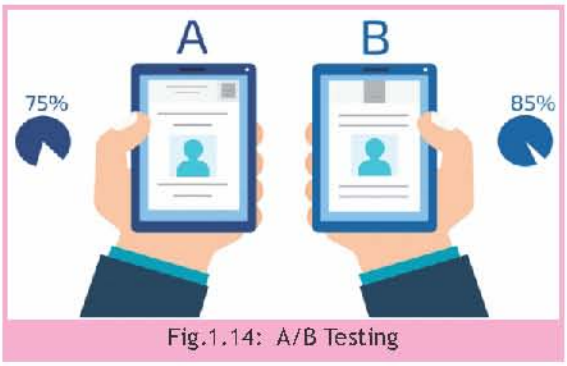


Fig.1.14: A/B Testing

A/B testing, also known as split testing, involves comparing two versions of a system or device to see which one performs better in terms of usability, as shown in Fig.1.14. This type of testing is commonly used in web design and online services.

Example: An online retailer wants to determine whether a new homepage layout will lead to more sales. They create two versions of the homepage and use A/B testing to randomly show each version to different users. By comparing the sales metrics and user engagement data between the two versions, the retailer can determine which layout is more effective.

1.1.3 Common Usability Issues and How to Avoid Them

Usability issues can significantly impact the effectiveness and user satisfaction of a system or device. When users encounter problems with usability, they may become frustrated, make errors, or even abandon the product altogether. Understanding common usability issues and knowing how to avoid them is crucial in designing user-friendly systems.

The following are some common usability issues, along with practical solutions and examples.

1. Complex Interfaces

Complex interfaces are those that overwhelm users with too much information, too many options, or overly complicated layouts. When users are faced with an interface that is difficult to understand or navigate, they are more likely to make mistakes, take longer to complete tasks, or feel frustrated.

Fig.1.15: Complex Interfaces

Solution: To avoid complexity, focus on simplicity. Simplify the interface by prioritizing core functions and removing unnecessary features. The design should highlight the most important tasks and provide a clear and intuitive path for users to accomplish them.

Example: Example: Consider a mobile banking app (Fig.1.15) that initially has dozens of features crammed into a single screen. Users might find it difficult to locate basic functions like checking their balance or transferring funds. By simplifying the interface—perhaps by organizing features into easily accessible categories or hiding advanced options behind menus—the app becomes much easier to use. A good example is the early versions of the PayPal app, which went through multiple iterations to simplify its interface, eventually focusing on core actions like sending money and checking balances on the main screen.

2. Inconsistent Navigation

Inconsistent navigation occurs when the navigation structure or elements change from one section of the system to another. This inconsistency can confuse users, making it difficult for them to predict how to move around the system or find the information they need. As a result, users may feel lost or frustrated.

Solution: To maintain consistency, ensure that navigation menus, buttons, and other interface elements remain uniform across all parts of the system. Consistency in design allows users to develop a mental model of how the system works, making it easier for them to navigate.

Example: Imagine a website where the "Contact Us" button is located at the top of the page on some sections and at the bottom on others. Users might struggle to find the contact information depending on where they are on the site. A solution would be to place the "Contact Us" button in the same location on every page, such as in a fixed header or footer. Amazon's website, for

instance, keeps its navigation consistent across different pages, with the search bar and navigation menu always in the same place, helping users find what they need quickly.

3. Poor Error Messages

Fig.1.16: Poor Error Messages

Poor error messages are those that are unclear, overly technical, or provide little to no guidance on how to resolve the issue. When users encounter such error messages, they can become confused, frustrated, and unable to continue using the system effectively.

Solution: Write error messages that are clear, concise, and helpful. Good error messages should explain what went wrong in plain language and provide actionable steps that users can take to fix the problem.

Example: Consider an online form where a user enters an incorrect date format, and the system simply displays "Error 150: Invalid input.", as shown in Fig.1.16." This message is vague and doesn't help the user correct the mistake. Instead, the error message should read something like, "Please enter the date in the format MM/DD/YYYY." This clear guidance helps the user understand the issue and correct it immediately. Google's forms often provide such clear and helpful error messages, ensuring users can easily fix any mistakes.

4. Lack of Accessibility

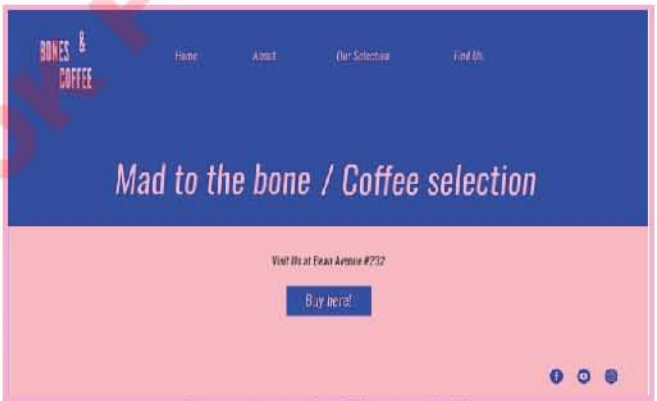


Fig.1.17: Lack of Accessibility

A lack of accessibility means that the system or device is not designed to accommodate users with disabilities, such as those who are visually impaired, hearing impaired, or have motor disabilities. Ignoring accessibility can exclude a significant portion of potential users and lead to a poor user experience for those with disabilities.

Solution: Implement accessibility features to make the system usable for people with a wide range of abilities. This includes adding screen reader support, keyboard navigation, alternative text for images, and options for adjusting text size and color contrast.

Example: Consider a website with small text and low contrast colors that are difficult for visually impaired users to read, as shown in Fig.1.17. By implementing accessibility features like text enlargement options, high-contrast color schemes, and ensuring the website is compatible with screen readers, the website becomes more accessible. Apple's website, for example, offers a high-contrast mode and supports screen readers, making it accessible to users with visual impairments.

1.1.4 Device/System Security

Security in the context of devices and systems refers to the measures and controls implemented to protect information and systems from unauthorized access, use, disclosure, disruption,

modification, or destruction. It aims to safeguard data and ensure that systems function as intended. The following are the key objectives of security.

Key Objectives:

Confidentiality: Ensuring that sensitive information is accessible only to authorized users.

Integrity: Protecting data from being altered or tampered with by unauthorized individuals.

Availability: Ensuring that systems and data are available for use when needed.

Principles of Security:

The development and maintenance of secure systems rely on several foundational principles, commonly referred to as the CIA triad—Confidentiality, Integrity, and Availability, as shown in Fig.1.18. These principles serve as the cornerstone for designing systems that protect data and ensure that it remains safe and accessible under various conditions.

Fig.1.18: Principles of Security

1. Confidentiality

Confidentiality involves protecting sensitive information from unauthorized access, ensuring that only those who have the appropriate permissions or clearance can view or use the data.

Example: Consider a company's database that stores customer credit card information. To keep this data confidential, the company might encrypt the credit card numbers before storing them in the database. This way, even if a hacker gains access to the database, they would not be able to read or misuse the credit card information without the decryption key.

2. Integrity

Integrity refers to the accuracy and consistency of data. It ensures that information remains unaltered during storage, transmission, or processing, except by those who are authorized to do so. This principle protects data from being tampered with or corrupted.

Example: When a user sends an important email, such as a contract, he/she might digitally sign it. A digital signature is a way to ensure that the content of the email has not been altered after it is sent. If the recipient receives the email with a valid signature, they can be confident that the content is exactly as intended and has not been tampered with.

3. Availability

Availability ensures that systems and data are accessible and usable when needed, even during or after a disruption. This principle involves designing systems to withstand and recover from failures or attacks quickly.

Example1: To guarantee availability, a business might implement more than one servers and data storage. For instance, if one server fails, another can take over immediately, preventing stoppage and ensuring that users can continue to access the system without interruption.

Security Policies and Best Practices:

Implementing strong security policies and best practices is essential for protecting devices and systems.

Data Protection Policies: These are guidelines for handling, storing, and transmitting sensitive information to prevent unauthorized access or disclosure. For example a company policy requiring encryption of all sensitive customer data before transmission.

Regular Security Audits: Audits are conducting thorough assessments of systems and networks to identify vulnerabilities and ensure compliance with security standards. For example an organization performing quarterly security audits to check for any gaps in their defenses.

Employee Training and Awareness: Educating employees on security risks and best practices to prevent human error, which is a common cause of security breaches.

Incident Response Planning: Preparing a plan to quickly and effectively respond to security incidents to minimize damage.

Learning from Security Breaches

Case Study: The Target Data Breach of 2013

Overview

In December 2013, Target Corporation, a major U.S. retail chain, suffered a significant data breach that compromised the personal and financial information of millions of customers.

Key Details

- How It Happened: Hackers accessed Target's network through a third-party vendor, using stolen credentials.
- Data Compromised: Approximately 40 million credit and debit card accounts and personal information of 70 million customers were affected.
- Financial Impact: The breach cost Target an estimated $162 million in legal fees, settlements, and security upgrades.
- Reputation Damage: Target's reputation suffered, leading to a decline in customer trust and sales.
- Regulatory Response: Target paid $18.5 million in a settlement with 47 states and implemented stricter security measures.

Lessons Learned

- Strengthen Cybersecurity: Companies must ensure robust security measures, especially with third-party vendors.
- Employee Training: Regular training on recognizing cyber threats is essential.
- Incident Response Plan: A clear plan for responding to breaches can mitigate damage.
- Regular Audits: Conducting security audits helps identify and fix vulnerabilities.

Assignment

Research two additional data breaches. For each breach, include:

i. Background: Company and context
ii. Date of Breach
iii. Data Compromised
iv. Financial Losses
v. Reasons for the Breach
vi. Remedies Implemented

This exercise will help you understand the importance of cybersecurity in today's digital world.

1.1.5 Device/System Accessibility

Accessibility refers to the design of devices, systems, and services in a way that they can be used by people with a wide range of abilities and disabilities. It involves making technology usable by everyone, including those with visual, auditory, physical, cognitive, or other types of impairments.

Key Aspects of Accessibility:

The following are the key aspects of Accessibility.

- **Inclusivity:** Ensuring that technology is accessible to the widest possible audience.
- **Adaptability:** Allowing users to customize the way they interact with the system to suit their needs.
- **Compliance:** Adhering to accessibility standards and legal requirements to ensure that products are accessible.

Principles of Accessibility:

Accessibility is a fundamental aspect of design that ensures technology can be used by everyone, including individuals with disabilities. The principles of accessibility guide the creation of systems and interfaces that accommodate a wide range of needs. By adhering to these principles, designers can create products that are inclusive and usable by the broadest possible audience.

1. Perceivability

Perceivability means that all users must be able to perceive the information and user interface components, regardless of their sensory abilities. This principle ensures that content is accessible to users with various sensory impairments, such as those who are blind, deaf, or colorblind.

Examples1: Websites should provide text alternatives (**alt text**) for images. This allows screen readers to describe the content of the images to users who are visually impaired. For example an online store includes alt text for product images so that a blind user can hear a description of the item, such as "Red wool sweater with V-neck and ribbed cuffs," through their screen reader.

Examples2: Captions and Transcripts for Audio/Video: Providing captions for videos and transcripts for audio content ensures that users who are deaf or hard of hearing can access the information. For example a YouTube tutorial video on cooking includes captions so that users who are hearing impaired can follow along with the instructions. As shown in Fig.1.19.

Fig.1.19: Transcripts for Audio/Video

2. Operability

Operability refers to the ability of users to navigate and use the interface, regardless of their physical or motor abilities. This principle focuses on ensuring that all users, including those with disabilities, can interact with the system effectively.

Example1: Keyboard Navigation: Users should be able to navigate through the interface using

just a keyboard, without needing a mouse.

Example2: Voice Commands: Implementing voice control features allows users to operate systems hands-free. For example a smart home system can be controlled via voice commands, enabling users with mobility impairments to turn on lights or adjust the thermostat without needing to physically interact with the controls. As shown in Fig.1.20.

Fig.1.20: Voice Commands

3. Understandability

Understandability means that the design should make it easy for all users to comprehend the information and know how to operate the interface. This principle focuses on creating content and interactions that are clear and straightforward.

Example1: Simple Language: A government website (www.fbr.gov.pk) provides instructions for filing taxes in simple language, making the process easier to understand for all tax payers, including those with learning disabilities, as shown in Fig.1.21.

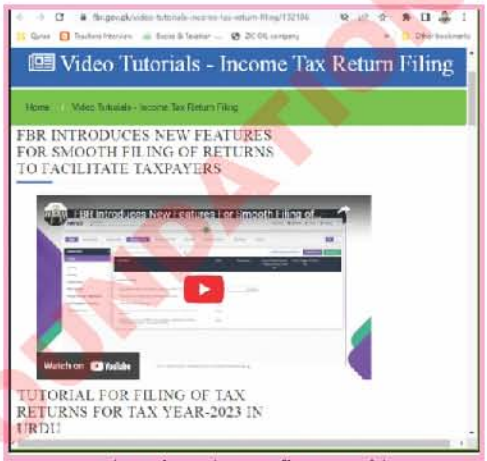


Fig.1.21: (www.fbr.gov.pk)

Example2: Error Prevention and Correction: An online form for registering an event highlights required fields and provides real-time feedback if a user forgets to fill out a section, along with instructions on how to correct the mistake.

4. Robustness

Robustness refers to the ability of content to be interpreted by a wide range of devices, including assistive technologies. This principle ensures that content remains accessible even as technologies evolve.

Example1: Cross-Browser Compatibility: Ensuring that a website functions correctly across different web browsers is crucial for accessibility. For example an educational platform tests its website to ensure it works well on Chrome, Firefox, Safari, and Edge(As shown in Fig.1.22), so that students using different browsers have the same experience.

Fig.1.22: Cross-Browser Compatibility

Example2: Support for Multiple Input Devices: A computer device is designed with multiple input options to cater to users with different preferences and needs. For example users can interact with the computer using voice commands for those who find it difficult to use a traditional mouse and keyboard. The touchscreen interface allows for gestures and touch interactions, providing an alternative for users who have difficulty with physical input devices.

1.1.6 Types of Disabilities and Accessibility Solutions

Designing technology that is inclusive and accessible to individuals with various disabilities involves understanding the unique challenges each type of disability presents and implementing tailored solutions.

The following are some accessibility solutions for different disabilities.

Visual Impairments:

Visual impairments encompass conditions such as blindness, low vision, and color blindness. Accessibility solutions for visual impairments aim to make content perceivable through alternative means.

Solutions and Examples:

Fig.1.23: Screen Readers

Screen Readers: These tools convert text and other visual elements into spoken words or braille. Effective implementation requires semantic HTML and ARIA (Accessible Rich Internet Applications) roles. For example **JAWS** (Job Access with Speech) or **NVDA** (Non Visual Desktop Access) reads aloud content on a web page, such as "Welcome to our website. Today's date is September 1, 2024." This enables blind users to navigate and interact with web content, as shown in Fig.1.23.

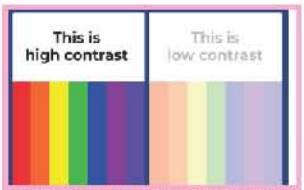


Fig.1.24: High Contrast

High Contrast Modes: Enhances readability by using contrasting colors to make text stand out against the background. For example Windows and macOS offer high contrast themes that switch text and background colors to improve visibility. Websites like BBC also provide high contrast modes for users with low vision.(Fig.1.24)

Zoom Functions: Allows users to magnify text and images on the screen to better accommodate low vision. For example Google Chrome and Mozilla Firefox browsers have built-in zoom features that enable users to increase the size of text and images, making them easier to read, as shown in Fig.1.25.

Fig.1.25: Zoom Functions

Hearing Impairments:

Hearing impairments can range from partial hearing loss to complete deafness. Accessibility solutions for hearing impairments focus on providing information through visual and textual means.

Solutions and Examples:

Closed Captions: Text that appears on screen to transcribe spoken dialogue, sound effects, and other audio cues in videos. For example YouTube provides closed captions for videos, enabling users who are deaf or hard of hearing to read the dialogue and sound effects as they watch, as shown in Fig.1.26.

Fig.1.26: Zoom Functions

Transcripts: Written versions of audio content, which allow users to read the content instead of listening. For example TED Talks provides transcripts for each talk, allowing users to read the content in addition to or instead of watching the video.

Sign Language Interpretation: Includes sign language videos that interpret spoken content into sign language. For example Microsoft Teams and other video conferencing tools allow users to include sign language interpreters in video calls. Similarly, news channels like PTV News enhance accessibility by adding sign language interpreters through a side video, making news broadcasts understandable for deaf viewers, as shown in Fig.1.27.

Fig.1.27: Sign Language Interpretation

Motor Impairments:

Motor impairments affect a person's ability to use traditional input devices like a mouse or keyboard. Accessibility solutions focus on providing alternative methods for interacting with devices.

Solutions and Examples:

Voice Recognition Software: Enables users to control their device and dictate text using voice commands. For example Dragon NaturallySpeaking allows users to navigate their computer, compose documents, and execute commands through voice recognition.

Alternative Input Devices: Includes specialized devices like adaptive keyboards, eye-tracking systems, and adaptive mouse.

On-Screen Keyboards: Virtual keyboards displayed on the screen that can be used with touchscreens or other input methods. For example On-Screen Keyboard provides a virtual keyboard that users can control using a mouse, touch screen, or other adaptive devices, as shown in Fig.1.28.

Fig.1.28: On-Screen Keyboards

1.2 Human-Computer Interaction (HCI)

Human-Computer Interaction (HCI) is a multidisciplinary field that studies how people interact with computers, digital devices, and software applications. The goal of HCI is to improve the interaction between users and systems by making these systems more user-friendly, efficient, and accessible. Good HCI can reduce user frustration, increase productivity, and ensure that technology meets the needs of its users.

Example: The design of a smartphone's touchscreen interface involves HCI principles to ensure that icons are large enough to be tapped easily, and that gestures (like swiping or pinching) are in-built.

1.2.1 Elements of Human-Computer Interaction

Human-Computer Interaction (HCI) is a complex and dynamic field that focuses on the design, evaluation, and implementation of interactive systems for human use. The effectiveness of any interactive system depends on several key elements that must be carefully considered during the design process. These elements include:

- Users
- Tasks
- Interface
- Environment

Users:

Users are the individuals who interact with a system. This involves studying their needs, abilities, limitations, and preferences. Users can vary widely in their skills, knowledge, experience, and expectations. Designers must consider the diversity of users, including factors such as age, cultural background, physical abilities, and technological proficiency.

Example: Designing a word processor for a professional writer involves understanding their need for robust editing tools, an organized workspace, and features that enhance their writing process, such as grammar checking, advanced formatting options, and distraction-free modes.

Tasks:

Tasks refer to the specific activities that users need to perform using the system. Identifying and understanding these tasks is essential for designing an interface that supports users in achieving their goals efficiently and effectively.

Example: On an e-commerce website, key tasks might include searching for products, filtering results, reading reviews, adding items to a shopping cart, and completing a purchase. Designers must ensure that each of these tasks is straightforward and easy to perform. As shown in Fig. 1.29.

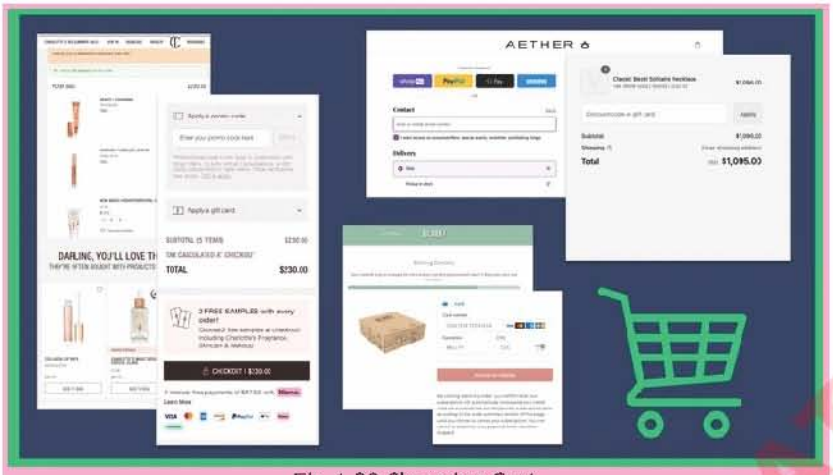


Fig.1.29 Shopping Cart

Interface:

The Interface is the point of interaction between the user and the system. It is where the design of the system comes to life, allowing users to interact with it through various input and output mechanisms. The interface can take many forms, including graphical user interfaces (GUIs), voice command systems, gesture-based interfaces, and more.

Example: The layout of a video editing software's interface is crucial for user productivity. Tools like cut, paste, and trim should be easily accessible, and the timeline should be carefully designed to allow for precise edits. A well-designed interface might also include customizable workspaces, where users can arrange tools and panels according to their workflow preferences. As shown in Fig.1.30.

Fig.1.30 Video Editing Software Interface

Environment:

The Environment refers to the physical and social context in which the interaction between the user and the system takes place. The environment can significantly influence how a system is used, and designers must consider these factors when creating interfaces. The environment includes the physical surroundings (e.g., lighting, noise levels, and available space) and the social context (e.g., whether the system will be used individually or collaboratively).

Fig.1.31: A Public Kiosk Interface

Example: A public kiosk interface, such as those found in airports or shopping malls, must be designed for quick, efficient use in potentially noisy, crowded environments. The interface should be simple, with large buttons and clear instructions to accommodate users who may be in a hurry or distracted by their surroundings. Additionally, the kiosk might need to be designed for use by standing users, meaning the screen should be at a comfortable height and angle. As shown in Fig. 1.31.

1.2.2 Usability in Human-Computer Interaction

Usability is a fundamental concept in Human-Computer Interaction (HCI), and it plays a crucial role in determining how effective and enjoyable a system is for its users. Usability refers to the extent to which a system can be used by specific users to achieve specified goals with effectiveness. A system with high usability is one that is easy to use and learn.

Key Components of usability in HCI:

Usability can be broken down into several key components, each of which contributes to the overall user experience in HCI. These components include Learnability, Efficiency, Memorability, Error Handling, and Satisfaction.

Learnability:

Learnability refers to how easy it is for new users to accomplish basic tasks the first time they encounter the system. A highly learnable system allows users to quickly grasp how to use it without needing extensive guidance or prior experience. This is especially important for applications or devices intended for a broad audience, where users may have varying levels of technical expertise.

Fig.1.32: Online Mobile Banking App

Example: Consider a mobile banking app designed for a diverse user base. A new user should be able to quickly learn how to navigate the app to check their account balance, transfer funds, or view transaction history. The use of onboarding tutorials or interactive guides can further enhance learnability, ensuring that users feel confident and capable from their first interaction as shown in Fig.1.32.

Efficiency:

Efficiency pertains to how quickly and effectively users can perform tasks once they have learned how to interact with the system. An efficient system allows users to achieve their goals with minimal time and effort.

Example: A professional graphic designer using design software like Adobe Photoshop needs to perform tasks such as editing images, creating layouts, and applying filters swiftly. Efficiency in this context means providing shortcut keys, customizable toolbars, and responsive tools that allow the designer to work with speed and precision.

Memorability:

Memorability refers to how easily users can remember how to use a system after a period of not using it. A system with high memorability ensures that users can return to it after some time and still remember how to perform key tasks without needing to relearn the interface.

Example: Imagine a user who only occasionally uses an online form submission system, such as a tax filing website. After several months, when they return to the site, they should be able to easily recall how to navigate the system, fill out the necessary forms, and submit their information. This can be facilitated through consistent design patterns, clear labeling, and a logical flow of steps.

Error Handling:

Error Handling involves the system's ability to prevent errors, guide users in correcting them, and help users recover from mistakes. Good error handling is essential for maintaining user trust and ensuring a smooth interaction, even when things go wrong.

Fig.1.33: Online booking system

Example: An online booking system (Fig.1.33) for airline tickets should prompt users to correct mistakes, such as entering an invalid date or leaving a required field blank. Instead of allowing the user to proceed with faulty data, the system might highlight the problematic fields in red and provide an error message explaining what needs to be corrected. Additionally, providing helpful suggestions or autofill options can prevent common errors, such as typos in names or addresses.

Useful Links!

Explore HCI Concepts: " Interaction Design Foundation offers free courses on HCI and usability principles."

Accessibility Tools: "Check out WebAIM for resources on making web content accessible to people with disabilities."

1.2.3 Common Problems in Human-Computer Interaction

Human-Computer Interaction (HCI) is a critical field that focuses on improving the interaction between users and computer systems. While significant progress has been made in designing

more user-friendly and efficient systems, several common problems still persist. These issues can negatively impact user experience, leading to frustration, errors, and even exclusion of certain user groups. Addressing these challenges is essential for creating systems that are accessible, user-friendly, and efficient for all users.

Interface Issues:

Poorly designed interfaces are one of the most common problems in HCI. The interface is the primary means through which users interact with a system, and its design can make or break the user experience. Common issues with interfaces include untidy layouts that could lead users to unclear navigation paths that make it difficult to locate important features. These problems can lead to confusion, inefficiency, and increased errors.

Example: Consider an e-commerce website with a poorly designed navigation menu. If the menu is untidy with too many options, users may struggle to find the categories they are looking for. For instance, a user searching for electronics might have to examine through an extensive list of unrelated categories, leading to frustration.

Accessibility Challenges:

Accessibility in HCI refers to the design of systems that can be used by people with a wide range of abilities and disabilities. However, many systems are not designed with accessibility in mind, leading to challenges for users with visual, auditory, motor, or cognitive impairments. These challenges often arise because designers do not fully consider the needs of all potential users.

Example: A website that lacks alternative text (Alt text feature) for images is inaccessible to users with visual impairments who rely on screen readers. These users might miss out on important content or context provided by images, making their experience incomplete or frustrating. Similarly, a mobile app that relies solely on sound for notifications may be unusable by individuals with hearing impairments.

Performance Issues:

Performance issues, such as slow or unresponsive systems, can severely impact the usability of a system. Users expect systems to respond quickly to their inputs, and any delay can lead to frustration and a decrease in productivity. Performance problems can be caused by a variety of factors, including poor system optimization, excessive resource use, or network latency. When systems are slow or fail to load content efficiently, users may abandon tasks or look for alternative solutions.

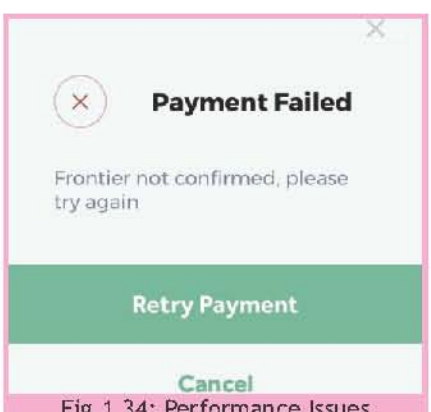


Fig.1.34: Performance Issues

Example: Imagine using an online banking app that takes an excessive amount of time to load account information or process transactions (Fig.1.34). Users may become impatient and concerned about the security of their transactions, leading them to avoid using the app in the future.

Poor Error Handling and Feedback:

Effective error handling and feedback are crucial components of a user-friendly system. When users encounter errors, they need clear, constructive feedback to understand what went wrong and how to fix it. Systems that provide poor or inadequate error messages can leave users confused and frustrated, leading to mistakes and inefficiency. Conversely, well-designed error handling helps users recover from mistakes quickly and with confidence, improving the overall user experience.

Example: Consider a form submission process on a website where the user is required to enter several fields of information. If the user leaves a required field blank or enters data in the wrong format, the system should provide an immediate, clear error message that explains what needs to be corrected. For instance, if an email field is left empty or filled incorrectly, the system should display a message like, **"Please enter a valid email address."** Without this guidance, users might not understand why their submission is not being accepted, leading to frustration and potentially abandoning the form altogether.

1.2.4. Methods for Improving Human-Computer Interaction

Improving Human-Computer Interaction (HCI) involves various strategies and methodologies aimed at creating systems that are more user-friendly, efficient, and complete. By addressing key aspects such as **user-centered design, accessibility, interface simplification, performance optimization, and error prevention,** developers can significantly enhance the users' experience.

User-Centered Design (UCD):

User-Centered Design (UCD) is a design philosophy that places the end users at the core of the design process. It is an iterative approach where user feedback is continuously gathered and integrated throughout the development cycle. This method ensures that the final product aligns closely with the users' needs, preferences, and limitations, resulting in a more effective and satisfying user experience.

Fig.1.35: User-Centered Design (UCD)

Example: Consider the development of a new educational software intended for use in schools. By involving teachers and students in the design process from the beginning, developers can gather valuable feedback on features such as interactive elements, user interfaces, and content organization. Early testing might reveal that students prefer interactive quizzes over static reading material, leading to adjustments that enhance engagement and learning outcomes. As shown in Fig.1.35.

Accessibility Enhancements:

Accessibility enhancements focus on making systems usable for individuals with a range of disabilities. This involves designing systems that are perceivable, practicable, understandable, and user-friendly, ensuring that people with visual, auditory, motor, or cognitive impairments can effectively interact with the system.

Example: An educational platform that includes captions and transcripts for all video content

improves accessibility for students with hearing impairments. Additionally, incorporating keyboard navigation and screen reader compatibility ensures that users with motor impairments or visual disabilities can interact with the platform effectively.

Simplifying User Interfaces:

Simplifying user interfaces involves reducing unnecessary complexity and presenting information in a clear, concise manner. A well-designed interface should prioritize essential features and streamline interactions to minimize user effort and cognitive load.

Example: A streamlined email application that features a minimalist design with only the most commonly used tools (such as compose, inbox, and search) can significantly enhance user efficiency. By avoiding complex menus and excessive options, users can quickly access and manage their emails without unnecessary distractions.

Improving Performance

Optimizing system performance is essential for ensuring that applications run smoothly and respond promptly to user inputs. Performance issues such as slow load times or unresponsive interfaces can detract from the user experience and hinder productivity.

Example: Reducing the load time of a large e-commerce website by optimizing images, compressing files, and implementing efficient coding practices can greatly improve user experience.

Error Prevention and Recovery:

Effective error prevention and recovery mechanisms are crucial for minimizing the impact of user mistakes and system errors. Systems should be designed to prevent errors whenever possible and provide clear feedback and recovery options when errors do occur.

Example: A spreadsheet application that prompts users to save changes before closing the file helps prevent data loss. Additionally, an online form with real-time validation checks for required fields and provides immediate feedback when errors are detected can help users correct mistakes before submission.

1.2.5 Ethical, Social, Economic, and Environmental Implications in HCI

Human-Computer Interaction (HCI) is not just about creating systems that are user-friendly and efficient; it also encompasses broader considerations that affect society at large. Ethical, social, economic, and environmental implications play a crucial role in the development and deployment of any new system. Understanding these aspects is essential for designing systems that are not only effective but also responsible and sustainable.

Ethical Implications:

Ethical implications in HCI address how technology affects users' rights, privacy, and dignity. Ethical design ensures that technology is developed and used in ways that are respectful and fair, protecting users from harm and misuse.

- **Data Privacy:** One of the major ethical concerns is how user data is collected, stored, and

used. Developers must ensure that user data is handled with confidentiality and used only for its intended purpose. Users should have control over their data and be informed about how it is used.

Example: Social media platforms such as, Instagram, Facebook, WhatsApp and Twitter (Fig.1.36), collect vast amounts of personal data from users. Ethical practices require these platforms to implement strong data protection measures, obtain explicit user consent before collecting data, and provide clear options for users to manage their privacy settings.

Fig.1.36: Social media platforms

- **User Consent:** Users should be informed and provide consent before engaging with systems that collect or use their data. Consent should be clear and informed.

Example: Mobile apps that request access to personal information, such as location or contacts, should provide users with a clear explanation of why this access is necessary and obtain explicit consent before proceeding.

Social Implications:

Social implications refer to how technology influences communication, social interactions, and access to information. HCI can have both positive and negative effects on social structures and relationships.

For Your Info!

Social Media and Mental Health: "Research has shown that excessive use of social media can lead to anxiety and depression, particularly among young people. Balanced use is a key."

Example: Smartphones and social media enable instant messaging and video calls, fostering global communication. However, excessive screen time and reliance on digital interactions can affect face-to-face communication skills and lead to issues such as social isolation.

Economic Implications:

Technology impacts the economy by creating new opportunities and challenges. Understanding these implications helps in designing systems that support economic growth while addressing potential disruptions.

Example1: The rise of e-commerce platforms like Amazon (Fig.1.37) has generated jobs in tech, logistics, and customer service. These platforms have also led to the growth of related industries, such as digital marketing and online payment systems.

Fig.1.37: Amazon

Example2: The automation of manufacturing processes through robotics (Fig.1.38) can lead to increased production efficiency but may also result in job losses for assembly line workers. Training and reskilling programs can help mitigate these effects.

Fig.1.38: Manufacturing through robotic

Environmental Implications:

The environmental impact of technology involves considering energy consumption, resource use, and waste management. Sustainable practices in HCI are crucial for minimizing the ecological footprint of technology.

Example1: Companies like Google and Microsoft are investing in renewable energy to power their data centers, aiming to reduce their carbon footprint and promote sustainability.

Example2: Electronics manufacturers are implementing take-back programs and designing products with recyclable materials to reduce e-waste. Initiatives such as Apple's recycling program help ensure that old devices are properly recycled.

1.3 Trade-offs between Usability and Security in Computing Systems

When designing computing systems, achieving a balance between usability and security is crucial. Both factors are essential, but they often come into conflict. The following is a detailed look at the trade-offs between usability and security, along with recommendations for cybersecurity measures considering efficiency, cost, privacy, and ethics.

1.3.1 Trade-offs between Usability and Security

Usability focuses on how easy and intuitive a system is for users, while **security** aims to protect the system from unauthorized access and attacks. The trade-offs arise because enhancing security can sometimes degrade usability, and vice versa.

The following table represents the trade-offs between usability and security in computing systems, along with examples.

Aspect	Security	Usability	Example
Authentication Complexity	Enhances security by adding extra verification steps like multi-factor authentication (MFA), making unauthorized access much harder.	Reduces usability as MFA requires additional steps like entering codes from a phone, slowing down user access and causing potential frustration.	A user logging into a bank account needs to enter a password and then a code sent to their phone, increasing security but delaying access.
Access Controls	Protects sensitive data by restricting user permissions, ensuring only authorized individuals can access or modify information.	Hinders usability when users frequently encounter permission issues, requiring them to request access, which can slow down their work.	An employee needs frequent permission from IT to access certain files, delaying their ability to complete tasks efficiently.

Aspect	Security	Usability	Example
Encryption	Secures data by converting it into a format only accessible with a decryption key, protecting against unauthorized access even if data is intercepted.	Impacts usability as encryption can slow down data access and processing, and managing encryption keys can be complex and time-consuming.	Encrypting patient records keeps them secure but may slow down access, which can be challenging during emergencies.
Regular Updates and Patches	Maintains security by fixing vulnerabilities and protecting against new threats through regular updates.	Disrupts usability as updates may require system restarts or interrupt work, and they can sometimes introduce compatibility issues.	Frequent software updates may interrupt employees' work, requiring them to restart their systems, which can be inconvenient during busy times.

1.3.2 Recommendations for Cybersecurity measures

When designing and implementing cybersecurity measures, it is important to consider various factors to strike a balance between security and user experience. The key factors to consider are efficiency, cost, privacy, and ethics. Each of these plays a crucial role in determining how effective and sustainable the security solutions will be.

Efficiency: Efficiency in cybersecurity involves implementing solutions that protect systems without significantly hindering user productivity. The following computing solutions illustrate the principle of efficiency in cybersecurity measures.

- **Auto-Save and Backup Systems:**

Automatically saving work and backing up files regularly helps ensure that data is not lost in case of a system crash or cyberattack. This increases efficiency by reducing the risk of lost work and the need to redo tasks.

Example: Google Docs automatically saves documents in real-time to Google Drive, so users don't have to worry about losing their work if their computer shuts down unexpectedly.

- **Password Managers:**

Password manager stores and autofill strong, unique passwords for different websites, reducing the time and effort users spend managing multiple passwords while enhancing security.

Example: Using a password manager like LastPass or Bitwarden, users can automatically log into websites without needing to remember complex passwords, making the login process quicker and more secure.

Cost: Cost considerations are critical when determining the scope and scale of cybersecurity measures. The goal is to implement effective security within budget constraints. The following computing solutions illustrate the principle of cost in cybersecurity measures.

Helping Tip!

Choosing Security Tools: Consider open-source security tools as a cost-effective alternative to proprietary software, especially for small businesses or personal use.

Freemium Security Software:

Many security software providers offer free basic versions of their products, which can be sufficient for small businesses or individual users, allowing them to secure their systems without incurring high costs.

Example: Avast offers a free version of its antivirus software that provides essential protection against viruses and malware, which is often adequate for home users.

Cloud-Based Security Services:

Cloud-based security solutions often have lower upfront costs compared to on-premises hardware, as they eliminate the need for expensive infrastructure and maintenance.

Example: A small business might choose to use a cloud-based firewall service, such as Cloudflare, to protect their website from attacks, avoiding the cost of purchasing and maintaining their own hardware.

Open Source Solutions:

Open source tools and frameworks offer cost-effective alternatives to expensive proprietary software, providing robust security features without significant financial investment.

Example: The OWASP (Open Web Application Security Project) provides a variety of open-source tools that help developers identify and mitigate security vulnerabilities in web applications, reducing the need for expensive commercial solutions.

Privacy: Privacy protection is a foundation of cybersecurity, ensuring that users' personal data is handled responsibly and securely. The following computing solutions illustrate the principle of privacy in cybersecurity measures.

Private Browsing Modes:

Private browsing, or "incognito" mode, in web browsers prevents the storage of browsing history, cookies, and other data, helping protect user privacy on shared or public computers.

Example: When using a public computer at a library, users can enable incognito mode in Chrome or Firefox to ensure their browsing activity isn't stored, protecting their privacy.

Anonymization and Encryption:

These techniques protect user privacy by ensuring that even if data is accessed by unauthorized parties, it cannot be traced back to individuals or easily deciphered.

Example: Healthcare providers often encrypt patient records and anonymize data before sharing it for research purposes, ensuring that personal health information remains confidential.

Two-Factor Authentication (2FA):

2FA adds an extra layer of security by requiring users to provide two forms of identification (such as a password and a code sent to their phone), which helps protect their accounts from unauthorized access. As shown in Fig. 1.39.

Fig.1.39: Two-Factor Authentication (2FA)

Example: When logging into their email, a user might enter their password and then receive a text message with a one-time code, ensuring that only they can access their account even if someone else knows their password.

Ethics: Ethical considerations in cybersecurity involve respecting user rights and maintaining trust through transparent and fair practices. The following computing solutions illustrate the principle of ethics in cybersecurity measures.

User-Friendly Privacy Settings:

Designing privacy settings that are easy to understand and adjust ensures that all users, regardless of their technical knowledge, can control how their data is used.

Example: Facebook provides a simple privacy checkup tool that guides users through their privacy settings step by step, making it easy to adjust who can see their posts and personal information.

Accessibility Features for Disabled Users:

Ensuring that security measures are accessible to users with disabilities is an ethical responsibility, helping to prevent exclusion from digital services.

Example: An online banking platform might offer voice recognition for account access, making it easier for users with physical disabilities to log in securely without needing to type a password as shown in Fig.1.40.

Fig.1.40: Online Banking-Voice Recognition

Useful Links

Cybersecurity and Usability Research: Usable Security and Privacy Group at Carnegie Mellon University explores ways to design systems that are both secure and user-friendly.

Summary

- **Usability:** Measures how effortlessly and effectively users can interact with a system.
- **Moderated Usability Testing:** Involves a facilitator guiding and observing users.
- **Unmoderated Usability Testing:** Users complete tasks independently, often remotely.
- **Qualitative Usability Testing:** Focuses on user experiences and feedback.
- **Quantitative Usability Testing:** Measures usability with numerical data.
- **Remote Usability Testing:** Testing conducted from the user's location.
- **A/B Testing:** Compares two versions of a system to evaluate performance.
- **Complex Interfaces:** Overwhelming users with too much information or complexity.
- **Inconsistent Navigation:** Variability in navigation elements across the system.
- **Poor Error Messages:** Unclear or unhelpful error messages.
- **Lack of Accessibility:** Failure to accommodate users with disabilities.
- **Security:** Measures to protect information and systems from unauthorized access and threats.
- **Malware:** Malicious software designed to harm systems.
- **Phishing:** Deceptive attempts to obtain sensitive information.
- **Man-in-the-Middle (MitM) Attacks:** Interception and potential alteration of communication.
- **Denial of Service (DoS) Attacks:** Overloading a system to make it unavailable.
- **Insider Threats:** Security risks from within an organization.
- **Accessibility:** Design to ensure usability by people with a range of abilities and disabilities.
- **Visual Impairments:** Solutions like screen readers, high contrast modes, and zoom functions.
- **Hearing Impairments:** Solutions like closed captions, transcripts, and sign language interpretation.
- **Motor Impairments:** Solutions like voice recognition software, alternative input devices, and on-screen keyboards.
- **Human-Computer Interaction (HCI):** A multidisciplinary field studying interactions between people and computers to make systems more user-friendly, efficient, and accessible.
- **Users:** Individuals interacting with a system; their needs, abilities, limitations, and preferences must be considered.

- **Tasks:** Specific activities users perform using the system.
- **Interface:** The point of interaction between the user and the system, including GUIs, voice commands, and gesture-based controls.
- **Environment:** The physical and social context of the interaction, such as lighting, noise, and whether the use is solitary or collaborative.
- **Interface Issues:** Problems with poorly designed interfaces, such as untidy layouts and unclear navigation.
- **Accessibility Challenges:** Issues in making systems usable for people with disabilities, including visual, auditory, motor, or cognitive impairments.
- **Performance Issues:** Problems such as slow or unresponsive systems affecting usability.
- **Poor Error Handling and Feedback:** Inadequate error messages and feedback leading to user frustration.
- **User-Centered Design (UCD):** A design approach focusing on users' needs and feedback throughout the development cycle.
- **Accessibility Enhancements:** Designing systems to be usable by people with various disabilities.
- **Simplifying User Interfaces:** Reducing complexity and presenting information clearly.
- **Improving Performance:** Optimizing system performance to ensure smooth operation.
- **Error Prevention and Recovery:** Mechanisms to prevent and recover from errors effectively.
- **Ethical Implications:** Concerns about user rights, privacy, and dignity.
- **Social Implications:** The impact of technology on communication and social interactions.
- **Economic Implications:** The effect of technology on the economy, including job creation and disruption.
- **Environmental Implications:** Considering energy consumption, resource use, and waste management in technology.
- **Authentication Complexity:** Balancing security measures like multi-factor authentication with user convenience.
- **Access Controls:** Restricting data access to protect security while considering the impact on user workflow.
- **Encryption:** Protecting data with encryption while managing the potential impact on performance.
- **Regular Updates and Patches:** Maintaining system security through updates while managing disruptions to usability.

Exercise

Select the best answer for the following Multiple-Choice Questions (MCQs).

1. A software product receives negative user reviews due to being difficult to use. Which of the following factors is least likely responsible?
   a. Inefficiency b. Ineffectiveness c. High satisfaction d. High complexity
2. A web form alerts the user when an email address is entered incorrectly before submission. Which usability principle does this best represent?
   a. User satisfaction b. Error prevention
   c. Performance optimization d. Feedback mechanism
3. If you want to observe users' emotional reactions and gather verbal feedback while they perform tasks, which usability testing method would be most suitable?
   a. Moderated Usability Testing b. A/B Testing
   c. Unmoderated Usability Testing d. Remote Usability Testing
4. Which practice best supports confidentiality in a hospital's patient records system?
   a. Ensuring the system is always online
   b. Allowing only authorized doctors to access patient data
   c. Letting patients edit their own files freely
   d. Encrypting data for public access
5. A user receives a seemingly legitimate email from their bank requesting login details. This is an example of:
   a. Phishing b. Malware c. Spyware d. Spoofing
6. A company wants to protect client data during transmission over the internet. Which security method should they apply?
   a) Firewall b. Encryption c. Access Control d. Antivirus Software
7. What is the main objective of Human-Computer Interaction (HCI)?
   a. System performance b. User interaction
   c. Software complexity d. Hardware costs
8. Which of the following is NOT considered a key element of HCI?
   a. Users b. Tasks c. Budget d. Interface
9. Which of the following is a common issue related to accessibility in HCI?
   a. Slow load times b. Inadequate error messages
   c. Lack of alternative text for images d. Complicated authentication processes
10. Which method involves placing the end users at the core of the design process in HCI?
    a. Performance Optimization b. Error Prevention
    c. User-Centered Design (UCD) d. Simplifying User Interfaces
11. What does the term 'interface' refer to in HCI?
    a. The physical device used b. Point where user interacts with system
    c. The software programming language d. The hardware configuration

12. Why is multi-factor authentication (MFA) often seen as a trade-off between security and usability?
    a. It's expensive to implement
    b. It improves system speed
    c. It increases security but can slow down user access
    d. It reduces the need for encryption
13. Which issue is caused by encryption?
    a. Processing speed b. Data access c. Key management d. Interface
14. How can auto-save improve cybersecurity efficiency?
    a. Recovery b. Encryption c. Complexity d. Management
15. Which ethical consideration involves transparency in data use?
    a. Efficiency b. Cost c. Privacy d. Accessibility

Give short answers to the following Short Response Questions (SRQs).

1. What are the key components of usability in HCI?
2. List and briefly describe the four key elements of HCI.
3. What is the difference between moderated and unmoderated usability testing?
4. What does 'error handling' mean in the context of HCI?
5. What is meant by 'User-Centered Design (UCD)' in HCI?
6. Give three benefits and two drawbacks of using password managers?
7. How does user feedback contribute to effective system design?
8. Provide an example of how user feedback led to a design change.
9. Give two considerations for designing accessible systems for users with disabilities.
10. Suggest three features that should be included in a new educational software application to enhance accessibility for users with disabilities.

Give long answers to the following Extended Response Questions (ERQs).

1. Describe how accessibility challenges impact the usability of a digital system and provide examples of improvements that can address these challenges.
2. What are the main elements involved in Human-Computer Interaction (HCI) design? Explain each element's role in creating effective interactive systems.
3. Explain the ethical considerations in Human-Computer Interaction (HCI) with examples. Why are these considerations important in the design and deployment of computing systems?
4. You are developing a new software tool for remote team collaboration. What usability and security features would you integrate to support efficient teamwork while protecting sensitive data?
5. You are tasked with designing an e-commerce website. The site must be both user-friendly and secure. List three features you would include to achieve this balance and explain why.
6. Discuss the trade-offs between usability and security in computing systems. Provide examples of how these trade-offs might manifest in real-world applications.

7. You are tasked with designing an e-commerce website. The site must be both user-friendly and secure. List three features you would include to achieve this balance and explain why.
8. How do, security measures like multi-factor authentication (MFA) and encryption impact usability, and how can designers balance these trade-offs?

Activity 1: Usability and Accessibility Design Challenge

Objective: To understand and apply principles of usability and accessibility in system design.

Group Project: Divide students into small groups and task them with designing a mock interface for a hypothetical app that must be user-friendly and accessible to people with disabilities.

Instructions:

- Choose an application (e.g., a fitness tracker, e-learning platform, or social media app).
- Apply usability principles (simplicity, consistency, feedback, error prevention, learnability) to the design.
- Incorporate accessibility features (e.g., voice guidance, large buttons, high-contrast modes).
- Create wireframes or mockups of the interface.
- Present the design to the class and explain how the features support both usability and accessibility.

Activity 2: Security Measures Evaluation

Objective: To analyze the impact of different security measures on system usability and functionality.

Individual Research and Presentation: Students research and present on different security measures and their potential impacts on usability.

Instructions:

- Choose one security measure (e.g., multi-factor authentication, encryption, or firewalls).
- Explain how this measure works and its purpose.
- Discuss how it might affect the usability of a system (e.g., increased steps for user login, potential for user confusion).
- Provide real-world examples of systems where this trade-off is evident.
- Present findings to the class and suggest ways to mitigate usability issues while maintaining strong security.

Activity 3: Design an Inclusive Interface

Objective: Students will design a user interface for a specific target audience considering different types of disabilities.

Instructions:

- **Identify User Needs:** Divide students into groups and assign each group a different disability (visual, auditory, motor).
- **Design Brief:** Each group will design an interface for a common application (e.g., a public information kiosk, a health tracking app) that caters to their assigned disability.
- **Design Elements:** Ensure designs include features like screen readers, high contrast modes, alternative input methods, etc.
- **Presentation:** Groups present their designs to the class, explaining how their interface meets accessibility standards.

Activity 4: Usability Testing Simulation

Objective: To practice and understand different usability testing methods.

Simulation Exercise: Conduct a simulated usability test of a given website or app, applying different testing methods.

Instructions:

1. **Preparation:** Select a website or app to test.
2. **Testing Methods:** Use different types of usability testing methods (moderated, unmoderated, qualitative, quantitative) to evaluate the system. For example:
   - **Moderated Test:** Have a classmate use the system while you observe and take notes.
   - **Unmoderated Test:** Have another student perform tasks on their own and record their interactions.
   - **Qualitative Test:** Conduct interviews with users to gather their feedback.
   - **Quantitative Test:** Measure task completion times and error rates.
3. **Analysis:** Compile the results from different tests and identify common usability issues.
4. **Reporting:** Prepare a report summarizing the findings and suggest improvements based on the testing.

Activity 5: Evaluating and Improving an Educational App's Interface

Suggested App: Taleemabad

- Taleemabad is a popular educational app aimed at teaching children in a fun and interactive way. It provides engaging content in subjects like mathematics, English, and Urdu through games and animated lessons.

Short Activity Overview:

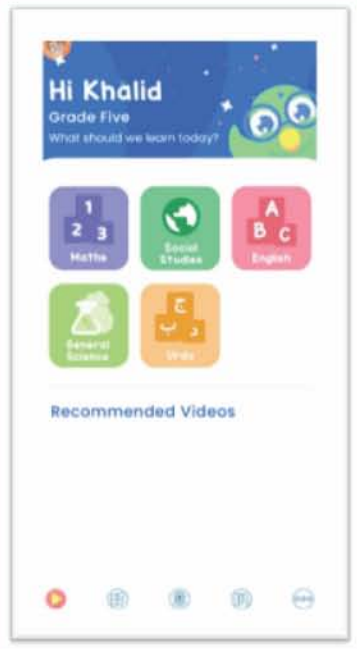
Evaluate the **Taleemabad app's** interface, identify areas of improvement, and suggest user-centered changes that could enhance learning and usability.

Questions for Evaluation:

1. Is the app easy to navigate for both children and parents?
2. Is the app's design colorful and engaging for children?
3. Does the design support ease of use, or is it overwhelming?
4. Can children easily start and finish a lesson?
5. Is the app's flow intuitive for young users without adult assistance?

Suggestions for Improvement:

1. What would you suggest to make the app more user-friendly?
2. Propose 1-2 design changes that would improve the app's usability for young learners.

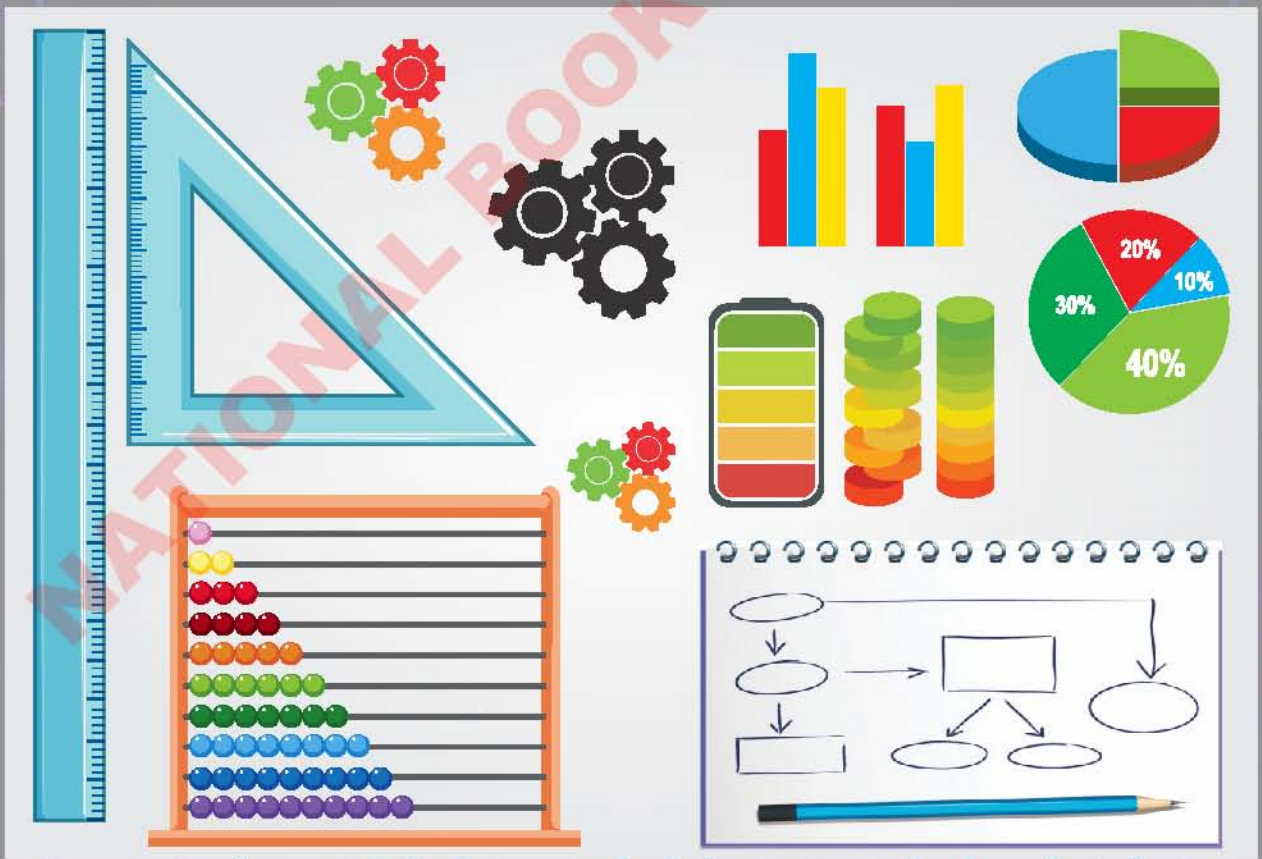


2 Computational Thinking & Algorithms

Learning Outcomes

At the end of this unit students will be able to:

- Identify and apply complex algorithms on data structures such as trees and binary search
- understand and evaluate the computational solutions in terms of efficiency , clarity and correctness



Introduction

Computational Thinking (CT) is a problem-solving process rooted in computer science principles. It enables individuals to tackle complex challenges by breaking them down into manageable parts and approaching them logically and systematically. Although CT is fundamental to computer science, its application extends far beyond – into science, engineering, business, medicine, and everyday life.

The core pillars of computational thinking include:

- **Decomposition:** breaking a problem into smaller, more manageable components.
- **Pattern Recognition:** identifying similarities or trends to simplify complex tasks.
- **Abstraction:** focusing on important information and ignoring irrelevant details.
- **Algorithm Design:** developing clear, step-by-step instructions to solve a problem.

Together, these elements form the foundation for designing effective and efficient computational solutions. At this advanced level, Computational thinking is not only applied to design algorithms but also is used to analyze and evaluate these solutions in terms of correctness, clarity, and efficiency. Understanding data structures is central to this process, as they greatly influence how well a solution performs.

In this chapter, it will be explored how computational thinking integrates with algorithms and data structures to create solutions that are not only correct but also optimized and scalable.

2.1 Data Structures

Data structure defines the way for organizing and storing data so that it can efficiently be accessed and modified. They provide a framework for managing and organizing data in computer programs, allowing for various operations to be performed effectively.

Data structures are essential for efficiently organizing and managing data in computer programs. They dictate how data is arranged in memory and impact the performance of various operations like insertion, deletion and retrieval. For instance, arrays provide quick access to elements by index, while linked lists offer flexibility in dynamic data manipulation. The choice of data structure e.g. stacks and queues, trees and graphs depends on the specific needs of the application and the operations it performs.

Following are some common data structures:

2.1.1 Arrays

Arrays are the simplest data structure used in programming that contain different elements of same size. Contiguous memory locations are used to store these elements. The memory locations are identified by indexes as shown in Fig. 2.1.

Fig.2.1: Arrays

The arrays are considered efficient because they do not require additional data (metadata) to store the elements. Additionally, because the memory locations are contiguous, therefore, it makes accessing the elements fast.

However, the arrays are fixed in size and it requires to declare its size at the start. At some later stage, when arrays need to be resized, it is costly to do so, because it may require creating new arrays with larger size and shifting of elements from old array to new. Similarly, the arrays could not be a good choice when elements are of different type.

The arrays are of two different types:

One-Dimentional (1-D) Array: In 1-D array , various elements (of same type) are stored in a single list. The element can be reached by knowing only a single index. One-Dimentional (1-D) Array example is shown in Fig.2.2.

Fig.2.2: One-Dimentional (1-D) Array

Two-Dimentional (2-D) Array: The 2-D array is also known as matrix, where the elements are stored in the form of matrix or table. The elements can only be reached by knowing two indexes e.g. row and column. Fig.2.3 shows an example of Two-Dimentional (2-D) Arrays.

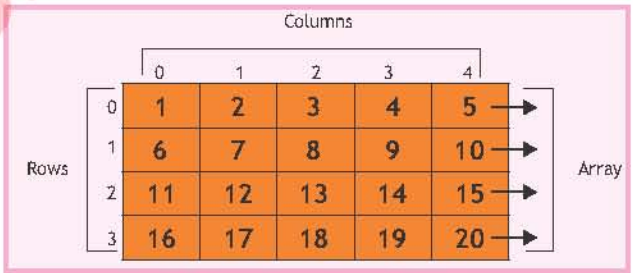


Fig.2.3: Two-Dimentional (2-D) Array

2.1.2 Linked Lists

Linked Lists are basic data structures in which elements (called nodes) are connected to each other using the concept of pointers. The nodes contain the values to be stored and pointer is the reference (memory address) of the next node in the sequence as shown in Fig.2.4.

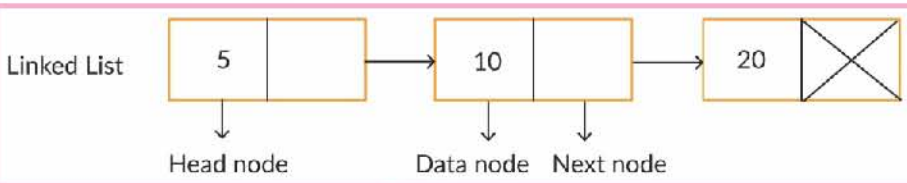


Fig.2.4: Linked Lists

The linked lists are capable to store different types of data and it also makes insertion and deletion operations easy as compared to arrays. The insertion/deletion could be performed at

any position in the sequence of linked list. Another advantage of linked lists is that they are dynamic in size (no need to specify the size of linked list at the start, rather they can grow and shrink as per requirement.

Singly Linked List

Singly linked list is basic and simple type of list, it maintains two parts in each node.

- data to be stored
- reference to next node in the sequence

Fig.2.5 shows an example scenario:

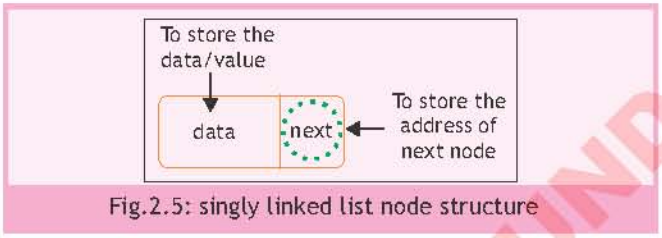


Fig.2.5: singly linked list node structure

The last node of the linked list contains NULL because it does not refer to any other node as shown in Fig.2.6.

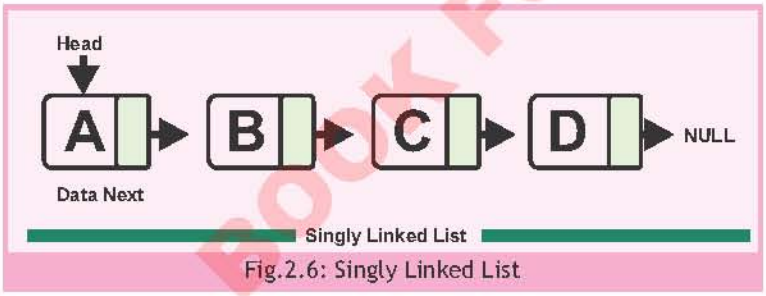


Fig.2.6: Singly Linked List

Doubly Linked List

The doubly linked list is complex as compared to singly linked list but it also provides more advantages. The major advantage is that it allows traversal in both directions and for that purpose it needs to maintain pointers for both directions e.g. reference to previous node and reference to next node. Therefore, the node in doubly linked list have three parts (Fig.2.7):

- reference to previous node in the sequence
- data to be stored
- reference to next node in the sequence

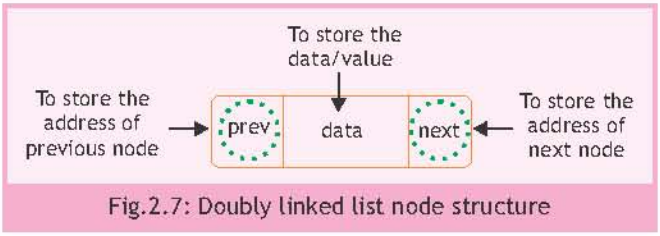


Fig.2.7: Doubly linked list node structure

Fig.2.8 shows an example scenario:

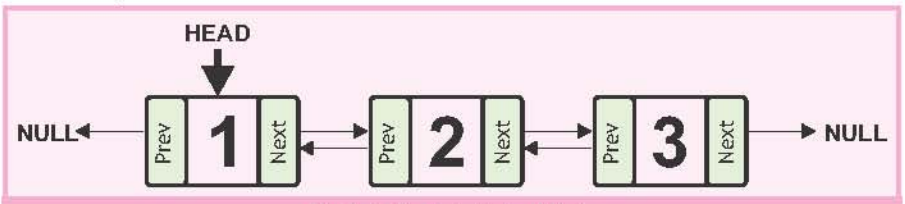


Fig.2.8: Doubly linked list

The first node and the last node in the sequence of linked list contains NULL because it does not refer to any other nodes.

Circular Linked List

It is a special type of linked list where all nodes are connected in a circle - forming a loop. Unlike other types, where last node points to NULL, the last node in circular linked linked is connected back to the first node. Therefore, while traversing in circular linked linked, it never reached to NULL. The structure of Circular Link List is depicted in Fig.2.9.

Fig.2.9: Circular linked list

The circular linked list could be made from both of the above types e.g. Singly linked list or doubly linked list. Fig.2.10 shows formation of a circular doubly linked list.

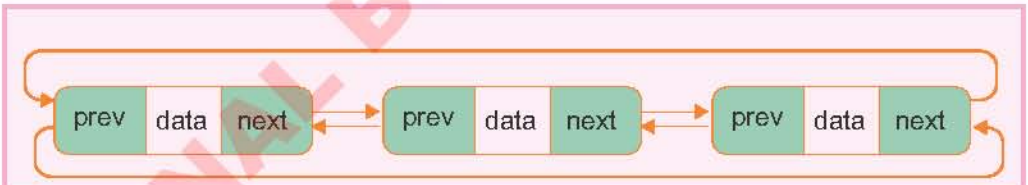


Fig.2.10: Circular Doubly linked list

2.1.3. Stacks

The stacks use the LIFO (Last-In, First-Out) principle, where the most recently added element (last element in the atack) is the first to be removed. This is important for understanding algorithms that need to manage tasks in a specific order. The LIFO principle makes it simple because the operations on the stack are performed on top position of the stack. Stack are considered memory efficient, the size of the stack could be fixed or dynamic, it depends upon the implementation.

Following are the basic operation that can be performed on stack:

- Push: Add a new element at the top of stack
- Pop: remove existing element from the top of stack

- Top: return the top element without removing it
- IsEmpty: Check, if stack is empty
- IsFull: Check, if stack is full

The working of Stack with core operations is shown Fig.2.11.

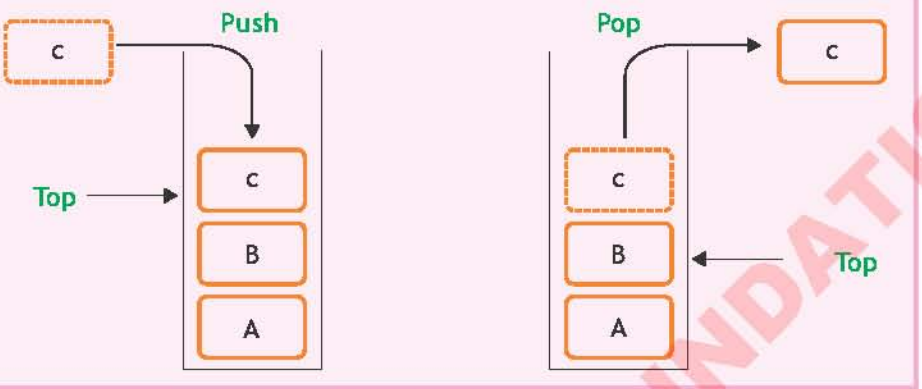


Fig.2.11: Stacks and its operations

2.1.4. Queues

The Queues use the FIFO (First-In, First-Out) principle, where the first element added is the first to be removed. This is useful for managing tasks that need to be processed in the order they arrive. Unlike stack, the operations on queues occur at two ends. The new elements are added at the end of queue, whereas the existing elements are removed from the start of the queue.

Following are the basic operation that can be performed on queues:

- Enqueue: Add a new element at the end(back/tail/rear) of the queue
- Dequeue: remove existing element at the start (front/head) of the queue.
- Front: return the start element without removing it
- IsEmpty: Check, if queue is empty
- IsFull: Check, if queue is full

The working of Queues with core operations is shown Fig.2.12.

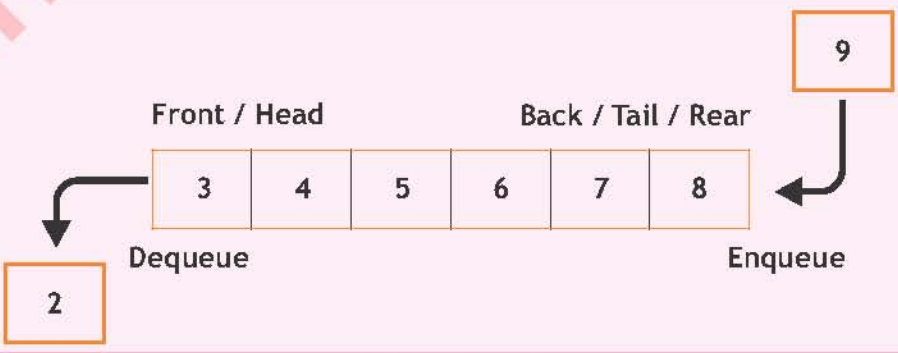


Fig.2.12: Queues

They are key consideres key data structure in algorithms like breadth-first search (BFS) and in various real-world scenarios like scheduling and buffering.

2.1.5. Graphs

The graph data structure also consist of nodes (called vertices) connected to each other by edges.

The graphs model complex relationships and networks. Graphs help in understanding problems related to social networks, routing and resource management. They are essential for algorithms related to network analysis, shortest paths (e.g., Dijkstra's algorithm) and connectivity as shown in Fig.2.13.

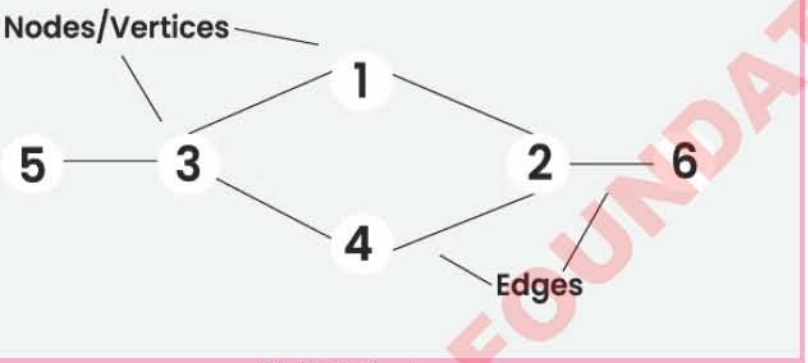


Fig.2.13: Graphs

Following are the basic operation that can be performed on graphs:

- Add Vertex: Add a new node to the graph
- Add Edge: Create connection between two nodes
- Remove Vertex/Edge: Delete node or edge from the graph
- Traversal: Visiting graph in specific order.

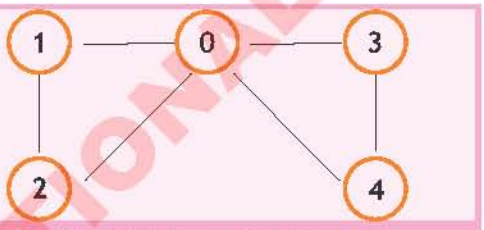


Fig.2.14: Connected Graph

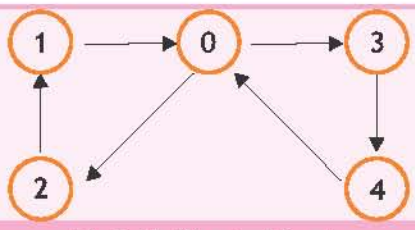


Fig.2.15: Directed Graph

Primarily, there are two types of graphs:

Connected Graph: A graph where at least one path exists between every pair of node. If you can start at any node and reach any other node (in any direction), it is connected. An example is shown in Fig.2.14.

Directed Graph: A graph where edges have direction associated with them as represented in Fig.2.15.

2.1.6 Trees

The trees represent hierarchical structures consisting of nodes. The top most node is called root. The directly connected nodes make a relationship of parent-child. The higher node in hierarchy is parent, whereas, lower is child. The node which do not have child is referred as leaf node. The relationship between nodes is referred as edge. Each node consist of two things: value and pointer to its child. An example is shown in Fig.2.16.

The trees are crucial for understanding complex relationships between data. Trees are used in various algorithms such as traversal (in-order, pre-order, post-order) and are foundational for understanding more advanced data structures.

In computer Science, Directory structure in an operating system or Tag structure in HTML Language are best examples to understand trees.

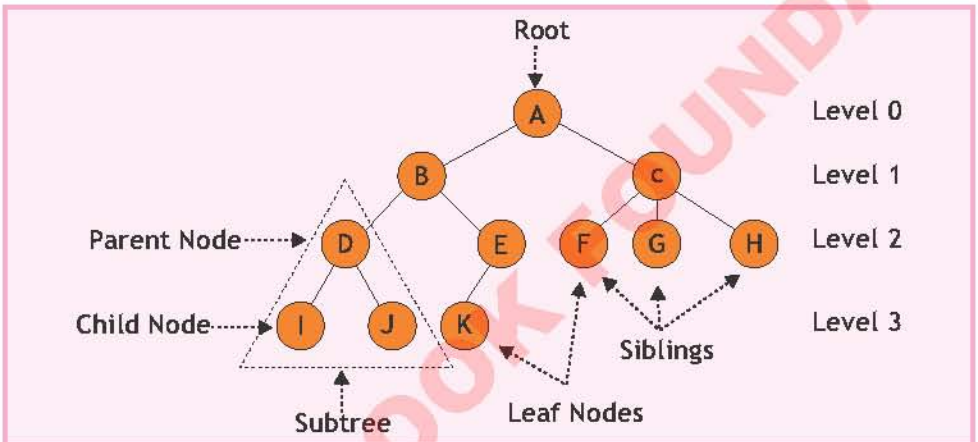


Fig.2.16: Trees

Following are the basic operation that can be performed on trees:

- Insertion: Add a new node at specified location in the tree
- Deletion: Remove existing node
- Search: Find specific node in the tree
- Traversal: Visiting tree in specific order

Tree traversal

A tree traversal is always done in a systematic way and it involves a mechanism to visit all the nodes. There are several methods for tree taversal and each have different purpose. Following is a brief overview of some common tree traversal methods:

- **In-Order Traversal (for Binary Trees) - see Fig.2.17:**

    1. Traverse the left subtree.
    2. Visit the root node.
    3. Traverse the right subtree.

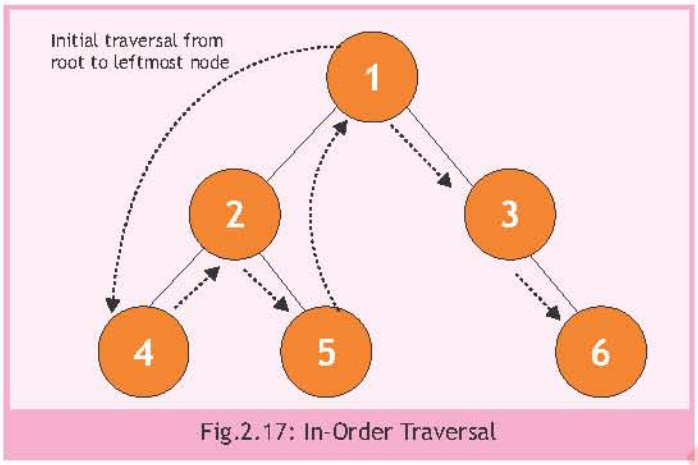


Fig.2.17: In-Order Traversal

So the order of traversal of nodes is 4 -> 2 -> 5 -> 1 -> 3 -> 6.

This traversal is often used with binary search trees (BSTs) because it visits nodes in ascending order.

Pre-Order Traversal - see Fig.2.18:

1. Visit the root node.
2. Traverse the left subtree.
3. Traverse the right subtree.

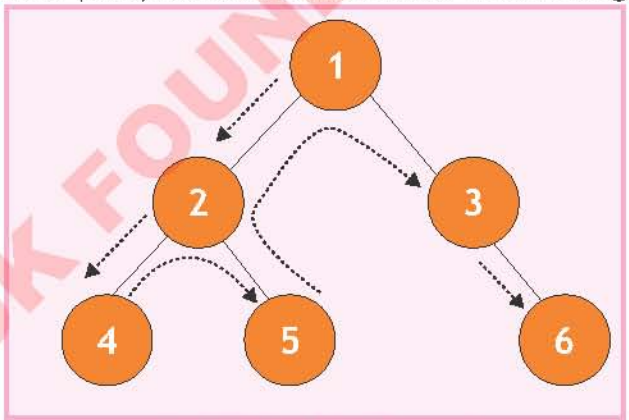


Fig.2.18: Pre-Order Traversal

So the order of traversal of nodes is 1 -> 2 -> 4 -> 5 -> 3 -> 6.

This is useful when we need to work with the root node before its children.

Post-Order Traversal - see Fig.2.19:

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root node.

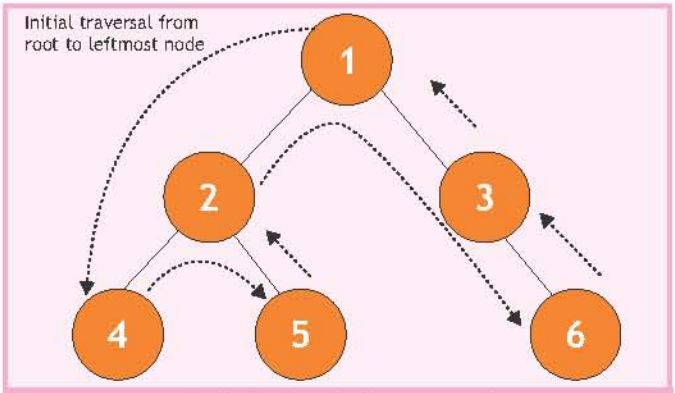


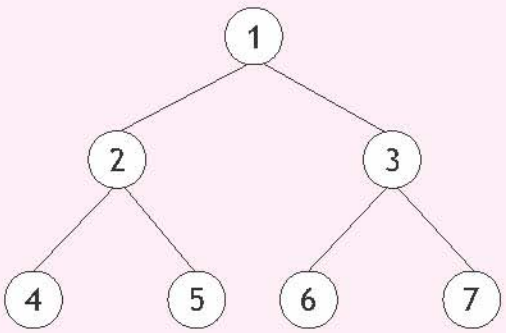
Fig.2.19: Post-Order Traversal

So the order of traversal of nodes is 4 -> 5 -> 2 -> 6 -> 3 -> 1.

This is often used for deleting nodes or freeing memory, as it processes children before their parent.

Example

Fig.2.20 shows examples of Inorder, Preorder and Postorder Traversal



Inorder Traversal

4	2	5	1	6	3	7

Preorder Traversal

1	2	4	5	3	6	7

Postorder Traversal

4	5	2	6	7	3	1

Fig.2.20: Inorder, Preorder and Postorder Traversal

Data Structures Support Computational Thinking:

- **Problem Decomposition:** Different data structures help break down problems into manageable components, making it easier to design and implement algorithms.
- **Efficiency Considerations:** They illustrate trade-offs between different operations (e.g., access time vs. insertion time), helping in choosing the most appropriate structure for a given problem.
- **Algorithmic Thinking:** Understanding these structures aids in developing and analyzing algorithms, improving problem-solving skills by selecting the right tool for each task.

Overall, data structures are not just tools for storing data; they shape the way we think about solving problems and designing algorithms. Mastery of these concepts is crucial for effective computational thinking and developing efficient algorithms.

2.2 Evaluating Computational Solutions

When evaluating computational solutions, it is essential to consider the following criteria:

2.2.1 Correctness

An algorithm is considered correct if it generates the expected output for all valid inputs. To verify correctness, multiple test cases should be used that cover various scenarios, including edge cases.

Example 1: Sorting Algorithm using Bubble Sort Algorithm

Test Case: Given an unsorted array [5, 2, 9, 1, 5, 6], Bubble Sort should correctly sort to [1, 2, 5, 5, 6, 9].

If the algorithm consistently produces the sorted array for various test cases, including edge cases like an empty array or an array with one element, it can be considered correct.

Example 2: Search Algorithm using Binary Search Algorithm

Case: Given a sorted array [1, 2, 3, 4, 5] and a search key 3, Binary Search should return the index 2 (assuming zero-based indexing).

To verify correctness, ensure Binary Search works for various search keys (existing and non-existing) and for arrays of different sizes.

2.2.2 Clarity

An algorithm is clear if its steps are logically structured and easily understandable. For this purpose, multiple things should be taken care of, for example using descriptive names for variables and functions and ensuring that the steps are ordered logically. Similarly, documentation and comments in code can also enhance clarity.

Example: Consider two python implementations of a function that calculates the factorial of a number.

Case 1: Clear Code

python

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

```

Case 2: Less Clear Code

python

```python
def fct(n):
    return 1 if n == 0 else n * fct(n - 1)

```

The first case is easily readable and understandable because the function name (factorial) represents what logic inside the function will be doing. Similarly, the conditional statement (if n == 0) is also very clear that what it means in the code. However, the second case uses a shorter function name and a ternary operator, which, while valid, may be less immediately understandable to someone new to the code.

2.2.3 Efficiency

Efficiency refers to how well an algorithm uses computer resources, such as time and memory. It is

often expressed using Big O notations (e.g., O(n), O(log n), O(n^2)).

Time Complexity:

Time complexity provides a way to evaluate efficiency of a solution. This is done by estimating the time required to solve a problem. While solving a problem, the main outcome of the solution is algorithm. Therefore, this measures the execution time of an algorithm with respect to the size of input. The input size refers to problem size and it represents the size of task and amount of data.

Space Complexity:

Like Time complexity, it evaluates the efficiency of solution but focus on memory used by the solution. This measures the amount of memory an algorithm uses with respect to the size of input.

When evaluating the efficiency of an algorithm, we often focus on the worst-case scenario. For instance, we are searching for a specific element in an array by comparing it with each element at every index, the worst-case situation occurs when the desired element is at the very last index. Big O notation helps us understand the upper bound of an algorithm's running time or space usage relative to the size of the input, providing a way to gauge its efficiency in the most challenging conditions.

Common Big O Notations:

- **O(1) - Constant Time:** The algorithm's running time is constant, regardless of input size. For example, finding a given member in a list involves the same length of time regardless the list has 10 or 10,000 entries.
- **O(logn) - Logarithmic Time:** Algorithm execution time gradually increases with increasing input size. This is comparable to looking for a word in a dictionary, in which each step reduces the search space by half.
- **O(n) - Linear Time:** The algorithm's execution time scales up with input size i.e. if an input size doubles, so does the processing time. One example is a basic loop that goes through each entry in a list.
- **O(n log n) - Linearithmic Time:** This time-based complexity combines linear and logarithmic growth. It is found in effective sorting algorithms such as Merge Sort and Quick Sort, in which the algorithm separates the problem into smaller sections and processes each one separately.
- **O(n^2) - Quadratic Time:** Execution time increases dramatically with input size. This is characteristic of algorithms that use nested loops and compare every member in a list to every other part. When the input size is doubled, the time required to finish the algorithm quadruples.

Example 1: Binary Search Algorithm

Binary search is a search algorithm that locates a target value in a sorted array. It operates by repeatedly partitioning the search interval.

Algorithm Steps:

1. Start by examining the number in the center of the list.
2. Check to see whether this middle number equals the one we are looking for.
3. If it matches, we have located the correct number.
4. If the number we are searching for is on the left part of the list. Repeat the process while ignoring the right side and focusing on the left.
5. If the number we are looking to get is in the list's right half. Ignore the left side and focus on the right portion, continuing the procedure again.
6. Carry on to filter across the list till we identify the number or figure out that it is not on the list.

Time Complexity

The Binary Search Algorithm has Time Complexity of O(log n). The binary search divides the search space in half with each step, this halving process means that the number of elements to search through is reduced exponentially.

When the algorithm starts its initial state is an array of n elements. At each step, we apply the divide and conquer process. This is done by comparing the target value to the middle element of the array or current search range. This comparison helps in eliminating half of the current search range (this half could be either half of the current search range.

- After the first step, we have left with n/2 elements.
- After the second step, we have left with n/4 elements.
- After k steps, we have left with n/2k elements.

The process continues until the number of remaining elements is 1 (or the target is found). The number of steps required is relative to the logarithm of n because we keep halving the problem size each time. Specifically, the number of steps is log2(n).

Space Complexity

The Binary Search Algorithm has the space complexity of O(logn) - Logarithmic Space. To track the current state of recursion, the binary search dependes upon the stack. Each recursive step adds a new frame to the stack. For worst case scenario, the number of the recursions is relative to the number of times the problem size can be halved, which is log2(n). Thus, the space complexity is logarithmic due to the recursive stack usage.

Example 2: Linear Search Algorithm

For Linear search, each element in the array is examined one by one till the target value is found or we reach at the end of the array.

Algorithm Steps:

1. Start at the beginning of the array and examine each element one by one.
2. Compare target value to each element of the array .

3. The search is completed when the target value matches with an element.

4. If the array ends and no match is found, conclude that the target is not in the array.

Time Complexity

The Linear Search Algorithm has the Time Complexity of O(n), because the required number of operations increase linearly with the size of the input. When the algorithm starts its initial state is an array of n elements. We check each element in sequence e.g. First element, second element and so on up to the nth element. For worst case scenario, every single element in the array is examined before finding the target value (or concluding that it's not in the array).

Space Complexity

The Linear Search Algorithm has the Space Complexity of O(1) - Constant Space. The Linear search utilizes fixed amount of extra space regardless of the size of the input. It only needs a few variables to store the current index and the target value, which does not scale with the size of the input. Thus, its space complexity is constant.

2.2.4 Case Studies

Problem 1: Find Greatest Common Divisor (GCD)

Problem Statement: Given two integers, find their greatest common divisor (GCD).

Solution: Euclidean Algorithm

Algorithm:

1. Take two integers a and b where a ≥ b.
2. While b is not zero:
   - Set a to b
   - b to a % b (the remainder of a divided by b).
3. When b becomes zero, a contains the GCD.

Efficiency:

- Time Complexity: O(log min(a,b))
- Space Complexity: O(1) as only a few variables are used.

Clarity:

- The Euclidean Algorithm is elegant and clear, involving a simple iterative process to find the GCD.

Correctness:

- The algorithm correctly finds the GCD based on the mathematical properties of division and remainders.

Problem 2: Count Numbers of Vowels in a String

Problem Statement: For an input string, count the number of vowels in that string.

Solution: Simple Iteration

Algorithm:

1. Initialize a counter to zero.
2. Iterate through each character in the string.
3. For each character, check if it is a vowel (i.e., one of 'a', 'e', 'i', 'o', 'u').
4. If it is a vowel, increment the counter.

After the iteration, the counter will contain the number of vowels.

Efficiency:

Time Complexity: O(n), where n is the length of the string.

Space Complexity: O(1) as only a counter variable is used.

Clarity:

The algorithm is simple and clear, involving basic iteration and conditional checks.

Correctness:

This method correctly counts the number of vowels by evaluating each character.

Summary

- **Decomposition:** Breaking down the complex problem into smaller parts, making it easier to manage and solve. For example, building a website involves separate tasks like designing, coding, backend setup, and testing.
- **Pattern Recognition:** Identifying commonalities or trends within a problem or across similar problems. For instance, in data analysis, recognizing user behavior patterns can help enhance product design.
- **Abstraction:** Focusing on the essential details of a problem while ignoring unnecessary ones, simplifying complex systems. For example, creating a user interface mockup focuses on key interactions without detailing all technical aspects.
- **Algorithm Design:** Creating a clear, step-by-step process to solve a problem efficiently, which is essential in programming. For example, sorting numbers involves using a defined procedure to arrange them from smallest to largest.
- **Correct:** Produces the desired result for all valid inputs.
- **Clear:** Easy to understand and follow.
- **Efficient:** Optimizes resources like memory and time.
- **Sorting Algorithms:** Organize data in a particular order (e.g., Quick Sort, Bubble Sort).
- **Searching Algorithms:** Find specific items in data (e.g., Linear Search, Binary Search).
- **Graph Algorithms:** Work with nodes and edges (e.g., Dijkstra's Algorithm, Depth-First Search).
- **Dynamic Programming:** Solves problems by breaking them into smaller subproblems and storing solutions (e.g., Fibonacci numbers).
- **Greedy Algorithms:** Make the best choice at each step for overall optimization (e.g., Huffman Coding).
- **Backtracking Algorithms:** Explore solutions and backtrack when necessary (e.g., N-Queens Problem, Sudoku Solver).
- **Binary Search:** Efficient for finding a target value in a sorted array, with a time complexity of O(log n).
- **Linear Search:** Examines each element sequentially in an unsorted array, with a time complexity of O(n).
- **Lists and Arrays:** Store elements in sequence, allowing random access via indices, and are crucial for operations involving iteration and loops.
- **Stacks:** Follow the Last-In-First-Out (LIFO) principle, commonly used in algorithms like depth-first search.

Exercise

Select the best answer for the following Multiple-Choice Questions (MCQs).

1. Which of the following best defines a data structure?
   a. A method of communication
   b. A way to store and organize data in memory
   c. A system for internet access
   d. A graphical interface design
2. Which data structure allows dynamic memory allocation and easy insertion/deletion?
   a. Array
   b. Stack
   c. Linked List
   d. Queue
3. In a singly linked list, each node contains:
   a. Data and address of previous node
   b. Only data
   c. Data and address of next node
   d. Index value and data
4. The main advantage of a doubly linked list over a singly linked list is:
   a. Random access
   b. Uses less memory
   c. Allows traversal in both directions
   d. Faster arithmetic operations
5. What type of linked list connects the last node back to the first?
   a. Doubly linked list
   b. Circular linked list
   c. Singly linked list
   d. Linear linked list
6. Which operation removes the top element from a stack?
   a. Enqueue
   b. Dequeue
   c. Pop
   d. Top
7. Which node in a tree does not have any children?
   a. Root node
   b. Leaf node
   c. Sibling node
   d. Parent node
8. Which data structure operates on a Last-In, First-Out (LIFO) principle?
   a. Array
   b. Stack
   c. Queue
   d. Tree
9. Which algorithm is used for sorting data?
   a. Quick Sort
   b. Binary Search
   c. Depth-First Search
   d. Euclidean Algorithm
10. Which algorithm is an example of a Greedy Algorithm?
    a. Bubble Sort
    b. Huffman Coding
    c. Depth-First Search
    d. Binary Search
11. What is the space complexity of the Binary Search algorithm?
    a. O(1)
    b. O(n)
    c. O(log n)
    d. O(n^2)
12. Which data structure allows random access to elements by index?
    a. Stack
    b. Queue
    c. Array
    d. Tree
13. A school wants to automate the process of generating student report cards based on input marks. Which step of computational thinking would most likely involve identifying how to separate marks by subject, calculate averages, and assign grades?
    a. Decomposition
    b. Pattern Recognition
    c. Abstraction
    d. Algorithm Design

14. What is the time complexity of Linear Search?

a. O(1) b. O(log n) c. O(n) d. O(n^2)

15. During an inter-school tech competition, students were asked to develop a solution for managing traffic signals based on traffic density. Which approach best shows their use of computational thinking?

a. They memorized how traffic lights work in real life.

b. They created a flowchart outlining how traffic signal timing changes with vehicle count.

c. They guessed a solution based on their intuition.

d. They programmed a random timer for each signal.

Give short answers to the following Short Response Questions (SRQs).

Problem 1: Finding the Sum of Digits

Problem Statement: Given an integer, calculate the sum of its digits.

Solution: Digit Extraction and Summation

Give answer of the following:

a) Algorithm

b) Efficiency

- Time Complexity
- Space Complexity

c) Clarity

d) Correctness

Problem 2: Finding the Factorial of a Number

Problem Statement: Compute the factorial of a non-negative integer n (i.e., n!).

Solution: Iterative Approach

Give answer of the following:

a) Algorithm

b) Efficiency

- Time Complexity
- Space Complexity

c) Clarity

d) Correctness

Problem 3: Finding the Largest Number in a List

Problem Statement: Given a list of integers, find the largest number in the list.

Solution: Linear Scan

Give answer of the following:

a) Algorithm

b) Efficiency

- Time Complexity
- Space Complexity

c) Clarity
d) Correctness

Problem 4: Checking for Prime Numbers

Problem Statement: Determine if a given integer n is a prime number (i.e., it has no divisors other than 1 and itself).

Solution: Simple Divisibility Test

Give answer of the following:

a) Algorithm

b) Efficiency

- Time Complexity
- Space Complexity

c) Clarity
d) Correctness

Problem 5: Finding the Average of a List

Problem Statement: Calculate the average value of a list of numbers.

Solution: Summation and Division

Give answer of the following:

a) Algorithm

b) Efficiency

- Time Complexity
- Space Complexity

c) Clarity
d) Correctness

6. How can abstraction help a student create a simple timetable app for school?
7. A shopkeeper wants to use a program to quickly find a product in a long list. Which type of algorithm should they use and why?

Give long answers to the following Extended Response Questions (ERQs).

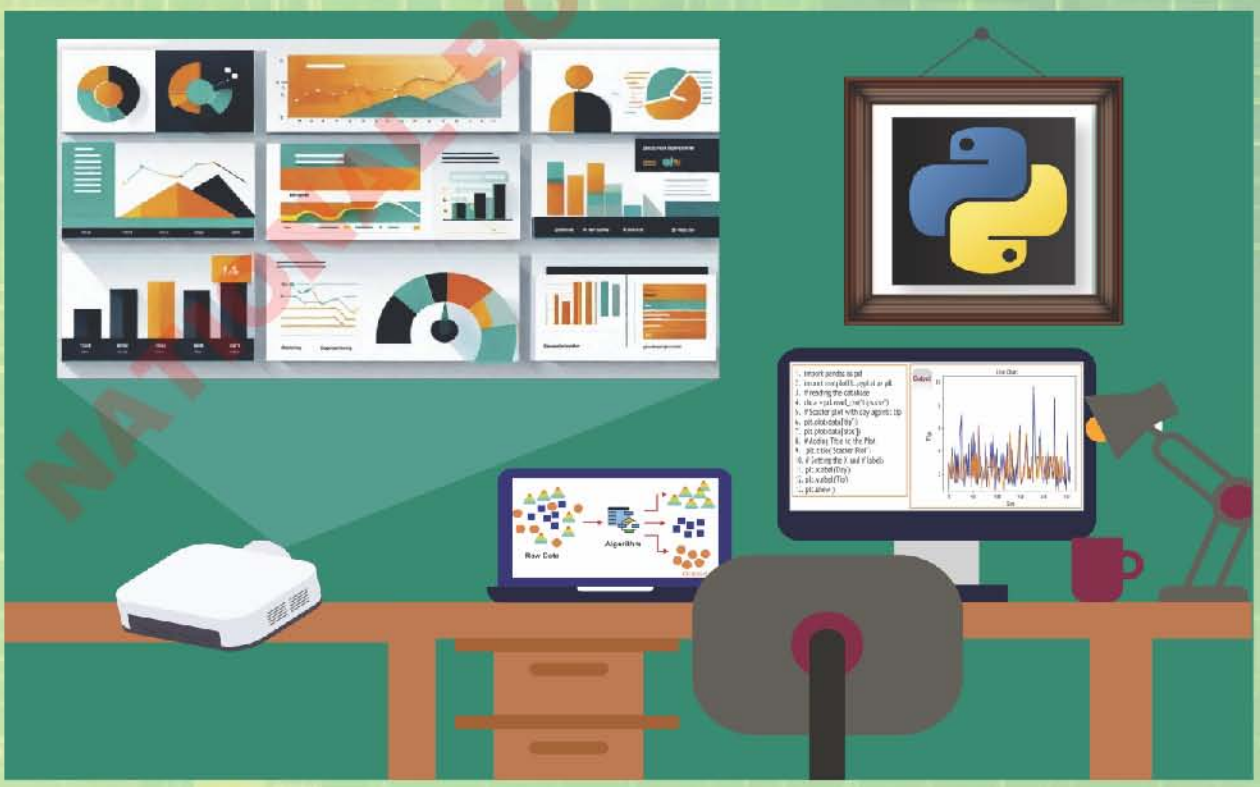
1. Compare and evaluate the efficiency of Bubble Sort and Merge Sort in sorting a list of 1,000 student names. Discuss in terms of time complexity and clarity. Which is more suitable and why?
2. Describe the difference between Stack and Queue. How does their operation affect algorithm design? Provide one real-life example for each data structure.
3. Explain the importance of tree traversal techniques. Compare in-order, pre-order, and post-order traversal with examples and their use cases.

3 Programming Fundamentals

Learning Outcomes

At the end of this unit students will be able to:

- understand and evaluate applications of various programming paradigms.
- use more advanced programming constructs such as data structures (lists etc.), file handling (disk IO to write to storage), and databases in Python.
- implement complex algorithms that use lists etc. in Python
- determine more advanced techniques (unit tests, breakpoints, watches) for testing and debugging their code in Python



3.1 The Programming Paradigm

The term "programming paradigm" describes a vital process for creating and organizing computer program code. It helps in the structure of code, program flow and managing of data for programmers. In other words it can be termed as a particular methodology to use programming language for problem solving. Although programmers can design their own unique structures and functions inside a language, these can't be termed as a paradigm, as it is the programming language that specifies and provides guidelines and functions in terms of what and how to develop. Thus, a programming language paradigm offers a particular way to arrange various functionalities in order to accomplish specified objectives. Additionally, it facilitates the developers with standard tools and implementation techniques that are used commonly for effective problem-solving which helps developers to focus on the specific task in hand and design solution effectively. The most commonly used programming languages include Python, Java, SQL, etc.

3.1.1 Functional Programming

Functional programming is the paradigm of software development which was and is followed by many, especially for small applications where the code is divided into functions based on functionality. The developers in functional programming focus more on 'what' is to be achieved rather than 'how' to address the problem. The solution of the problem in hand is divided into segments and each segment is typically addressed in functions. This way, every step of the solution is mapped in a group of functions as shown in Fig. 3.1.

Fig.3.1: Functional programming

Table.3.1: Pros & Cons of Functional Programming

Pros	Cons
Simpler code.	Not easy for beginners. Needs practice
Code is easier to track for logic and debug.	May involve with overhead especially when handling large data structures.
Functions are easier to test and run.	Complex algorithms may be harder to express, often requiring higher-order functions.
Naturally supports parallel and concurrent programming.	Requires careful handling of data sharing and synchronization for efficient execution.
Emphasizes on what to compute rather than how, hence more concise code.	May be less efficient in performance-critical scenarios.

So, if the flow is clear how to achieve the solution, it becomes easier in functional programming. This way, updating the code for any changes becomes simpler as only that particular functions(s) need to be modified and whole program does not need to be read and understood

3.1.2 Object Oriented Programming

Object Oriented Programming (OOP) is software development paradigm based on the data and objects, which include functions (methods) that define the behavior and data (attributes) to indicate the properties of objects. These are the program's building blocks to regenerate the actual objects and their functionalities, in code. The programs designed using OOP paradigm represents the physical attributes of the objects from real world and therefore developers have a more natural understanding of the problem to be solved by devising the program around objects and their actions.

The main advantage of using OOP paradigm is that the emphasis in problem solving is on objects and not the logic. The program reflects real-world objects like car, fruit, etc. This approach makes it easier to understand the problem, design solution and change the code if needed. OOP is implemented through classes which allow defining initially the object and their functionalities and later physical instances are created on spot when need to be used in the program. This is quite useful for complex systems where developers can map objects and their associations to be realistic and handle them more efficiently. OOP limits direct access to data for safety and reliability of data and programs.

Table.3.2: Pros & Cons of Object Oriented Programming

Pros	Cons
Simplifies complex systems by breaking them into smaller reusable components.	May result in complex designs if not implemented thoughtfully.
Hides implementation details, hence improving code maintainability and security.	At times code is harder to understand and debug due to encapsulation.
Establishes hierarchical relationships between classes.	Rigid design due to inheritance may reduce flexibility.
Objects of different types are treated as if they were of the same type.	Polymorphism increases complexity, making code harder to understand and reason about.
Represents real-world objects and their relationships effectively.	May not be well-suited for certain problems, such as computationally intensive tasks.

Key Concepts

A **class** is an object creation stencil or template. It specifies a collection of methods (functions) and attributes (variables or properties) that its objects will hold. You are creating an instance of a class when you create an object from it. Although each object has a distinct set of data (attribute values), they all use the same class-defined methods. You may effectively express and control complicated data structures and behaviors in your program with this paradigm.

An **object** is an instance of a class that denotes a physical thing, such a person, a car or a bank account. Every object has distinct properties (attributes) and behaviors (methods). For instance, each home (object) constructed from a class that specifies the structure of a house has particular

properties, such as size, color or number of rooms. All objects have the same basic structure as defined by the class, apart its own specific characteristics. Objects in a program interact with one another to imitate real-world situations and improve the organization and modularity of the code.

The 'StationaryItem' class (as shown in table) acts as the stencil for all stationary items in the bookstore. Individual goods developed by this session include pens, pencils and notebooks. Each item has unique values for attributes such as name, price and color. The methods of the class allow you to retrieve or change these attributes which allows managing of information about various stationary items easily.

Class: StationaryItem		
Attributes:	Object 1: A blue XYZ pen.	Object 2: A red ABC pencil.
name (e.g., "Pen", "Pencil", "Notebook")	name: "Pen"	name: "Pencil"
price	price: 200	price: 100
color (e.g., "Blue", "Black", "Red", "Green")	color: "Blue"	color: "Red"
brand (e.g., "XYZ", "ABC", "DEF")	brand: "XYZ"	brand: "ABC"
It can have methods like get_price() to obtain the price of the item, set_price() to adjust the price of the said item, etc.		

Encapsulation is the process of information hiding by combining data (attributes) and the methods that operate on the data into a single entity, i.e. a class. The main concept is to keep together the data and methods that are relevant. Encapsulation does not allow direct access to some components but via some designated methods provided by the object. This way internal working of an object is hidden. Encapsulation protects data from unwanted access or changes and also improves code to be written in modules and easy to read and understand.

For instance, in a bank, the account holder's name, account number and balance are critical information. Rather than changing the balance directly, deposit(amount) method will add the amount to the balance while withdraw(amount) method will subtract the said amount from the balance. Further, get_balance() method will retrieve current balance without changing the balance figures. This way, critical information are encapsulated and cannot be modified directly.

Abstraction simplifies creating complicated scenarios by showing only the necessary information and usefulness. It provides only essential information about the data to others, hiding the background details or implementation. Consider a vehicle for example, where we are mainly concerned with common operations such as start, stop and accelerate. But the underlying mechanism of how engine burns fuel and shifting of gears, etc. are kept hidden.

Inheritance allows you to create new classes (subclasses or derived classes) from existing parent classes. Subclasses inherit the attributes and methods of their parent class, helping in reducing repetition of codes. This idea promotes the efficient extension of existing structures. In general

terms, a 'Vehicle' may have attributes such as speed, color, wheels and operations like accelerate() and brake(). If we specify it further for a 'Car' which is a type of 'Vehicle', so we do not need to define the above stated attributes and operations again because 'Car' inherited these from 'Vehicle' being a derived/sub class and automatically owns these properties and functions of 'Vehicle'. However, 'Car' may have its own additional properties like 'seatingCapacity' and operation like openSunroof().

Polymorphism allows objects from various classes to be considered as instances of a common type, though the implementation of the common type differs among the various classes in their own unique way. This allows writing generic code that can deal with a wide variety of objects. Polymorphism makes it easier to modify object behavior.

For example, the common type 'Shape' can be a triangle, rectangle or circle. Though all three are shapes but have their own distinct properties and operations like draw, resize, calculate-area, etc. Polymorphism is useful in such scenarios where a common operation applies for all but implementation differs based on the type of shape. In context of specific shapes, triangle needs base and height, rectangle uses length and width while circle requires radius to calculate the area. Instead of writing three separate specific calculate-area methods (which may increase the more shapes we take into consideration), via polymorphism we can send a simple instruction to different objects such as 'draw-yourself' and each object will perform the operation in its own prescribed way on the basis of type of shape.

3.2 Programming Constructs in Python

3.2.1 List

List is a basic data structure in Python that holds groups of elements in a sequential manner. This flexibility allows list to store elements of various data types such as boolean, integers, etc. The order that things are added is maintained by lists. Index 0 will have the first item inserted, index 1 will have the second, and so on. This arrangement is essential for retrieving elements based on their location. Different data types can be stored as elements in the same list. List sizes are dynamic, i.e. elements can be added or removed as required.

In Fig.3.2, let's utilize the common functions of adding and removing items from the list. Initially, on line 1 we defined a list namely 'student_grades' with 5 elements in it, to store grades of a student for instance. On line 3, we inserted an additional grade of 90 in the list on index 2, using the **insert()** function. Alternative to insert any other value is by using the **append()** function, which we used in line 5 and inserted another value of 100 which was added at the end of the list. Thereafter, on line 8 the value of 78 from the list was deleted using **remove()** function. Another way to remove any element from the list is using the **pop()** function, such that index number needs to be mentioned. This way, we can remove an element from the list either if the element or its index number is known.

```
1  student_grades = [85, 92, 78, 95, 82]
2
3  student_grades.insert(2, 90)
4  print("Grades after adding a new grade:", student_grades)
5  student_grades.append(100)
6  print("Grades after adding another new grade:", student_grades)
7
8  student_grades.remove(78)
9  print("Grades after removing a grade:", student_grades)
10 lowest_grade = student_grades.pop(2)
11 print("Grades after removing the grade on index 2:", student_grades)
12
13 if 88 in student_grades:
14     student_grades.remove(88)
15 else:
16     print("Grade 88 not found in the list")
17
18 average_grade = sum(student_grades) / len(student_grades)
19 print("Average grade:", {average_grade})
```

Output

```
Grades after adding a new grade: [85, 92, 90, 78, 95, 82]
Grades after adding another new grade: [85, 92, 90, 78, 95, 82, 100]
Grades after removing a grade: [85, 92, 90, 95, 82, 100]
Grades after removing the grade on index 2: [85, 92, 95, 82, 100]
Grade 88 not found in the list
Average grade: {90.8}
```

Fig.3.2: Add and remove items from a list

Activity 1

Extend the program of Fig.3.2 to search for all odd numbers in the list and remove them.

Moreover, there are scenarios, where a particular value is not in the list but you intend to remove it and results in an error. So, a good programming practice is to use conditional (if-else) statement as we have used in lines 13-16. This way, if the number is in the list, it will be deleted otherwise it won't abnormally terminate the program rather a message is printed that the said number is not found, or something like this. Lastly, using the **sum()** and **len()** functions, we calculated the average grade of the student without using any loop.

3.2.2 Tuples

In Python, tuples are represented using '(_,_,_)' and are a special type of list having one feature that they are immutable i.e. a tuple's contents cannot be updated once it has been created. We can view the objects and their arrangements but we cannot add, remove or rearrange them. Tuples are used when a set of values must remain in order and unchanging throughout the program. For example the days of the week (("Monday", "Tuesday", "Wednesday")) or geographical coordinates (latitude and longitude), etc.

```
1 grades = (85, 92, 78, 92, 85, 95)
2 print(grades)
3 print("First grade:", grades[0])
4 print("Third grade:", grades[2])
5 if 92 in grades:
6   print("Student got 92")
7 print(grades.count(92))
8 new_grades = grades + (88, 90)
9 print(new_grades)
```

Output

```
(85, 92, 78, 92, 85, 95)
First grade: 85
Third grade: 78
Student got 92
2
(85, 92, 78, 92, 85, 95, 88, 90)
```

Fig.3.3: Tuples

On line 1 we create a tuple of student grades namely 'grades' and initialize it with some sample grades and print it in the next line which shows duplicate values as well. However, individual grades are accessed by specifying the index (which starts from 0) on line 3 and 4. To check if a specific grade exists in the tuple the `in` operator can also be used with tuples as we used in line 5. Additionally, to find the number of occurrences of a grade `count()` method returns the number of times a specific value appears in the tuple as shown in line 7. Line 8 shows that we can create a new tuple by concatenating existing tuples using the `+` operator.

3.2.3 Sets

Sets in Python are represented using '{_,_,_}' and are collections of unique and unordered elements i.e. a set cannot comprise of duplicate values. A set will only hold one instance even if the same value is added to it more than once. Sets can be used to store a list of unique usernames, product IDs, etc. Using the 'in' operator it is quite easy to check whether a particular element exists in a set or not.

Initially on line 1 a set of student grades is created namely 'grades' and is initialized with some sample grades. Notice that duplicate grades (85 and 92) exist in the initial assigned values. Since sets only store unique values, the duplicates will be automatically removed. Line 2 prints the set 'grades', demonstrating that the duplicates have been removed.

Next, we add a new grade to the set i.e. 88 on line 3 using the `add()` method. Line 4 prints the set again showing the updated values. To check if a specific grade exists in the set, the `in` operator is used. In line 5, if 92 is found in the set, then a message is printed.

```
1 grades = {85, 92, 78, 92, 85, 95}
2 print(grades)
3 grades.add(88)
4 print(grades)
5 if 92 in grades:
6   print("Student has 92 among the grades.")
```

Output

```
{92, 85, 78, 95}
{78, 85, 88, 92, 95}
Student has 92 among the grades.
```

Fig.3.4: Sets

3.2.4 Dictionary

Dictionary Store Key-Value Pairs

Dictionary is an essential data structure in Python that is used to store and arrange data in an organized way using key-value pairs such that each key represents a distinct value, i.e. each key is associated with a particular value. List have index while in dictionary it can be of any data type. For example, in a phone book where the phone number (value) and the person's name (key) are associated with each other. By searching for the matching name, we may quickly get the phone number. Because of this fundamental idea, dictionaries may be used for storing and retrieving information by associating unique keys with values.

In order for keys to be used for searching, they must be unchangeable types of data like strings, integers or tuples. Values may store complicated data structures with flexibility since they can be any data type including texts, integers, lists and even other dictionaries. By design dictionaries avoid duplicate keys such that every key acts as a unique label hence avoiding reshuffling of data. Dictionaries maintain an ordered link among keys and their corresponding values, like keeping track of passwords and usernames, etc.

The Python code in Fig.3.5 shows how to add, create and retrieve entries from a dictionary. Initially a dictionary is initialized with 3 student records where student names being the key and their respective marks are the values associated with the keys. Immediately after that on line 7 and 8 using for loop, we print the dictionary. On line 10, we add another student 'Zaid' in the dictionary, which is added at the end of the list. A similar statement for 'Bilal' is written, since it is already in the dictionary, its value will be updated. On line 13, we removed 'Dilawar' from the dictionary. For confirmation, in the next 4 lines using conditional statements (if-else) we checked for 'Dilawar' in the dictionary. Lastly, final dictionary is printed again. Point to note here is that if we comment line 13, record of 'Dilawar' will not be deleted. The corresponding outputs of the said program and after commenting line 13, both are shown for better understanding.

Next, we modify the code such that an empty dictionary is created and thereafter, name of

students and their respective test numbers from the user are taken in, as shown in Fig.3.6. Lastly, the student name is asked to be searched in the dictionary and corresponding search result is displayed.

```
1  student_grades = {
2      'Amir': 90,
3      'Bilal': 85,
4      'Dilawar': 92
5  }
6
7  for student, grade in student_grades.items():
8      print(f"{student}: {grade}")
9
10 student_grades['Zaid'] = 88
11 student_grades['Bilal'] = 87
12
13 del student_grades['Dilawar']
14
15 if 'Dilawar' in student_grades:
16     print(f"\n Dilawar's grade:", student_grades['Dilawar'], "\n")
17 else:
18     print("Student Dilawar not found \n")
19
20 for student, grade in student_grades.items():
21     print(f"{student}: {grade}")
```

Output

```
Amir: 90
Bilal: 85
Dilawar: 92

 Dilawar's grade: 92

Amir: 90
Bilal: 87
Dilawar: 92
Zaid: 88
```

output without line 13

Confirmation via printing

```
Amir: 90
Bilal: 85
Dilawar: 92
Student Dilawar not found

Amir: 90
Bilal: 87
Zaid: 88
```

Output with line 13

Confirmation via printing

Fig.3.5: Program to create dictionary for storing students' grades

Activity 2

Extend the program of Fig 3.5 in such a way that if the searched item is not found in the dictionary, ask the user if it needs to be added. If user presses 'y' for yes, then the key and respective value (to be taken input) are to be added.

```
student_test_number = {}
while True:
  name = input("Enter student name (or 'q' to quit): ")
  if name.lower() == 'q':
    break
  grade = float(input("Enter Test-Number for {}: ".format(name)))
  student_test_number[name] = grade

name = input("Enter student name to view test-number: ")
if name in student_test_number:
  print(f"{name}'s test-number: {student_test_number[name]}")
else:
  print(f"Student '{name}' not found.")
```

Output

```
Enter student name (or 'q' to quit): Amir
Enter Test-Number for Amir: 90
Enter student name (or 'q' to quit): Bilal
Enter Test-Number for Bilal: 87
Enter student name (or 'q' to quit): Dilawar
Enter Test-Number for Dilawar: 92
Enter student name (or 'q' to quit): Zaid
Enter Test-Number for Zaid: 88
Enter student name (or 'q' to quit): q
Enter student name to view test-number: Bilal
Bilal's test-number: 87.0
```

Fig.3.6: Search an entry in dictionary

Note that, it takes significantly less time to find a value in a dictionary than it does to search a list in Python. This is the reason because in lists we need to go through each item individually in order to locate what you're searching for whereas in dictionaries using the key-value pair, we can locate the searching item quickly.

Finding a Value in a Dictionary v/s List

For the sake of better understanding, a program was written where a list and dictionary were randomly populated with 1 lakh entries. Such that a list of strings of 4-characters was setup while for every 4-character in the dictionary a corresponding key value was randomly arranged. Next 1 random element was searched in both the programming constructs and their times were noted. The difference of time clearly shows the searching in dictionary is faster as compared to the list, as shown in Fig.3.7. Sample of populated data in dictionary and list are shown in first two lines.

```
Search results (Dictionary): {20145: 'dlei'}
Search results (List): ['uiib']
Dictionary search time: 1.000166 msec
List search time: 3.000259 msec
Difference of time: Dictionary Time - List Time: -2.000093 msec
```

Fig.3.7: Difference of time to Search in dictionary v/s list.

3.2.5 Nested List

A nested list is a list of lists such that the inner lists or sub-lists can further contain list as elements. Nested list is quite useful in handling complex data like matrices. So, to explore nested list let's try to create a matrix and print it. First, we define a 3x3 matrix. This is achieved using a nested-list such that every element of the list is itself a list and contains row-wise data. As shown in Fig.3.8, we initialized a 3x3 matrix and thereafter on line 6 just by passing the matrix name as an argument to the print() function, we can get it printed. The first line of the output clearly distinguishes the multiple lists that exist in the list. Next, we print the matrix on line 8 using a for-loop. Since, the list has only 3 members, the loop will execute thrice and in every iteration a row gets printed, hence in the form of a matrix. Lastly, on line 11 we use nested loop, such that the outer loop index is 'row' while the inner loop index is 'col'. This way, for the first row, all columns will be printed and then the same will be processed for second and third rows and the whole matrix gets printed in a single line. To modify the code and get it printed in the form of a matrix, is left as an activity for you.

```
1  matrix = [
2      [1, 2, 3],
3      [4, 5, 6],
4      [7, 8, 9]
5  ]
6  print(matrix)
7
8  for i in matrix:
9      print(i)
10
11 for row in range(len(matrix)):
12     for col in range(len(matrix)):
13         print(matrix[row][col], end=', ')
```

Output

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Fig.3.8: Printing a Matrix using Nested List

In this code, a 3x3 matrix is defined, and its transpose is calculated and printed. In line 2-4, the original matrix is given with numbers. In line 7, the code initializes an empty list to store the transpose. Lines 9-12 iterate through each column of the original matrix and create rows for the transpose. In line 13, each new row is appended to the transpose list. Finally, in lines 16-17, the code prints the transposed matrix. The transpose swaps the rows and columns of the original matrix.

Thereafter, we extend the program to calculate the transpose of the matrix and print it. For this purpose, on line 10 of Fig.3.9 we initialized another matrix. On line 16, just like in the last example, we placed a nested loop as we know how to access the nested list contents one-by-one. But, while assigning these values to the transpose matrix the rows should be converted into columns and vice versa. Therefore, note that the indices of the transpose matrix are reversed, which will result in the element at row 1 and column 2 to be placed at column 1 and row 2, which was the desired outcome. In lines 17-19, the output reflects successful creation of transpose matrix of the given matrix.

Activity 3

Tailor the code of Fig 3.8, such that for all odd values of the matrix the sign should be inverted while for all even values multiply it by 2.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print('Matrix is: ', matrix)
for i in matrix:
    print(i)

trans = [
    [0,0,0],
    [0,0,0],
    [0,0,0]
    ]

for row in range(len(matrix)):
    for col in range(len(matrix)):
        trans[col][row] = matrix[row][col]

print('Transpose of the Matrix is: ',trans)
for j in trans:
    print(j)
```

Be careful about the indices as it may return the original matrix and not the transpose of the matrix.

```
Matrix is:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Transpose of the Matrix is:  [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
```

Output

```
Matrix is:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Transpose of the Matrix is:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

Fig.3.9: Program to create Transpose of a matrix

Activity 4

A matrix 'X' is symmetric matrix, if X = (Transpose of X). Write code to determine whether a given matrix is symmetric matrix or not.

3.2.6 List as a Value in Dictionary

A dictionary comprises of a key-value pair and for multiple values against a key can be stored in the dictionary in the form of a list. For example, every record of a student can be stored in the dictionary such that the student's name will act as the key while the registration number and marks will be the values. To store multiple values, a list can be used.

```
students = {
    "Sara": ["345", 85, 90, 78],
    "Jamal": ["678", 75, 88, 92],
    "Zaid": ["912", 80, 79, 95]
}

sara_reg_num = students["Sara"][0]
print("Sara's Reg. No.: ",sara_reg_num)
jamal_marks = students["Jamal"][1]
print("Jamal's Marks for first subject are: ",jamal_marks)

students["Azra"] = ["234", 78, 85, 90]
print(students, "\n")

student_name = "Azra"
subject_index = 2
new_mark = 100
if student_name in students:
      students[student_name][subject_index] = new_mark
      print(f"*** Marks updated for {student_name} at index {subject_index} \n")
else:
    print("Student not found \n")

for name, details in students.items():
    print(f"Name: {name}")
    print(f"Registration Number: ", {details[0]})
    print(f"Marks: [", end='')
    print(details[1],',',details[2],',',details[3], "]\n")
```

Output

```
Sara's Reg. No.:  345
Jamal's Marks for first subject are:  75
{'Sara': ['345', 85, 90, 78], 'Jamal': ['678', 75, 88, 92],
'Zaid': ['912', 80, 79, 95], 'Azra': ['234', 78, 85, 90]}

*** Marks updated for Azra at index 2

Name: Sara
Registration Number:  {'345'}
Marks: [85 , 90 , 78 ]

Name: Jamal
Registration Number:  {'678'}
Marks: [75 , 88 , 92 ]

Name: Zaid
Registration Number:  {'912'}
Marks: [80 , 79 , 95 ]

Name: Azra
Registration Number:  {'234'}
Marks: [78 , 100 , 90 ]
```

Fig.3.10: A dictionary with a list.

Fig 3.10 shows the code how to create a list in a dictionary and initialize it, such that the list comprises of all the data related to student in every row. Line 7 shows how to extract the registration number from the list while line 9 depicts how to select marks from the list. Furthermore, on line 12 we insert a new record in the dictionary and print the whole dictionary in the next line which shows all the records in the dictionary. To understand how to access the list elements in the dictionary, let's assume that the second subject marks of the record that we just inserted needs to be updated. In line 19, based on the key being the student's name, we update the marks of the second subject. Lastly, on line 24, using a for loop in terms of the key-value pair we print all the items of the dictionary.

Activity 5

A primary school has subject teachers such that a teacher teaches only 2 subjects to a class. Assuming, each class studies 8 subjects in total. Write a program using list as a value in dictionary, so that 'class' becomes the key and respective 'teachers' are stored as values.

3.2.7 Files in Python

Importance of Disk I/O

Disk input/output (I/O) is vital for efficient data handling and optimal system performance in computers. Disk I/O bridges between the CPU and HDD for information retrieval, such that relevant data is passed to the CPU for processing and later is stored back on permanent storage devices. Almost all computer operations are supported by this continuous data interchange, from launching of the programs, opening files and save changes made by user.

In Python, files are mainly of two types:

Text File:

In a text file, data is stored through characters like .txt, .csv, etc. Point to note here is that a special marker called 'End Of Line' (EOL) generally '\n' is used to highlight completion of a line. This way, Python knows that a new line is starting after EOL marker.

Binary File:

In a binary file data is stored in combination of 0s and 1s, i.e. binary data. Binary files are not meant for humans but rather for other programs to read information. Images (having extension .jpg, .png), audio (.mp3), video (.mp4) and executable programs (.exe) are all examples of binary files.

File Handling Methods and Operations

Data in files from storage devices is brought up in RAM, where functions like creation of new file, reading and updating of already existing file, closing the file, etc. can be performed on the files. Python has file handling methods that let you work with files and you may interact with files in an organized manner and create, read, write and modify their contents programmatically, using:

- The **open()** method is used to access a file such that it takes in 2 arguments, one is file name

(with complete path) while the other argument is the desired mode of operation (reading, writing, appending, etc.).

- After the file is opened in read mode, its contents can be extracted by using **read()** function and the complete contents of the file is obtained as a string.
- Write mode allows you to add new material or edit already-existing information to the file using **write()** function.
- You may add new data at the end of an already-existing file without erasing its contents, by using the append mode using the **add()** technique.
- The **close()** function must be used to close a file once you have completed working with it. This guarantees data integrity and releases system memory that was in use by the file.

Example: Program To Create An Empty File And Write Content In It.

For the code in Fig.3.11, in the first line a file named 'abc.txt' is created (if it already does not exist) with **open()** function and returns a file handler. Thereafter, every operation uses this file handler to operate on the file. For example, in the very next line a sentence is written in the file, using **write()** function using 'new_file' file handler. In line 4, a message for the user is displayed such that the contents of the file can be checked by opening it, as shown in the output. Lastly, for every file that is opened, it needs to be explicitly closed before the program terminates, and that is generally the last line of the program.

```
1 new_file = open("abc.txt", "w")
2 new_file.write("The quick brown fox jumps over the lazy dog !!!")
3
4 print("File created and written to successfully!")
5 new_file.close()
```

Output

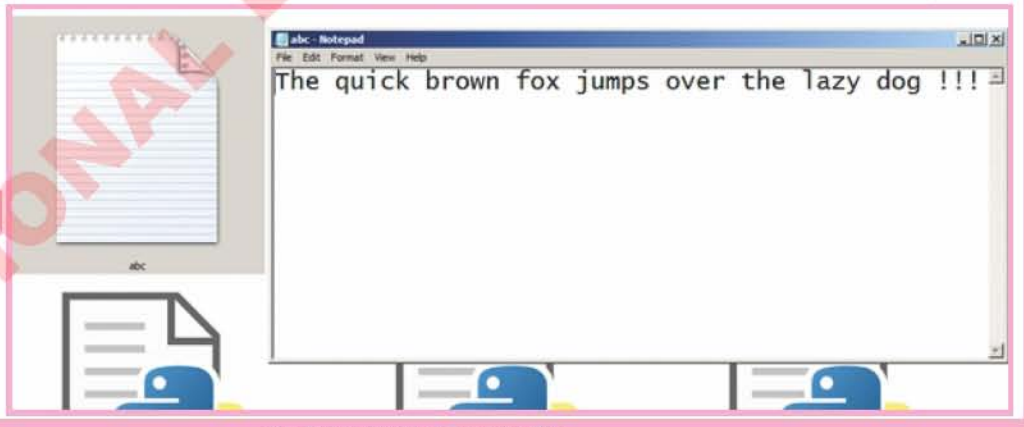


Fig.3.11: Writing Text in file

Alternatively, as shown in Fig.3.12, we can use the 'with' keyword. The benefit of using 'with' keyword for opening file is that the file does not need to be closed explicitly and specially in case of an error, it will automatically be closed. For the sake of clarity, as we are writing the same sentence as of Fig 3.11, but without '!!!' at the end of the sentence, just to distinguish that new text has been updated. When executed, the updated output of the same file is shown in the output.

```
1 with open("abc.txt", "w") as file:
2   file.write("The quick brown fox jumps over the lazy dog")
3
4 print("File created and written to successfully!")
```

Output

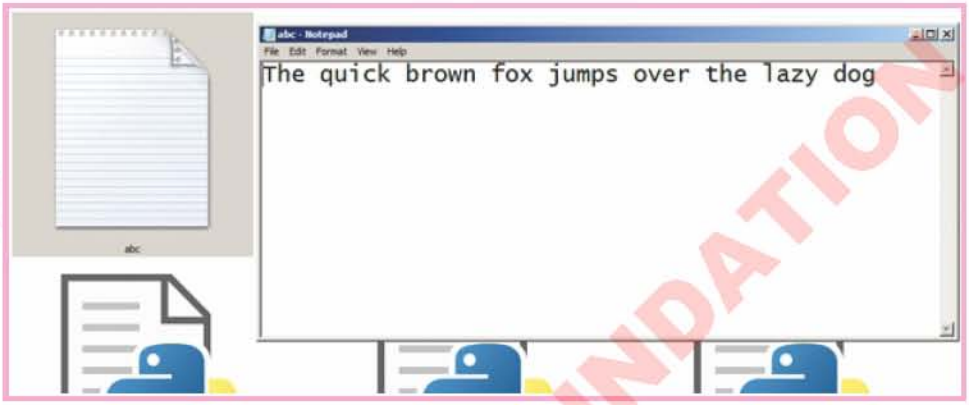


Fig.3.12: Opening file to write with 'with' keyword

Read Existing Files

Next, let's read the contents of an existing file, like the file 'abc.txt' that we just created and saved. The file "abc.txt" is opened in read mode ("r") in the first line of Fig.3.13. An error message will be displayed if the file is missing. As can be seen in line 2, using a for-loop the whole file is read character-by-character and printed on the screen without adding a new line after each character. Lastly, a confirmation message is displayed on line 4 that the file is successfully read, as shown in the output.

```
1 with open("abc.txt", "r") as file:
2   for char in file.read():
3     print(char, end="")
4 print("\nFile read character by character successfully!")
```

Output

```
The quick brown fox jumps over the lazy dog
File read character by character successfully!
```

Fig.3.13: Program to read file

Errors may arise while dealing with files and may result in abnormal termination of the program. To avoid this, 'Try-except' blocks are used to handle possible problems while accessing and working with files. Typical issues include trying to open a nonexistent file, reading from a file or running into issues when writing data. Alternative actions can be implemented by enclosing file operations in a try block and addressing particular exceptions such as 'FileNotFoundError', 'PermissionError' or 'IOError' in the except block as shown in Fig 3.14.

```
try:
    with open("abc.txt", "r") as file:
        for char in file.read():
            print(char, end="")
    print("\nFile read successfully")
except FileNotFoundError:
    print("Error: The file does not exist.")
except IOError:
    print("Error: An issue occurred while reading the file.")
```

Fig.3.14: try-except block

Activity 6

Write a program to open the file 'abc.txt' in read-mode. Create another file 'xyz.txt' and write the contents of abc.txt by reading character by character.

3.2.8 GUI in Python

Graphical User Iinterface (GUI) improves the usability and accessibility of applications as compared to Command-Line Interface (CLI) systems which require users to write commands. In GUI through components like menus, text fields, buttons etc., users may interact with the program more smoothly. Python provides a library for developing GUIs and games namely 'Tkinter to handle common tasks like user input i.e. the steps to be taken when a button is clicked. Tkinter is a built-in Python library used to create GUIs. It allows programmers to create interactive application components, like desktop Applications including word processors, media players and basic utilities where all their interfaces can be made with Tkinter. Tkinter also can be used for creating 2D games with simple graphics. Game elements and windows can be created and user inputs like keyboard and mouse clicks can be controlled using functions provided by Tkinter. Options provided by Tkinter, include:

- The basic components of the interface are called widgets which include checkboxes, buttons, labels and text boxes. We can organize and modify widgets to make the layout as desired.
- Tkinter also is used for event handling like mouse clicks, key presses and window resizing. We can use code to create responses to these events for interactive and dynamic applications.

The arrangement of GUI components and the underlying logic are generally kept separate in the code.

Simple Number Adder with GUI

Before moving on with the code, it is quite helpful to understand the steps involved in creating a GUI using Tkinter library. Let us create a simple Adder application which takes in two numbers as input and when the 'Add' button is pressed, the result gets displayed. Major steps are:

1. First of all, import the tkinter library in Python code and create the main application window which will hold all

```
import tkinter as tk
```

components and is created using tkinter.Tk().

2.Next we define a function to add numbers on the screen.

```
import tkinter as tk

def add_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result_label.config(text=f"Sum: {int(num1 + num2)}")
    except ValueError:
        result_label.config(text="Invalid input. Please enter numbers."
```

We take in the user input into the first and second boxes using get() function. Next we add the numbers and update the result label to display the sum. If the user types something that's not a number like letters, etc. an error message is displayed.

3.Thereafter we defined another function to clear fields.

```
def clear_fields():
    entry1.delete(0, tk.END)
    entry2.delete(0, tk.END)
    result_label.config(text="Result will be displayed here.")
```

To clear the contents of the first text entry box, we use the delete command, starting at the 'position 0' and going all the way to the end. Next, we do the same thing for the second text entry box, clearing its text as well. Finally, we reset the result label to its default message by configuring its text to "...".

4.Create main window along with the title of the program should be the next step using:

```
root = tk.Tk()
root.title("Simple Adder")
```

5.As a next step we setup input fields:

```
entry1 = tk.Entry(root, width=10)
entry1.grid(row=0, column=0, padx=5, pady=5)

entry2 = tk.Entry(root, width=10)
entry2.grid(row=0, column=1, padx=5, pady=5)
```

We create a little text box for the user to type in. The 'tk.Entry()' creates the box itself and we have set it to be 10 characters wide. Then using 'grid()' we put it in the top left corner of our window (row 0 and column 0 to be specific). We also added a little bit of space around it i.e. 5 pixels on all sides using padx and pady. Similarly for the second text box next to the first one, in the same row (row 0) but one column over (column 1).

6.Subsequently buttons are added in the screen.

```
add_button = tk.Button(root, text="Add", command=add_numbers)
add_button.grid(row=1, column=0, padx=5, pady=5)

clear_button = tk.Button(root, text="Clear", command=clear_fields)
clear_button.grid(row=1, column=1, padx=5, pady=5)
```

We create an "Add" button which when clicked will call "add_numbers()" function. This button is placed in the second row and the first column of our window with a little bit of space around it as in the last step. We also have a "Clear" button that when clicked will call "clear_fields()" function.

7.To display result, a label needs to be added. So, a label is created to show the results. Initially, it says "Result will be displayed here." This label is positioned on the grid in row 2, stretching across both columns using the 'columnspan' attribute. We also add a little bit of padding around the label to make it look nicer.

```
result_label = tk.Label(root, text="Result will be displayed here.")
result_label.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
```

8.Lastly we run the GUI in a continuous manner. Without it, the window will just flash and disappear. This loop ensures that the windows stays open and interactive.

```
root.mainloop()
```

The complete code will look like as shown in Fig 3.15.

```
import tkinter as tk

def add_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result_label.config(text=f"Sum: {int(num1 + num2)}")
    except ValueError:
        result_label.config(text="Invalid input. Please enter numbers.")

def clear_fields():
    entry1.delete(0, tk.END)
    entry2.delete(0, tk.END)
    result_label.config(text="Result will be displayed here.")

root = tk.Tk()
root.title("Simple Adder")

entry1 = tk.Entry(root, width=10)
entry1.grid(row=0, column=0, padx=5, pady=5)
```

```
21
22    entry2 = tk.Entry(root, width=10)
23    entry2.grid(row=0, column=1, padx=5, pady=5)
24
25    add_button = tk.Button(root, text="Add", command=add_numbers)
26    add_button.grid(row=1, column=0, padx=5, pady=5)
27
28    clear_button = tk.Button(root, text="Clear", command=clear_fields)
29    clear_button.grid(row=1, column=1, padx=5, pady=5)
30
31    result_label = tk.Label(root, text="Result will be displayed here.")
32    result_label.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
33
34    root.mainloop()
```

Output

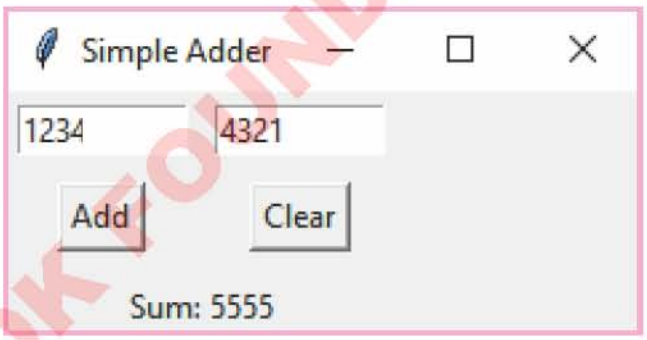


Fig.3.15 : tikinter

Developing Game using tkinter

In terms of GUI application, even games can be developed using tkinter. Let us create a simple game of correctly identification of color such that the text color needs to be identfied and written with correct spellings. The twist however is that the text show name of a different color to confuse the player. Only right color input with correct spellings will be counted.

Steps to create the color game:

1. Create a new Python file and import necessary libraries:
    1. import tkinter as tk
    2. import random

2. Set a list of colors:

colors = ["red", "blue", "green", "yellow", "pink", "black", "orange", "purple", "brown"]

3. Sketch game logic

- Initialize variables like score = 0.
- Outline 'new_round()' function:
    - Select a random color from the colors list for 'color_to_guess'.
    - Select another random color for the text's display color 'random_color'.

- Align the label with the 'color_to_guess' text and 'random_color'.
- Clear player's input from the entry field, if any.

- For 'check_answer()' function:
  - Acquire the user_input from the entry field and check if it matches the random_color.
  - If correct, increment the score and update the score_label.
  - If incorrect, display a message showing incorrect answer.
  - Call 'new_round()' to start the next round.

4. GUI

- Create the main window:

1. window = tk.Tk()
2. window.title("Color Game")

- GUI elements that we need to place:

a) Add label 'Display the word to be guessed'.

```
27    label = tk.Label(window, font=("Arial", 24))
28    label.pack(pady=20)
```

b) Input field for entering answer.

```
30    entry = tk.Entry(window, font=("Arial", 18))
31    entry.pack()
```

c) Add a button to check player's answer when pressed.

```
33    check_button = tk.Button(window, text="Check", command=check_answer)
34    check_button.pack(pady=10)
```

d) Displays the player's current score.

```
36    score_label = tk.Label(window, text="Score: 0")
37    score_label.pack()
```

5. Starting the game

- Initially 'new_round()' function will be called to start the first game round.
- Call 'window.mainloop()' to run the game.

The complete code is shown in Fig 3.16.

```
1  import tkinter as tk
2  import random
3
4  window = tk.Tk()
5  window.title("Color Game")
6  score = 0
7  colors = ["red", "blue", "green", "yellow", "pink", "black",
   "orange", "purple", "brown"]
8
9  def new_round():
10     global score
11     global color_to_guess, random_color
12     color_to_guess = random.choice(colors)
13     random_color = random.choice(colors)
14     label.config(text=color_to_guess, fg=random_color)
15     entry.delete(0, tk.END)
16
17 def check_answer():
18     global score
19     user_input = entry.get().lower()
20     if user_input == random_color:
21         score += 1
22         score_label.config(text="Score: " + str(score))
23     else:
24         score_label.config(text="Score: " + str(score) + "
       (Incorrect)")
25     new_round()
26
27 label = tk.Label(window, font=("Arial", 24))
28 label.pack(pady=20)
29
30 entry = tk.Entry(window, font=("Arial", 18))
31 entry.pack()
32
33 check_button = tk.Button(window, text="Check",
   command=check_answer)
34 check_button.pack(pady=10)
```

① ② ③ ④ ⑤

```
35
36    score_label = tk.Label(window, text="Score: 0")
37    score_label.pack()
38
39    new_round()
40    window.mainloop()
```

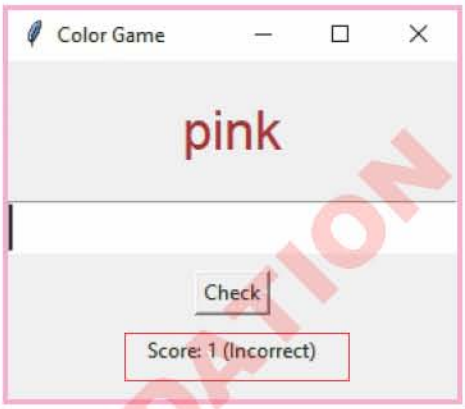


Fig.3.16 : tikinter

3.2.9 Databases

A set of data organized in a systematic manner, usually in a computer system, is called a database. A database stores, retrieves and handles large number of records for multiple users. Databases allow users to locate specific information quickly and accurately. The central source of data for every software program is a database. For instance, when a user logs in, shops online or makes a transaction, a database system saves the relevant data so that it may be retrieved later, i.e. what time the user logged in, which items the user purchased or the transaction to which account took place, etc.

Databases remove redundancies and inconsistencies that are possible in manual record keeping techniques by organizing data and ensuring its reliability. Databases also provide concurrent access and changes which allows several users to work on the same data-set at the same time. They also include security measures to safeguard sensitive information from unwanted access. Databases provide effective information retrieval and help in data analysis.

Types of Databases

- Most databases are relational. They organize data into rows and columns of a table such that each table represents an individual entity (for example customers or items).
- NoSQL databases provide flexibility and can manage vast amounts of unstructured or semi-structured data. Unlike relational databases, they do not have a fixed table structure.

Key Concepts involved in databases are:

- **Table:**

A table is the fundamental structure of a relational database and is made up of rows and columns. Each row denotes a record, while each column represents a field.

- **Record:**

A record is a single table item with information about a specific entity. For example a client record could include information like his or her name, address and telephone number.

Field:

A field is a piece of data within a record that defines the information placed in a certain column. For example, a Name field would store text data, whereas an Age field would store numerical data.

Data Integrity

Data integrity refers to the accuracy and consistency of data within a database. It ensures that the data is accurate, dependable and appropriate for usage.

Primary & Secondary Keys

Keys are essential in creating database to uniquely identify records and create links among tables. The most common types of keys used in normalization are:

Primary Key:

A primary key is a minimal set-of-attributes that is exclusively able to identify a record. It can be a single attribute or a combination of attributes.

Candidate Key:

A candidate key is a single or a set-of-attributes that allow identifying of a record uniquely. Though there may be several candidate keys, only one is chosen to be the primary key. The rest are called secondary keys.

Foreign Key:

A foreign key is a field in one table that creates a relationship with another table by referring to its primary key.

Composite Key:

A composite key is a combination of two or more attributes that distinctively identify a row in a database table.

Data Normalization

A key component of relational database management is data normalization, which focuses on minimizing duplicate data. To ensure effective and consistent data storage and speed-up database access, the contents are divided into smaller interlinked tables. Data redundancy or duplicate storage of the same information is a common problem in databases which causes problems in maintaining consistency and increase chances of inaccuracies during updating of data. By dividing the data into more manageable and inter-linked tables, normalization solves these problems. Every table represents a separate entity with keys defining the relationships among the entities.

- Without normalization databases frequently show inconsistencies and abnormalities during updates or deletions. For example, a database containing customer information with several addresses would require changes in many locations for every address change, increasing the chances of mistakes. Normalized data addresses this by creating various tables for customers and addresses, which are connected by a key like customer-ID.

Normalization steps for a database are:

- **First Normal Form (1NF):**

In 1NF tables have atomic values along with repeating groupings / multi-valued cells are removed.

- **Second Normal Form (2NF):**

2NF ensures that every non-key attribute is dependent on the whole Primary key.

- **Third Normal Form (3NF):**

In 3NF transitive dependencies, in which one non-key attribute relies upon another non-key attribute, are eliminated. For example, while developing a small database for a laptop shop the initial information may look like:

For a laptop shop if we want to create a database, the steps will be as follows:

Step 1:

Classify entities along with the attributes, which are:

- Customer: C_ID, C_Name, C_Address, C_Phone
- Laptop: L_ID, L_Name, L_Price, L_Brand
- Order: OrderID, OrderDate, C_ID, L_ID, Quantity

Step 2:

Let us initially create a single table to represent all the data:

OrderID	OrderDate	C_ID	C_Name	C_Address	C_Phone	L_ID	L_Name	L_Price	L_Brand	Quantity
1	20-11-2024	C1	Qamar	House:123	123-4567	L1	Dell XPS 13	100000	Dell	1
2	21-11-2024	C2	Mahtab	House: 456	987-6543	L2	MacBook Air	90000	Apple	2
3	22-11-2024	C3	Qamar	House:123	123-4567	L3	HP Spectre x360	120000	HP	1
4	23-11-2024	C4	Mahtab	House: 456	987-6543	L4	Dell XPS 13	100000	Dell	1
5	24-11-2024	C5	Chaand	House: 789	543-2109	L5	MacBook Air	90000	Apple	2

Step 3:

To normalize to 1NF we need to find repeating groups, which are:

Customer information that is repeated for each order.

Laptop information that is repeated for each order.

So, let us create separate tables as follows:

- Customer (C_ID, C_Name, C_Address, C_Phone)
- Laptop (L_ID, L_Name, L_Price, L_Brand)
- Order (OrderID, OrderDate, C_ID, L_ID, Quantity)

Hence, 1NF Tables are:

Customer

C_ID	C_Name	C_Address	C_Phone
C1	Qamar	House:123	123-4567
C2	Mahtab	House: 456	987-6543
C3	Chaand	House: 789	543-2109

Laptop

L_ID	C_Name	L_Price	L_Brand
L1	Dell XPS 13	100000	Dell
L2	MacBook Air	90000	Apple
L3	HP Spectre x360	120000	HP

Order

OrderID	OrderDate	C_ID	L_ID	Quantity
1	20-11-2024	C1	L1	1
2	21-11-2024	C2	L2	2
3	22-11-2024	C1	L3	1
4	23-11-2024	C2	L1	1
5	24-11-2024	C3	L2	2

Step 4:

To normalize to 2NF we need to detect partial dependencies, which come out to be:

- OrderID depends on OrderDate, but not on C_ID or L_ID.

As a result, we need to create a new table:

- OrderDetails (OrderID, OrderDate)

As a result, 2NF Tables are:

OrderDetails

OrderID	OrderDate
1	20-11-2024
2	21-11-2024
3	22-11-2024
4	23-11-2024
5	24-11-2024

Order

OrderID	C_ID	L_ID	Quantity
1	C1	L1	1
2	C2	L2	2
3	C1	L3	1
4	C2	L1	1
5	C3	L2	2

Step 5:

Lastly, to normalize to 3NF we ought to identify transitive dependencies, i.e.:

- L_ID in the Order table depends on L_Name, L_Price, and L_Brand, which are attributes of the Laptop table.

To remove transitive dependencies:

- The Order table now only references the L_ID.

Thus, the 3NF Tables are:

OrderDetails

OrderID	OrderDate
1	20-11-2024
2	21-11-2024
3	22-11-2024
4	23-11-2024
5	24-11-2024

Customer

OrderID	C_ID	L_ID	Quantity
1	C1	L1	1
2	C2	L2	2
3	C1	L3	1
4	C2	L1	1
5	C3	L2	2

Order

C_ID	C_Name	C_Address	C_Phone
C1	Qamar	House:123	123-4567
C2	Mahtab	House:456	987-6543
C3	Chaand	House:789	543-2109

Laptop

L_ID	L_Name	L_Price	L_Brand
L1	Dell XPS 13	100000	Dell
L2	MacBook Air	90000	Apple
L3	HP Spectre x360	120000	HP

Database Tools

Various database tools are available for data management, with Microsoft Access and SQLite being two popular options. MS Access, part of the MS Office suite, is simple to install, easy to use, and offers a user-friendly drag-and-drop interface. It is commonly used by individuals and small organizations for tasks like inventory, project management, and event planning. MS Access also integrates well with other MS Office programs like Excel and Project.SQLite is a lightweight database system that stores data in a single file. It is ideal for applications needing local storage and is widely used in mobile apps and scenarios that do not require a large database server. SQLite is simple, efficient, and easy to use.

Advantages of SQLite:

- SQLite occupies small space making it suitable for applications with limited resources.
- It is easily integrated into applications, hence becomes easier for use and deployment.
- SQLite has a serverless architecture which eliminates unnecessary dependencies.
- The complete database is saved in a single file.

3.2.10 Create a Database

1. Installation of SQLite

To download and install SQLite on your Windows system, first go to the official SQLite website (https://sqlitebrowser.org/dl/). Look for a "Download" button and select the right download for your system. SQLite normally comes with already compiled files. Select the version that matches your Windows operating system and click the download link. The browser will prompt you to save the file. Choose a suitable location on your computer, such as the 'Downloads' folder. After the download is finished, you will notice that the file is a compressed (e.g., a ZIP) file. To access the SQLite files, right-click the downloaded file and select "Extract All". This will create a new folder containing the SQLite files which are ready to be used.

To check whether the SQLite was successfully installed, launch the 'Command Prompt' by typing "cmd" into the Windows search bar and click it. After opening the Command Prompt, use the 'cd' command to navigate to the folder where you extracted the SQLite files (for example, 'cd C:\Downloads\sqlite\'). Next type'sqlite3' and click Enter. If the installation was successful, SQLite's command prompt will appear showing that it is ready for use.

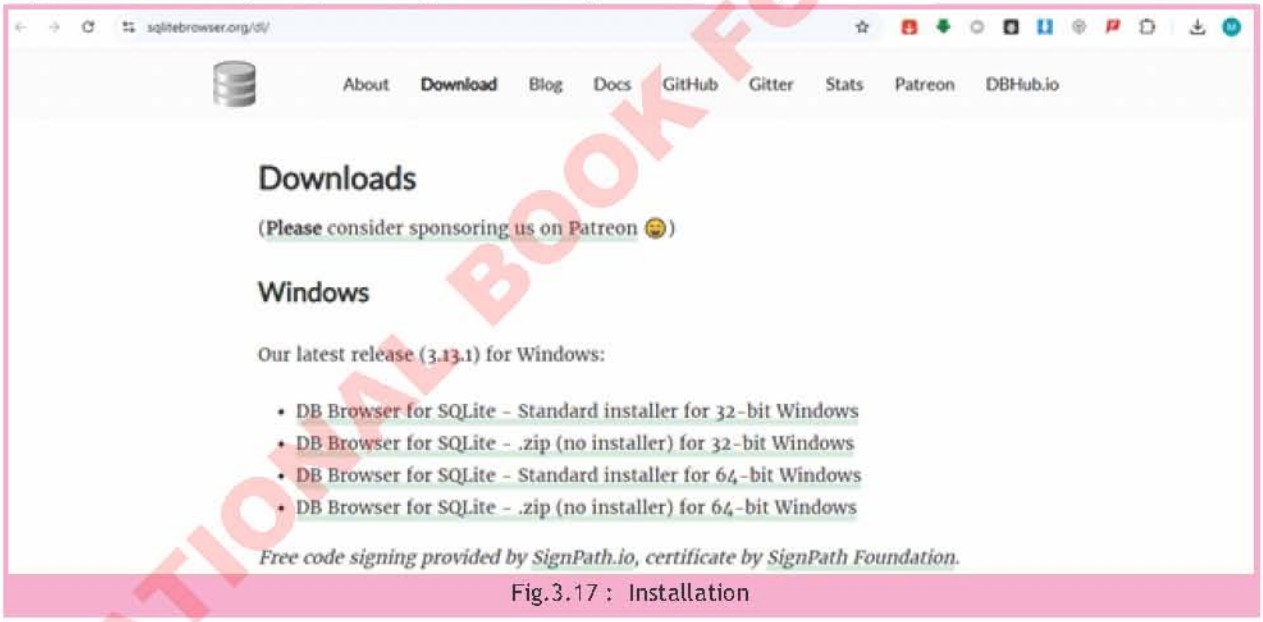


Fig.3.17 : Installation

2. Database Creation

To create a new SQLite database, launch the SQLite Command-Line Interface (CLI). To access the interactive SQLite shell:

- Open your terminal or command prompt and type'sqlite3' and press Enter.
- Once inside the SQLite shell, use the command '.open NBF.db' to create a new database namely 'NBF'.

This command creates a new file, 'NBF.db' in the current directory to store your database. And we have successfully created a new SQLite database.

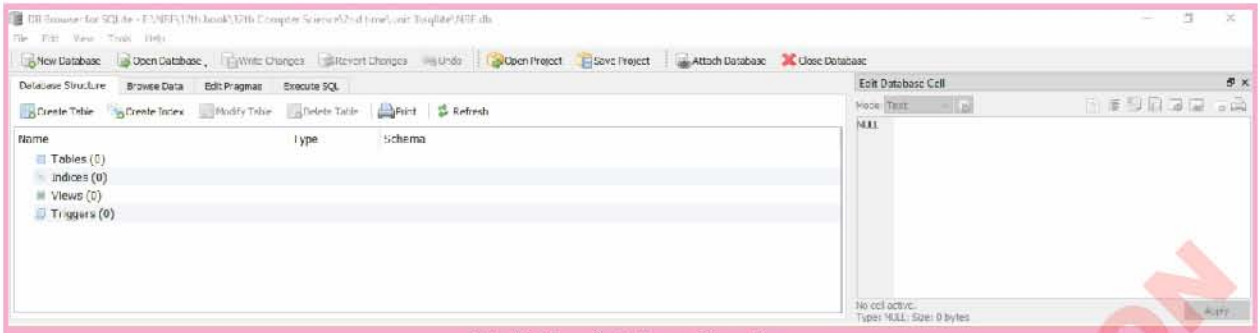


Fig.3.18 : Database Creation

3. Creating a Table

As a next step, use the 'CREATE TABLE' command to construct a table in the database. This command takes in the table's name, column names and data types. For example, we can create a table called 'Laptop' with four columns: 'L_ID' (an integer primary key), 'L_Name' (text), 'L_Price' (integer) and 'L_Brand' (text). So far only the schema is created and it does not contain any record.

```
1  CREATE TABLE Laptop (
2      L_ID TEXT PRIMARY KEY,
3      L_Name TEXT NOT NULL,
4      L_Price INTEGER NOT NULL,
5      L_Brand TEXT NOT NULL
6  );
```

Output

Fig.3.19 : Creating a Table

4. Adding Record(s)

Once the table is created we can use the 'INSERT INTO' command to enter a single row in the table. This command requires the table name, column names and values for the record we wish to add in the table. Be careful to use the values according to the schema, sequence matters.

```
1  INSERT INTO Laptop (L_ID, L_Name, L_Price, L_Brand)
2  VALUES ('L1', 'Dell XPS 13', 100000, 'Dell');
```

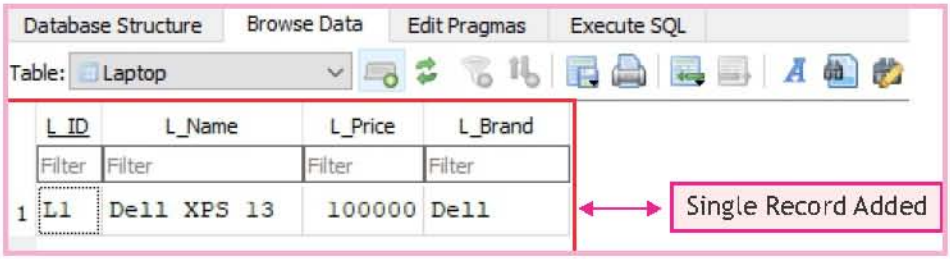


Fig.3.20 :Adding single record

Rather than adding single record, we can add multiple records as well using a single 'Insert Into' command as shown in Fig 3.21.

```
1  INSERT INTO Laptop (L_ID, L_Name, L_Price, L_Brand)
2  VALUES
3      ('L2', 'MacBook Air', 90000, 'Apple'),
4      ('L3', 'HP Spectre x360', 120000, 'HP');
5
```

Output

Database Structure | Browse Data | Edit Pragmas

Table: Laptop

	L_ID	L_Name	L_Price	L_Brand
	Filter	Filter	Filter	Filter
1	L1	Dell XPS 13	100000	Dell
2	L2	MacBook Air	90000	Apple
3	L3	HP Spectre x360	120000	HP

Multiple Record Added

Fig.3.21 :Adding multiple records

In this fashion, we can create additional tables and populate the data.

5. Querying Records from Table using Select

'Select' is the mostly used query to select records from the table. An '*' means all records in that particular table. Alternatively, we can mention the column names and only the specified columns will be displayed for all records.

Though select query fetches all records from the table but we can apply condition using where clause to get matching records. The matching can be a single or multiple records satisfying the condition.

```
1  SELECT * FROM Customer;
2
```

Output

	C_ID	C_Name	C_Address	C_Phone
1	C1	Qamar	New Address	123-4567
2	C2	Mahtab	House: 456	987-6543
3	C3	Chaand	House: 789	543-2109

Select All Records

Fig.3.22 : Querying Records from Table using Select

```
1  SELECT * FROM Customer
2  WHERE C_ID = 'C2';
3
```

	C_ID	C_Name	C_Address	C_Phone
1	C2	Mahtab	House: 456	987-6543

Record with ID 'C2'

Output

Fig.3.23 : Select query fetches all records from the table

Activity 7

Populate data for tables 'Customer', 'Order' and 'OrdreDetails'.

6. Update Record

For updating a record or a particular column, 'update' command is used. A good practice is to use select statement with where clause so that the desired record is retrieved. Thereafter for the same condition use the update command.

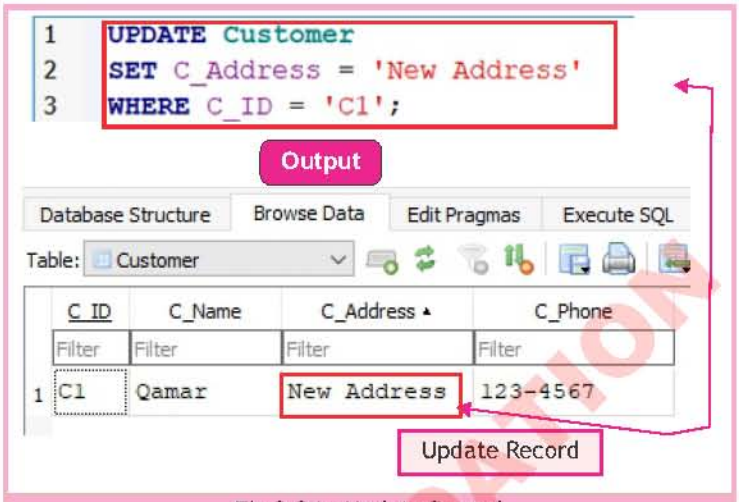


Fig.3.24 : Update Record

7. Use of Order By Clause

There are scenarios where records should be displayed in a sorted order. For that reason 'order by' clause is used in the query and the result will be displayed in ascending order, by default. An additional 'DESC' keywords is used with 'order by' clause to display the results in descending order.

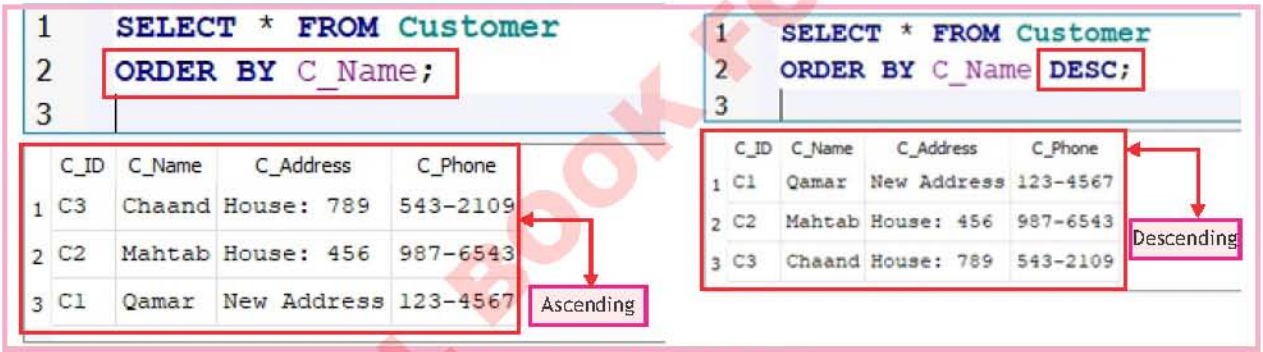


Fig.3.25: Use of Order By Clause

8. Deletion of Record(s)

You may delete a single or multiple entries from a table using 'Delete' command. However the table structure will remain in the database. It works just like select statement with an '*' all records will be deleted while using 'where' clause only the selected record(s) will be deleted.

Fig.3.26: Deletion of Record(s)

3.3 Techniques for Testing & Debugging

3.3.1. Debugging

Debugging is the process to analyze the code for bugs and errors and help in locating them, such that when fixed error-free program can run smoothly and deliver desired results. The simplest of debugging involves adding multiple print statements in a program to get the idea whether the program is behaving as expected or not. This does not mean that we should add print statements on each line or every second line. The idea is that wherever some formulae, expression or a critical value update is in the code, using print statements it should be validated that particular variable, line or block of code is behaving as desired.

To make the point clearer, let us refer the matrix transpose example as shown in Fig. 3.27. Here on line 8, we have intentionally placed a bug, by switching the positions of the matrix indices. Once we run it the output shows that matrix and transpose matrix are same. So, within the nested loop, we have placed multiple print statements which we not only label the print statement for identification purposes, but also print the values of different variables and data structures. For example, 'Print-1' on line 6 ensures that the 'new_row' gets always initialized whenever the outer loop index updates. Similarly, 'Print-3' on line 10 prints the values which will be appended in the transpose matrix after completion of the inner loop. On line 9, 'Print-2' is placed inside the nested for-loop to print the values of matrix indices and points out that the values of matrix are as it is assigned to transpose and needs to be changed.

```
1  matrix = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
2  transpose = []
3
4  for col in range(len(matrix[0])):
5      new_row = []
6      print('Print1-Outer Loop: ',new_row)
7      for row in range(len(matrix)):
8          new_row.append(matrix[col][row])
9          print('Print2-Inner Loop: row: ',row,'- col: ', col)
10     print('Print3-Outer Loop: ',new_row)
11     transpose.append(new_row)
12 for row in transpose:
13     print(row)
```

Output

```
Print1-Outer Loop:  []
Print2-Inner Loop: row:  0 - col:  0
Print2-Inner Loop: row:  1 - col:  0
Print2-Inner Loop: row:  2 - col:  0
Print3-Outer Loop:  [1, 2, 3]
Print1-Outer Loop:  []
Print2-Inner Loop: row:  0 - col:  1
Print2-Inner Loop: row:  1 - col:  1
Print2-Inner Loop: row:  2 - col:  1
Print3-Outer Loop:  [4, 5, 6]
Print1-Outer Loop:  []
Print2-Inner Loop: row:  0 - col:  2
Print2-Inner Loop: row:  1 - col:  2
Print2 Inner Loop: row:  2   col:  2
Print3-Outer Loop:  [7, 8, 9]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

Fig.3.27: Debugging with Print Statements

This code performs the transpose of a 3x3 matrix (swapping rows with columns). Here's a simple explanation:

- Line 2-3: Initializes a matrix and an empty list transpose to store the result of the transposed matrix.
- Line 5-8: The outer loop goes through each column in the original matrix. Inside this loop, a new row (new_row) is created.
- Line 9-12: The inner loop iterates through each row of the matrix, appending the corresponding column value to new_row.
- Line 14-15: Once the inner loop finishes, the new_row (a transposed row) is added to transpose.
- Line 16-17: Finally, the code prints the transposed matrix by printing each row in transpose.

Advanced debugging involves more than basic print commands by using techniques like breakpoints and watchpoints to identify and fix bugs and errors. Breakpoints are placed on the line of code while watchpoints are set on variables and expressions or formulae. Multiple breakpoints and watchpoints can be placed in a program. These methods are especially useful for handling complex issues.

The same code of Fig.3.27 can be placed in a debug mode in an IDE like Visual Studio.Net or an online IDE like REPLIT which provide a better GUI interface like selecting the line and setting the breakpoint on the said line. When run in debug mode step by step, the debugger not only stops at the breakpoint but also provides the values of the different variables and data structures. This helps in analyzing the data and overall behavior of the program. Fig. 3.28 shows the code while the debugger is on, reflecting the different variable values and the corresponding breakpoint set at line 6.

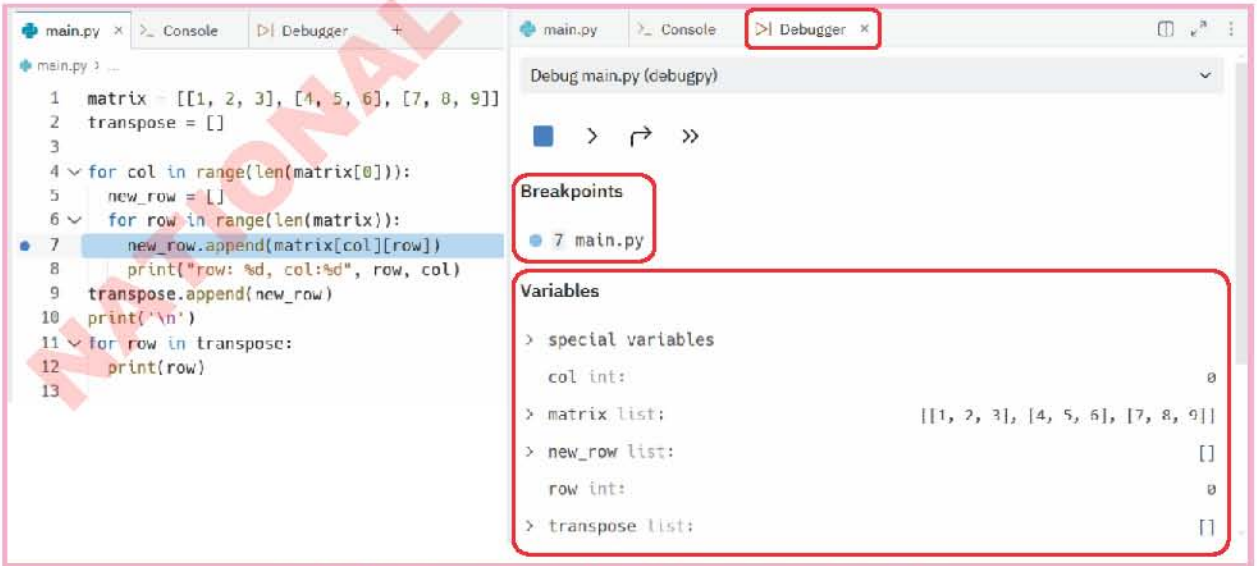


Fig.3.28: Debugging the code with breakpoints and watchpoints

3.3.2 Python Unit test

Python's built-in unittest framework is a tool that helps you check if your code works in the desired way. Rather than manually testing your code by running it, unittest lets you write small tests that check specific parts of your program. You may check it for some values to check whether it is returning the desired and correct result. But as the lines of code increase the code may become compilcated to test it manually. unittest is quite useful in such scenarios to write a set of tests to ensure the function is executing correctly for various inputs.

So, for a written function that performs some operation, you would write another function (a test function) to check if the first function gives the correct result. Use assertions to compare the actual output with the expected output. For example, you have a function that adds two numbers, like:

```
1   def add(a, b):                  # Function to be tested
2       return a + b
3
4   def test_add():                 # Test function
5       result = add(2, 3)          # Test case 1: Check if 2 + 3 equals 5
6       assert result == 5, f"Test failed: Expected 5 but got {result}"
7
8       result = add(0, 0)          # Test case 2: Check if 0 + 0 equals 0
9       assert result == 0, f"Test failed: Expected 0 but got {result}"
10
11      result = add(-1, 1)         # Test case 3: Check if -1 + 1 equals 0
12      assert result == 2, f"Test failed: Expected 0 but got {result}"
13
14      print("All tests passed!")
15
16  test_add()                      # Run the test function
```

Fig.3.29: Function to add 2 numbers with tests

On line 1 the add() function adds two numbers. The test_add() function on line 4 tests the add() function with different inputs, like as input parameters two positive numbers are provided on line 5, two null values on line 8 and a positive and negative number on line 11. The assert statement checks if the output of add() matches the expected value. If not, it raises an error with a custom message. If all assertions pass, it prints "All tests passed!"

To have a better understanding you may check if for different values and results. As a next step, we assume that expected result was 2 on line 12 and we oversaw the negative value on line 11. So, definitely we would get the 'Test failed: ...' message. To investigate it we can debug the code by applying breakpoints in the code:

```python
def add(a, b):                  # Function to be tested
    return a + b

def test_add():                 # Test function
    result = add(2, 3)          # Test case 1: Check if 2 + 3 equals 5
    assert result == 5, f"Test failed: Expected 5 but got {result}"

    result = add(0, 0)          # Test case 2: Check if 0 + 0 equals 0
    assert result == 0, f"Test failed: Expected 0 but got {result}"

    result = add(-1, 1)         # Test case 3: Check if -1 + 1 equals 0
    assert result == 2, f"Test failed: Expected 0 but got {result}"

    print("All tests passed!")

test_add()                      # Run the test function
```

Fig.3.30: Adding Breakpoints

Let's execute the code and debug it. We deployed 3 breakpoints on line 6, 9 and 12 to track the values to be checked. As watchpoint we added 'result' variable. As the debugging starts and it stops on first breakpoint, the result is as expected.

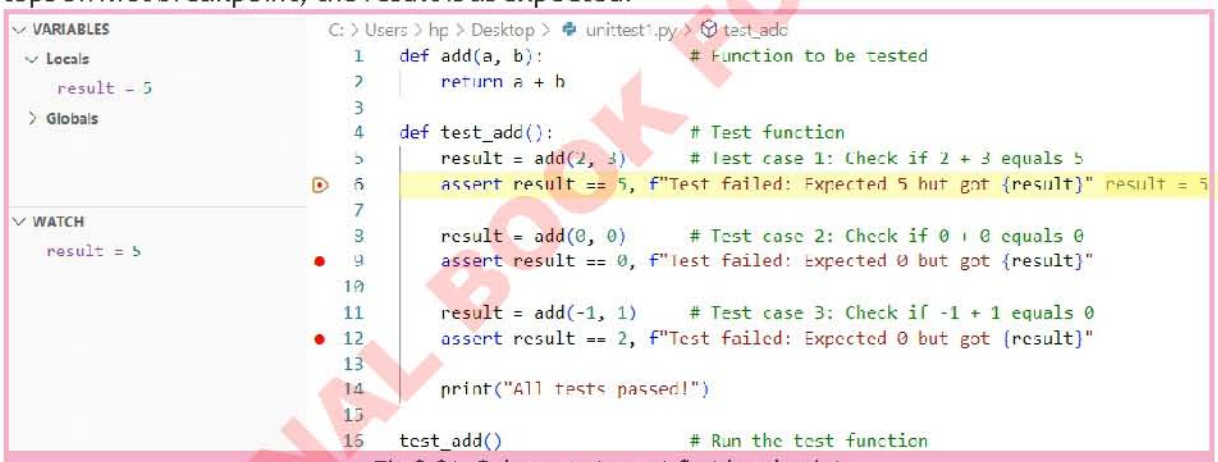


Fig.3.31: Debugger stops at first breakpoint

Same goes with the second breakpoint.

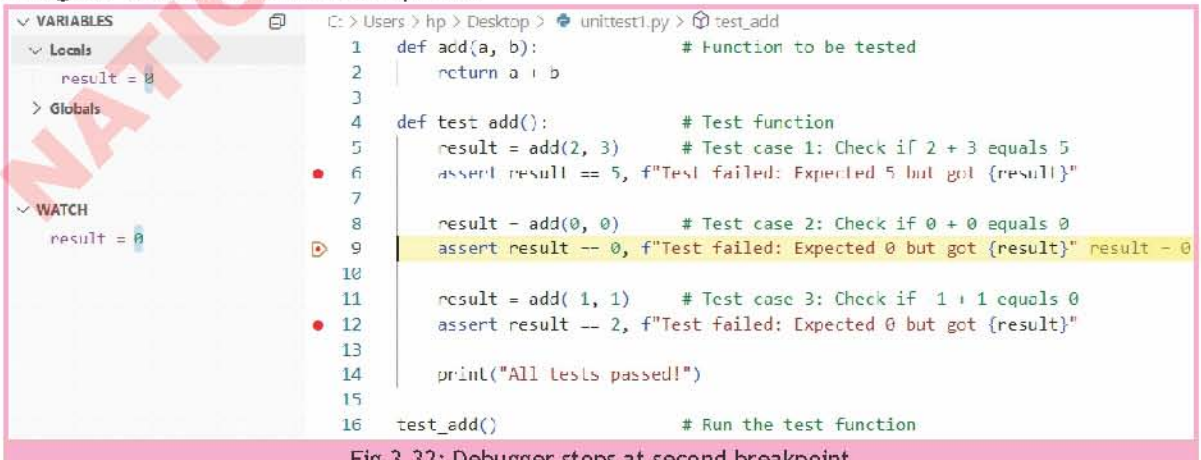


Fig.3.32: Debugger stops at second breakpoint

However, for the third breakpoint the result variable is flagged in the code and watchpoint shows the mismatched value as compared to the expected value to track the bug.

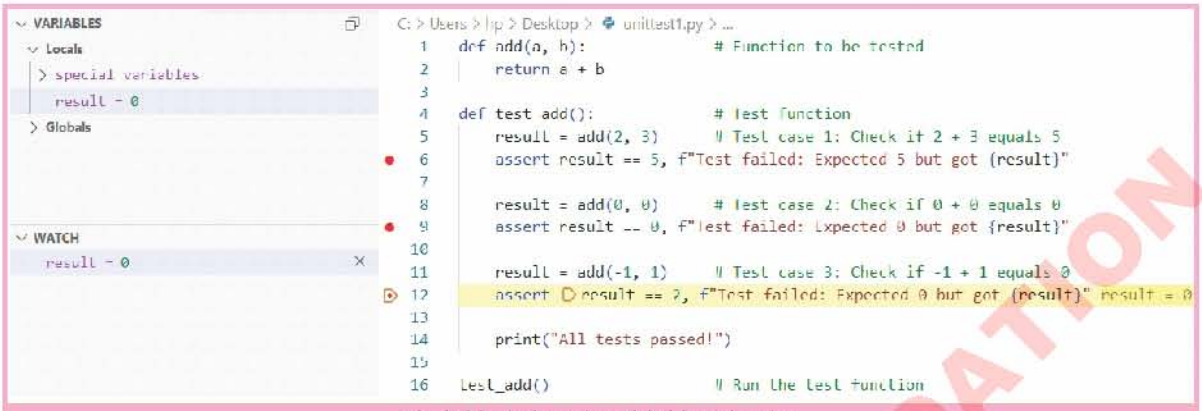


Fig.3.33: Debugging third breakpoint

The same code is changed for multiplication and tests are adjusted accordingly. Will all tests be passed? If not, which one will fail and why? Try debugging the code.

```python
def multiply(a, b):
    return a * b

def test_multiply():
    result = multiply(2, 3)
    assert result == 6, f"Test failed: Expected 6 but got {result}"

    result = multiply(4, 0)
    assert result == 1, f"Test failed: Expected 1 but got {result}"

    result = multiply(-1, -1)
    assert result == 1, f"Test failed: Expected 1 but got {result}"

    print("All tests passed!")

test_multiply()
```

Fig.3.34: Function to multiply 2 numbers with tests

Summary

- **Programming Paradigm:** Refers to different styles and methodologies of programming used to solve problems, making code more structured and easier to manage.
- **Complexity Reduction:** Helps to break down complex problems into simpler parts resulting in easier to understand and solve.
- **Functional Programming:** programs are developed in the form of functions.
- **Object-Oriented Programming (OOP):** Revolves around the use of objects and classes to represent real-world entities.
- **List in Python:** A data structure that stores an ordered collection of items which can be accessed by their index. Lists are mutable i.e. their contents can be changed.
- **Dictionary in Python:** A key-value pair data structure used for mapping keys to values such that every key in a dictionary is unique and values can be accessed using their corresponding keys.
- **Files in Python:** File handling allows Python programs to read from and write to files and store data permanently. Common operations include opening, reading, writing and closing files.
- **Databases:** A structured collection of data used for efficient storage and retrieval. Tools like MySQL, SQLite, and PostgreSQL are commonly used for managing databases.
- **Data Normalization:** A database design technique aimed at reducing redundancy and dependency by organizing fields and tables of data.
- **Primary Key:** Is a unique identifier for records in a database table emphasizing each record can be distinctly retrieved or updated.
- **GUI in Python:** A graphical user interface that allows users to interact with applications visually, using elements like buttons, text fields and windows, etc. Python's pygame library is commonly used to create GUIs.
- **Nested Lists:** Lists within lists are nested lists that are used for storing and organizing complex data structures like matrices.
- **List as Value in Dictionary:** Python allows the use of lists as values in dictionaries such that multiple items can be stored under a single key.
- **Testing in Python:** Testing ensures the correctness of code.
- **Unit Test:** Unit testing focuses on testing small, isolated units of functionality, such as individual functions or methods.
- **Debugging:** The process of identifying and fixing bugs or errors in a program. Techniques include using print messages, where messages are printed to the console to help track the flow of a program, and more sophisticated tools like debuggers that allow stepping through the code line by line.

Exercise

Select the best answer for the following Multiple-Choice Questions (MCQs).

1. Main purpose of the programming paradigm is _____ .
   a. Increase code length
   b. Complexity reduction
   c. Eliminate bugs
   d. Improve hardware performance
2. _____ programming paradigm emphasizes the use of functions without changing state.
   a. Procedural b. Functional c. Object-oriented d. Structural
3. Object-Oriented Programming primarily revolves around _____ of the following.
   a. Data structures b. Functions c. Objects d. Variables
4. In Python _____ data structure is used to store a collection of items.
   a. List b. Dictionary c. Tuple d. Set
5. A dictionary stores _____ in Python.
   a. Keys only b. Values only c. Key-value pairs d. Sequential data
6. An example of disk I/O _____ .
   a. Storing a variable in memory
   b. Reading a file from the hard disk
   c. Defining a function
   d. Looping through a list
7. To write content to a file in Python _____ method is used.
   a. file.write() b. file.insert() c. file.add() d. file.modify()
8. A primary key in a database is _____ .
   a. A key that has a default value
   b. A unique identifier for records
   c. A key for linking multiple databases
   d. A key used for temporary tables
9. A common database normalization goal is _____ .
   a. Reducing redundancy
   b. Maximizing data size
   c. Increasing the number of tables
   d. Storing duplicate records
10. _____ key is used to uniquely identify a record in a related table.
    a. Primary b. Secondary c. Foreign d. Unique
11. To create a simple GUI in Python, _____ library is used.
    a. Pygame b. OS c. Sys d. math
12. A nested list in Python is referred as _____ .
    a. A list containing another list
    b. A list of dictionaries
    c. A dictionary of lists
    d. A set containing multiple lists
13. The purpose of unit testing in Python is _____.
    a. Test individual units of code
    b. Test the entire program
    c. Debug the entire program
    d. Optimize the code for speed

14. The primary goal of debugging is _____ .

a. Writing new code
b. Fixing errors in the code
c. Increasing program speed
d. Creating a graphical interface

15. Python function used to print debugging messages to the console _____ .

a. print()
b. debug()
c. trace()
d. log()

16. Output of the following code will be:

```
my_dict = {"a": [1, 2], "b": [3, 4]}
print(my_dict["a"][1])
```

a. 1
b. 2
c. [1, 2]
d. [3, 4]

17. The code will print "Inner" _____ times.

```
for i in range(3):
    for j in range(2):
        print("Inner")
```

a. 2
b. 3
c. 5
d. 6

Give short answers to the following Short Response Questions (SRQs).

1. Elaborate programming language paradigm and why is it important for software development?
2. How do you interpret the concept of complexity reduction in programming? How does dividing code into modules help manage complexity?
3. Compare Functional Programming and Object-Oriented Programming (OOP) paradigms. Enlist main differences in their approach to problem-solving.
4. What is Disk I/O and why is it important for system performance?
5. Describe the difference modes in file handling in Python.
6. What is the benefit of using the 'with' keyword while working with files in Python?
7. How can you create a simple GUI application using Pygame? Enlist the steps involved.
8. Describe the use of a nested list in Python and how can it be used to represent a matrix?
9. What is data normalization and why is it used in database management?
10. What is debugging and why is it important in software development?
11. What seems to be the problem in:

```
sample_dict = {"a": 1, "b": 2}
print(sample_dict["c"])
```

12. Final value of my_list after the following lines would be:

```
own_list = [1, 2, 3];
own_list.append(4, 5)
```

Give long answers to the following Extended Response Questions (ERQs).

1. Write a Python function that calculates the factorial of a given number. Test the function with 5 different inputs.
2. Implement a Python class 'Rectangle' with methods to calculate the area and perimeter of a rectangle. Create instances of the class and print the perimeter of rectangle.
3. Write a Python program highlighting the use of functions to simplify a problem. For example, create a program that calculates the average of a list of numbers and by using separate functions to add all numbers, , marking repeated values and calculate the average of the list with and without repeating values.
4. Create a Python dictionary to store student names as keys and their grades as values. Write a program to add new students, update existing students' grades, and remove students from the dictionary.
5. Write a Python program to create a file named data.txt, write a string to it and then read the content of the file and display the output.
6. Create a Python program that appends a new line of text to an existing file. If the file does not exist, it should create a new file.
7. Write Python code using Pygame that creates a simple window with a button. When the button is clicked, it should display a message on the window.
8. Develop a Python program that calculates and prints the transpose of a 3x3 matrix represented using a nested list.
9. Write a Python function that counts the number of occurrences of each letter in a text file and prints the results.
10. Write a Python program to connect to a database and demonstrate how to insert, update and retrieve data from the database. Explain the importance of database connectivity in Python applications.
11. Explain unit testing and its importance in Python programming. How does it help in catching errors early during the development process? Include an example of a unit test for a simple Python function.
12. Write a Python code that selects a number between 1 and 10 and the player will try to guess the number in 5 tries. For every incorrect guess, the program provides a hint whether the last guess was on higher or lower side. If the number is successfully guessed then a Congratulating message should be displayed along with in how many attempts the number was guessed correctly. Otherwise another message should be displayed along with the chosen number.

    The outputs should look like:

```
Hello, What's your name?Abrar
okay! Abrar I am Guessing a number between 1 and 100:
50
Your guess is too low
75
Your guess is too high
60
Your guess is too high
55
Your guess is too low
58
You guessed the number in 5 tries!
```

```
Hello, What's your name?Azam
okay! Azam I am Guessing a number between 1 and 100:
50
Your guess is too low
75
Your guess is too low
90
Your guess is too high
85
Your guess is too high
80
Your guess is too high
You did not guess the number, The number was 78
```

13. The following code is supposed to open a file 'abc.txt' and counts the number of occurrences of the character 's' and displays output like:

```
1   file_name = 'abc.txt'
2   target_letter = 'S'
3   try:
4       file = open(file_name, "r")
5       text = file.read()
6       file.close
7   except FileNotFoundError:
8       print(f"Error: File '{file_name}' not found.")
9       exit()
10  count = 0
11  for char in text:
12      if char = target_letter:
13          count += 1
14  print("The letter '{target_letter}' appears " + count + " times in the file '{file_name}'.")
```

- Sample text in file 'abc.txt': I scream, you scream, we all scream for ice cream.
- Sample Output: The letter 's' appears 3 times in the file 'abc.txt'.

However, the code does not execute. Debug and remove any bug(s) from the code.

Mini Project 1

Design a game where multiple objects are falling from the top. The objective is to "Catch The Falling Objects" using arrow keys. For every catch, a score increases to 1 point. The game ends whenever an object is missed.

Mini Project 2

Design two textboxes such that the first is for entering text while the second shows the text converted to uppercase. The user can click on the first textbox and type a sentence. As the user types, the second textbox automatically displays the uppercase version. The second textbox is not editable.

- **Example:** If you're launching a fitness app designed for new mothers, your targeted audience would be women who have recently given birth and are looking for postpartum fitness solutions. Instead of targeting all women or fitness enthusiasts, focus your marketing campaigns on this specific demographic through platforms like Instagram and parenting blogs that cater to this group. This helps ensure that the product resonates deeply with the users who are most likely to benefit from it.

2. Soft Launch

- **Strategy:** Consider a soft or limited launch initially to gather feedback from a smaller user base and make improvements before a broader release.
- **Example:** Airbnb used a soft launch when it first introduced its platform. They initially offered their service only in San Francisco, allowing the team to refine the user experience, handle logistical issues, and build trust with both hosts and guests before scaling to other regions. This helped them test their MVP with real users without overwhelming their resources or reputation.

Fig.8.4: Soft Launch

- **Approach:** A soft launch could involve limiting access to a specific city or group, and in the case of a mobile app, only releasing it to users who sign up for early access or join an invitation list. This helps iron out bugs, optimize the user experience, and understand user behavior before scaling up.

3. Beta Testing

- **Strategy:** Invite a select group of users to participate in beta testing to uncover potential issues, generate early user reviews, and gather testimonials.
- **Example:** Dropbox initially ran a private beta test before launching to the public. They invited a small number of users (mostly tech-savvy individuals) to test the platform and provide feedback. This group was instrumental in identifying bugs, suggesting improvements, and refining the product's features. In exchange, users were provided with extended free storage as a reward.

Fig.8.5: Beta Testing

- **Approach:** You can run a closed beta for a small set of users, offering them early access in exchange for detailed feedback. For example, a new e-commerce platform could send invites to a few hundred users, track their behavior on the site, and get detailed feedback about navigation, checkout experience, and product offerings.

4. Marketing and Promotion

- **Strategy:** Develop a marketing plan that includes strategies for creating awareness,

generating interest, and driving user acquisition through digital marketing, social media, and content marketing.

Fig.8.6: Marketing and Promotion

- **Example:** Slack, a business communication tool, created awareness by using content marketing early on. They targeted a niche audience of tech-savvy companies that could benefit from their real-time messaging service. They ran targeted ad campaigns on sites like TechCrunch and offered a freemium model, which encouraged users to sign up and experience the product.
- **Approach:** Start with digital marketing channels that resonate with your targeted audience. For example, if you're launching a food delivery app, focus on local SEO, social media ads targeting food lovers, and partnerships with food bloggers to drive interest. Create engaging content (e.g., how-to guides, blog posts, and user stories) to generate organic interest.

5. Feedback Channels

- **Strategy:** Set up clear channels for users to provide feedback within the product. Encourage users to report issues, suggest improvements, and share their experiences.
- **Example:** Tesla has a built-in feedback system in their cars where users can report bugs, suggest new features, and provide feedback directly to the company. They also engage with users through social media and forums to understand their needs and improve their products based on this input.

Fig.8.7: Feedback Channels

- **Approach:** You can integrate an in-app feedback form or provide an email address for users to contact your support team. It's important to not just collect feedback but to act on it by updating the product and informing users of the improvements based on their suggestions. This builds trust and helps in refining the MVP.

6. Monitoring and Analytics

- **Strategy:** Implement robust analytics tools to track user behavior, engagement, and key performance metrics. This data will help you make informed decisions and measure the MVP's success.
- **Example:** Spotify uses advanced analytics to track user behavior, including the types of music played, skips, and playlist creation. This allows them to understand user preferences and personalize

Fig.8.8: Monitoring and Analytics

recommendations, which directly affects engagement and retention.

- **Approach:** Set up analytics tools like Google Analytics, Mixpanel, or Amplitude to track how users interact with your MVP. Measure important metrics like user retention, conversion rates, feature usage, and session duration. If your product is an app, track install-to-sign-up conversion rates and feature adoption to understand how users are engaging with the core functionality of the MVP.

7. Customer Support

- **Strategy:** Provide responsive customer support to address user questions, issues, and concerns promptly. A positive support experience can lead to user retention and referrals.
- **Example:** Zappos, the online shoe retailer, is known for exceptional customer service. They offer free returns and 24/7 customer support, ensuring that users feel supported throughout their buying journey. This has built a loyal customer base that values the brand's dedication to service.

Fig.8.9: Customer Support

- **Approach:** Offer multiple support channels, such as live chat, email, and a comprehensive FAQ section. Additionally, creating a community forum or leveraging user guides and video tutorials can help users find answers to common issues without needing to contact support. Addressing problems quickly and positively builds a strong reputation, especially during the MVP stage when bugs or limitations are inevitable.

Summary of Approach:

Launching an MVP requires a well-thought-out strategy to ensure that you're focusing on the right audience, iterating based on feedback, and delivering a strong first impression. By following these strategies and examples, you can reduce the risks associated with a new product, build a loyal user base, and ensure that your MVP can scale effectively once it's ready for a broader release.

8.1.3 Collecting User Feedback and Data

Collecting user feedback and data is crucial during the MVP phase to drive continuous improvement:

Feedback Mechanisms: Implement various feedback mechanisms within the MVP, such as in-app surveys, feedback forms, or direct contact options. Make it easy for users to share their thoughts.

User Interviews: Conduct one-on-one interviews or focus groups with users to gain deeper insights into their experiences and pain points.

Analytics Tools: Use analytics tools to track user behavior and gather quantitative data. Monitor user engagement, conversion rates, and other relevant metrics.

Feedback Analysis: Regularly review and analyze the feedback and data collected. Look for patterns and trends that can inform feature enhancements and optimizations.

Iterative Development: Based on user feedback and data analysis, prioritize and implement changes and new features in subsequent iterations of the MVP.

Engagement Metrics: Pay attention to user retention rates and user engagement metrics. These indicators can reveal the MVP's stickiness and user satisfaction.

Feature Validation: Use data and user feedback to validate or pivot on certain features or aspects of the product. Ensure that changes align with user needs and preferences.

The MVP launch is just the beginning of your product's journey. By actively collecting user feedback and data, you can make informed decisions, continuously improve the product, and work towards achieving your long-term product goals. This iterative approach is key to creating a successful and user-centric product.

Fig.8.10: Feedback Mechanisms

8.2 Iterative Development

8.2.1 The Concept of Continuous Improvement

Continuous improvement, often referred to as iterative development, is a fundamental principle in the product development process. It involves a cyclical approach to refining and enhancing a product or service over time. The core idea is to never stop looking for ways to make your product better, more efficient, and more aligned with user needs and market trends.

Key aspects of continuous improvement include:

Feedback-Driven: Continuous improvement relies heavily on feedback from users, stakeholders, and data analysis. It's about actively seeking and listening to input and insights that can inform changes and refinements.

Iterative Cycles: Development occurs in iterative cycles or sprints, with each cycle focused on specific improvements or enhancements. These cycles are typically short and result in frequent updates or releases.

Flexibility: Teams must be adaptable and open to change. The ability to pivot or adjust course based on feedback and new information is a hallmark of continuous improvement.

Prioritization: Not all improvements are equal. Prioritization is essential to focus efforts on changes that have the most significant impact on user satisfaction, usability, or business goals.

Data-Driven Decision-Making: Decisions regarding improvements should be supported by data and evidence, rather than assumptions or opinions.

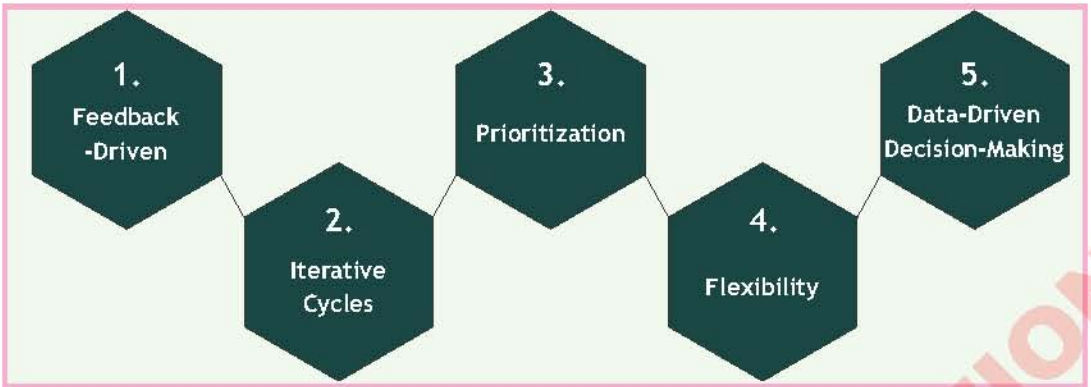


Fig.8.11: Continuous improvement

8.2.2 Using Feedback to Refine and Enhance the Product or Service

Feedback plays a central role in the iterative development process. Here's how to effectively use feedback to refine and enhance your product or service:

Fig.8.12: Using Feedback to Refine and Enhance the Service

Gathering Feedback: Actively seek feedback from users through various channels, including in-app feedback forms, surveys, customer support interactions, and user testing sessions.

Data Analysis: Analyze user behavior and engagement data to identify patterns, trends, and areas where users may be experiencing difficulties or drop-offs.

Prioritization: Prioritize feedback and improvement ideas based on their potential impact and alignment with your product goals. Not all feedback should result in immediate action; some may be deferred for future iterations.

Clear Objectives: Define clear objectives for each iteration. What specific improvements or changes are you aiming to achieve, and how will you measure success?

Cross-Functional Collaboration: Collaboration among multidisciplinary teams (designers, developers, marketers, etc.) is crucial to implement feedback effectively. Different teams can contribute their expertise to address various aspects of the product.

User Testing: Continuously conduct user testing with real users to validate proposed improvements and gather fresh feedback on the changes made.

Communication: Keep stakeholders informed about the progress and outcomes of each iteration. Transparent communication helps maintain alignment and support for the continuous improvement process.

8.2.3 Scaling and Expanding Based on MVP Success

As your MVP (Minimum Viable Product) gains traction and success, the natural progression is to scale and expand. Here's how to do it effectively:

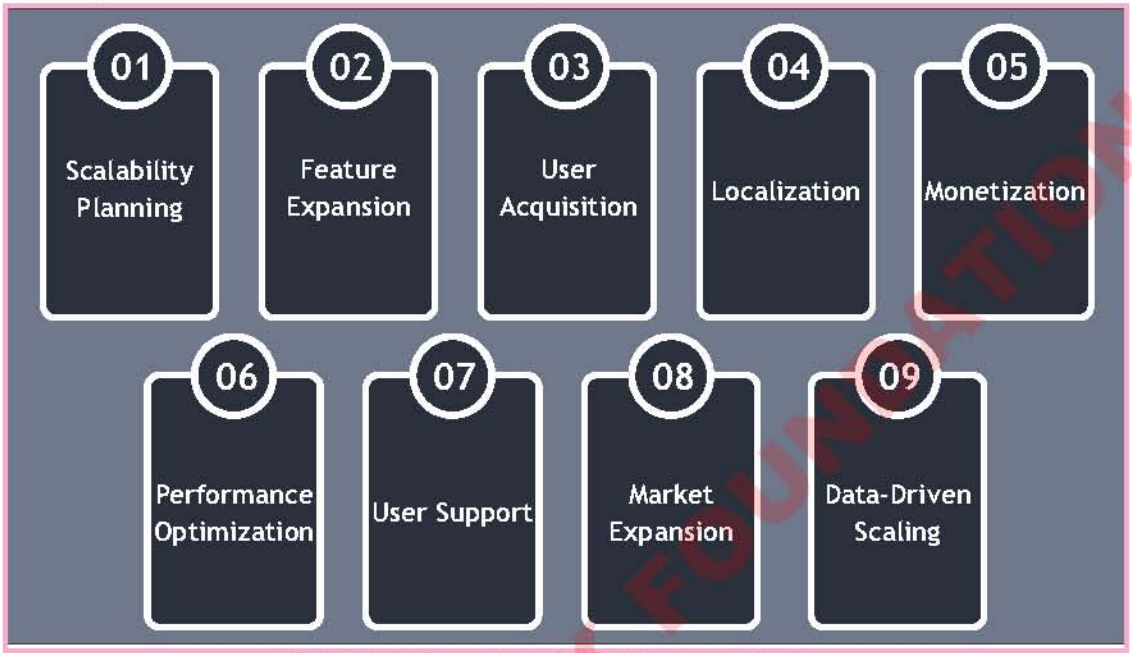


Fig.8.13: Scaling and Expanding Based on MVP Success

Scalability Planning: Ensure that the architecture and infrastructure of your product can handle increased demand and usage as you scale. Be prepared to invest in server capacity, security measures, and scalability features.

Feature Expansion: Gradually add new features and functionalities based on user needs, feedback, and market trends. However, maintain a focus on simplicity and user-centric design, avoiding feature bloat.

User Acquisition: Develop strategies to acquire more users. This may involve marketing efforts, partnerships, or referral programs. Consider user acquisition costs and strategies for retaining new users.

Localization: If applicable, consider localization for different regions or languages to expand your user base globally.

Monetization: Explore different monetization models such as subscriptions, advertising, or in-app purchases. The choice of monetization should align with your user base and product niche.

Performance Optimization: Continuously monitor and optimize the performance of your product, ensuring that it remains fast, reliable, and user-friendly even as it scales.

User Support: Be prepared to scale your customer support and user assistance capabilities to handle increased user inquiries and issues.

Market Expansion: If your MVP has succeeded in a specific market or niche, consider expanding to related markets or niches with similar needs and preferences.

Data-Driven Scaling: Make scaling decisions based on data and user behavior. Avoid premature scaling that can strain resources unnecessarily.

In summary, continuous improvement through iterative development is a core principle in product development. It involves using feedback and data to refine and enhance your product, and as your MVP succeeds, you can strategically scale and expand to reach a broader audience and achieve long-term success.

8.2.4 Evaluation Criteria for Prototypes and MVPs

When evaluating prototypes and Minimum Viable Products (MVPs), it's essential to consider specific criteria that align with the goals of the project. Here are some evaluation criteria:

Alignment with User Needs: Assess how well the prototype or MVP addresses the identified problem or need of the target audience. Does it effectively solve the problem?

Usability: Evaluate the user interface, navigation, and overall user experience. Is the product easy to use, and does it provide a smooth interaction flow?

Functionality: Assess whether the product's core features and functionalities work as intended. Are there any technical issues or bugs?

Scalability: Consider the potential for scalability and future growth. Is the architecture and infrastructure designed to handle increased demand?

Market Fit: Evaluate whether the product aligns with market demands and trends. Does it offer a unique value proposition or competitive advantage?

Feedback Incorporation: Assess how well user feedback has been incorporated into the prototype or MVP. Has feedback-driven improvement been evident?

Iterative Development: Consider the evolution of the product over time. Has it gone through multiple iterations to enhance its features and user experience?

Performance Metrics: Use relevant metrics to measure the success of the MVP. For example, track user engagement, conversion rates, or user satisfaction scores.

Fig.8.14: Evaluation Criteria for Prototypes and MVPs

8.2.5 Providing Constructive Feedback

Providing constructive feedback is essential for guiding students' learning and improvement. Here are tips for offering effective feedback:

Specificity: Be specific about what the student did well and what needs improvement. Vague feedback is less helpful.

Balance: Offer a balance of positive feedback and areas for improvement. Acknowledge the student's strengths before addressing weaknesses.

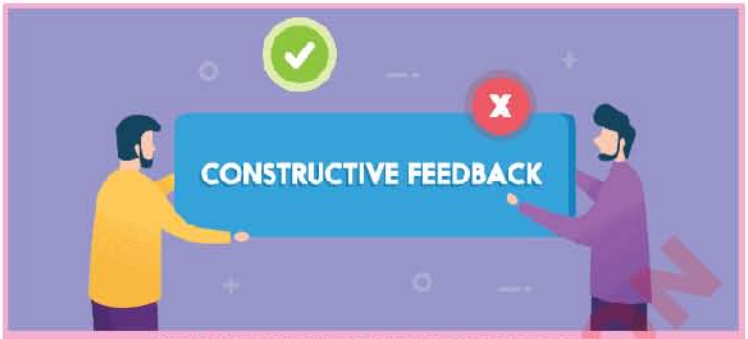


Fig.8.15: Providing Constructive Feedback

Focus on the Work, Not the Student: Frame feedback in terms of the project or assignment rather than making it personal. For example, say, "The report lacks sufficient data analysis," instead of "You didn't analyze the data well."

Use the "Feedback Sandwich": Start with positive feedback, provide constructive criticism, and end with encouraging remarks or suggestions for improvement.

Suggest Actionable Steps: Offer specific suggestions or steps the student can take to address the identified areas for improvement.

Encourage Self-Reflection: Prompt students to reflect on their work and what they've learned from the feedback. This encourages metacognition.

Timeliness: Provide feedback promptly, ideally soon after the completion of assignments or assessments, so it's still relevant to the student.

Open Dialogue: Encourage students to ask questions or seek clarification about the feedback. Foster a dialogue to ensure they understand and can act on the feedback.

Remember that effective feedback is a two-way communication process, and it should ultimately help students grow and develop their skills in product development.

8.3 Conclusion and Future Directions

8.3.1 Recap of Key Concepts and Skills Learned

In this course on product development, you've gained valuable knowledge and skills essential for creating successful products and services. Let's recap some of the key concepts and skills you've learned:

Product Development Process: You've explored the step-by-step process of product development, from ideation and prototyping to launching Minimum Viable Products (MVPs) and iterative development.

Prototyping: You've learned the importance of prototyping and how to create low-fidelity and high-fidelity prototypes to validate your product ideas.

User Testing: You've discovered the significance of user testing in refining your products, gathering feedback, and ensuring a user-centric approach.

Iterative Development: You've embraced the concept of continuous improvement, where

feedback and data drive the enhancement of your products over time.

Launching MVPs: You've explored strategies for launching MVPs and scaling products based on their success, all while maintaining a user-centric focus.

Assessment and Evaluation: You've understood how to assess student progress and how to evaluate prototypes and MVPs effectively.

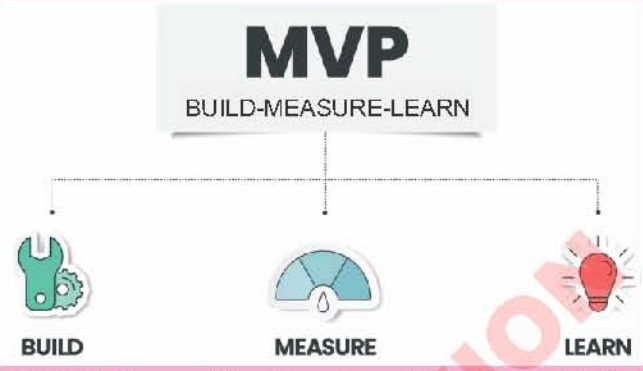


Fig.8.16: Recap of Key Concepts and Skills Learned MVP

8.3.2 Encouragement for Students to Apply These Principles in Real-Life Projects

As you move forward in your academic and professional journey, remember that the principles and skills you've learned in product development are highly transferable and valuable in real-life projects. Here's some encouragement:

Hands-On Experience: Seek opportunities to apply what you've learned in real-world projects or internships. Practical experience will deepen your understanding and expertise.

Entrepreneurship: If you're inclined towards entrepreneurship, consider starting your own venture. Apply the principles of product development to bring your ideas to life.

Innovation: Whether you work for a startup or a well-established company, innovation is key. Use your knowledge of product development to contribute fresh ideas and improvements to existing products or services.

Problem Solving: Product development skills are problem-solving skills. Apply them to tackle complex challenges in various domains, from technology and healthcare to education and beyond.

Collaboration: Collaboration is often essential in product development. Work effectively in interdisciplinary teams, leveraging each member's expertise to create successful products.

Fig.8.17: Encouragement for Students to Apply These Principles in Real-Life Projects

8.3.3 Mention of Potential Career Paths Related to Product Development

Product development skills open doors to a variety of exciting career paths. Here are some potential directions you might consider:

Product Manager: As a product manager, you'll oversee the development of a product from concept to launch, making strategic decisions and collaborating with cross-functional teams.

UX/UI Designer: Focus on user experience and user interface design, creating visually appealing and user-friendly products.

Software Developer/Engineer: Build and maintain software products, coding the features and functionalities envisioned in the product development process.

Entrepreneur: Start your own company and use product development principles to bring your innovative ideas to the market.

Business Analyst: Analyze market trends, user data, and feedback to make data-driven decisions in product development.

Project Manager: Manage the execution of product development projects, ensuring they are completed on time, within budget, and meet quality standards.

Innovation Consultant: Help organizations identify opportunities for innovation and guide them through the product development process.

Marketing Manager: Develop and execute marketing strategies for products and services, understanding the product's unique value proposition.

The field of product development is dynamic and offers a range of opportunities for those who are passionate about creating impactful and user-centered solutions. Whether you choose a career in technology, healthcare, entertainment, or any other industry, the principles you've learned here will serve as a solid foundation for success. Embrace the journey ahead with enthusiasm, and continue to learn and grow in the exciting world of product development.

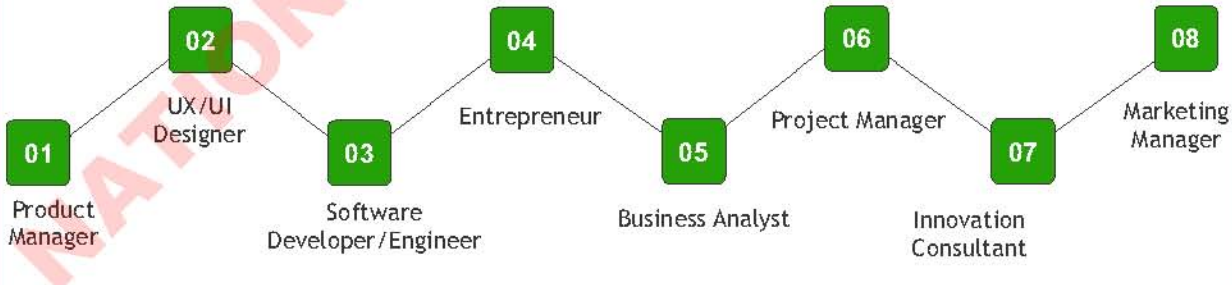


Fig.8.18: Potential Career Paths Related to Product Development

Case Study

Developing an MVP for a Sustainable Packaging Solution

Background:

A group of students is tasked with developing a Minimum Viable Product (MVP) for a new business that aims to tackle the growing issue of plastic waste in packaging. The students are required to identify a problem related to sustainability, brainstorm possible solutions, create an MVP, and present their business solution. Their goal is to launch a product or service that reduces the environmental impact of packaging while also meeting the needs of consumers and businesses.

Step 1: Identifying the Problem

Problem Identification:

The team conducts research on the environmental impact of plastic packaging and identifies the following issues:

- High Plastic Waste: Over 300 million tons of plastic are produced every year, with much of it used in packaging that ends up in landfills and oceans.
- Consumer Awareness: While consumers are increasingly aware of the need for sustainability, many still struggle to find accessible, affordable, and eco-friendly packaging alternatives.
- Business Pressure: Companies face pressure to meet sustainability goals, but the transition to green packaging solutions is often expensive or logistically difficult.

Customer Research:

To validate their findings, the team surveys local businesses, consumers, and environmental experts. They find that consumers prefer sustainable products, but are often unwilling to pay a premium for eco-friendly packaging. Businesses express frustration about the high cost of transitioning to greener alternatives.

Step 2: Creating Business Solutions

Brainstorming and Ideation:

The team holds a brainstorming session to generate solutions. A few ideas they consider include:

- Compostable Packaging: Packaging that breaks down naturally, such as plant-based materials like cornstarch or mushrooms.
- Reusable Packaging: Packaging that can be returned and reused, such as glass or metal containers.
- Eco-friendly Shipping Materials: Alternative shipping materials that are both sustainable and protective for products, such as biodegradable peanuts or cardboard made from recycled materials.

They decide to focus on **compostable packaging** made from plant-based materials, which they believe will be most accessible to consumers and businesses alike.

Feasibility Study:

The team conducts a feasibility study and determines that while plant-based materials are an effective alternative to plastic, there are challenges related to cost, scalability, and consumer acceptance. However, they believe they can mitigate these challenges by offering the product as a more affordable, bulk-purchase solution for small and medium-sized businesses, who are often the least able to afford expensive eco-friendly packaging.

Value Proposition:

Their value proposition is: "Affordable, compostable packaging solutions for small businesses that help reduce environmental impact and meet sustainability goals without breaking the bank."

Step 3: Building the Minimum Viable Product (MVP)

Core Features:

The students focus on developing an MVP that includes:

- Compostable Packaging: Simple, single-use packaging made from plant-based materials, such as cornstarch or sugarcane pulp.
- Custom Branding: The ability for businesses to print their logos and designs on the packaging to enhance brand recognition.
- Bulk Purchase Option: The option for businesses to order in bulk at a lower price point to make the solution more affordable.

Prototyping:

The team creates wireframes and mock-ups of the packaging design and how it will look when printed with branding. They also prototype a basic website for businesses to place orders.

MVP Development:

The students use a low-code platform to quickly set up an online store where businesses can place orders for the compostable packaging. They use a third-party supplier to create a small batch of the packaging to test with local businesses.

Iterative Feedback:

After launching the MVP, the students collect feedback from local small business owners. They learn that some businesses want packaging with more varied sizes and shapes, while others are interested in ensuring the material meets specific durability standards. They decide to iterate on the MVP by offering a few different sizes and improving the strength of the material.

Step 4: Presenting the Business Solution

Elevator Pitch:

The team crafts an elevator pitch: "Our sustainable packaging solution helps small businesses

reduce their carbon footprint by offering compostable packaging that's both eco-friendly and cost-effective. By switching to our affordable, plant-based packaging, businesses can meet sustainability goals and show consumers they care about the planet."

Business Model Canvas:

They use a Business Model Canvas to outline their plan:

- Key Partners: Suppliers of plant-based materials, local delivery companies.
- Key Activities: Product sourcing, packaging design, marketing, and customer support.
- Key Resources: Website platform, design tools, and supplier relationships.
- Value Proposition: Affordable and eco-friendly compostable packaging for small businesses.
- Customer Segments: Small to medium-sized businesses in the food, beauty, and retail industries.
- Channels: Online store, social media, and partnerships with environmental organizations.
- Revenue Streams: Direct sales, subscription model for regular deliveries.

Pitch Deck:

The team creates a pitch deck that includes:

1. Problem: The environmental impact of plastic packaging.
2. Solution (MVP): Affordable compostable packaging for small businesses.
3. Market Opportunity: Growing demand for sustainable products and packaging.
4. Business Model: Direct-to-business sales with a focus on bulk orders.
5. Go-to-Market Strategy: Marketing through social media, partnerships with eco-friendly influencers, and offering free samples to local businesses.
6. Financials/Projections: Expected costs, revenue from bulk orders, and long-term scalability.

Storytelling:

The team tells a compelling story about how their solution came about by combining their passion for the environment and their desire to help small businesses succeed. They emphasize the global need for sustainable solutions and how small actions, like switching packaging, can have a big impact.

Step 5: Practical Experience

Hands-On Learning:

Students learn by creating the MVP and testing it in the real world with actual businesses. They face challenges like manufacturing delays and unexpected costs but also gain insight into customer preferences and the need for continuous product iteration.

Peer Feedback:

The students present their pitch to their peers, receiving feedback about potential risks, such

as competition from larger packaging companies. They refine their presentation and value proposition based on the feedback they receive.

Assessment of Learning:

The students are evaluated on:

- Problem Identification: How well they identified a pressing environmental issue and understood the needs of their target customers.
- Business Solution: The strength of their solution in addressing the problem, its feasibility, and its value proposition.
- MVP Development: The effectiveness of their MVP in solving the problem while keeping it simple and scalable.
- Presentation: The clarity and persuasiveness of their pitch, the completeness of their business model, and the professionalism of their final presentation.

Conclusion:

By the end of the project, the students have not only developed a viable business solution but also learned important lessons about product development, market research, and the iterative process of refining a product based on user feedback. They now have the skills to identify problems, create innovative solutions, build an MVP, and present a business idea effectively.

Summary

- **Lean Start-Up:** Process of rapidly developing simple prototypes to test key assumptions with real customers.
- **Minimum Viable Product (MVP):** Simplest version of a product that enables sustainable business creation.
- **Dropbox:** A platform to share photos, documents, and videos across devices, using a video for initial testing.
- **User Testing:** Ensures the prototype evolves into a final product that meets user needs and enhances experience.
- **Feature Prioritization:** Means to Identify core features necessary to address primary problems or needs of target audience.
- **MPV Scalability:** Design the MVP for future scalability in architecture and technology.
- **User Onboarding:** Ensure users can easily understand and use the product effectively.
- **Targeted Audience:** Identify specific segments to focus initial marketing efforts.
- **Soft Launch:** Gather feedback from a smaller user base before broader release.
- **Beta Testing:** Invite select users to uncover potential issues and generate early reviews.
- **Marketing and Promotion:** Develop a plan to create awareness and drive user acquisition.
- **Monitoring and Analytics:** Implement tools to track user behavior and measure success.
- **Customer Support:** Provide responsive support to enhance user retention and referrals.
- **Feedback Mechanisms:** Use surveys and direct contact options for user feedback.
- **User Interviews:** Conduct interviews for deeper insights into experiences and pain points.
- **Analytics Tools:** Monitor user engagement and relevant metrics for data-driven decisions.
- **Engagement Metrics:** Track user retention rates and satisfaction indicators.
- **Continuous Improvement:** Refine products iteratively based on user feedback and data analysis.
- **Iterative Cycles:** Develop in short cycles for frequent updates.
- **Data-Driven Decision-Making:** Support improvements with evidence rather than assumptions.
- **User Acquisition:** Develop strategies for attracting and retaining users.
- **Performance Optimization:** Monitor product performance to maintain user-friendliness.
- **Market Expansion:** Consider related markets for broader reach.
- **Principles Application:** Apply learned principles in real-world projects and entrepreneurial ventures.

Exercise

Select the best answer for the following Multiple-Choice Questions (MCQs).

1. What is the primary goal of developing a Minimum Viable Product (MVP)?
   a. To include as many features as possible
   b. To create the most advanced version of the product
   c. To address the core problem or need of the target audience with the simplest version
   d. To achieve the highest possible user engagement immediately
2. Which strategy is NOT typically recommended when launching an MVP?
   a. Conducting a soft launch to gather initial feedback
   b. Expanding the product's features significantly before launch
   c. Using targeted marketing to reach the most relevant audience
   d. Inviting a select group of users for beta testing
3. What role does user feedback play in iterative development?
   a. It is used to maintain the product's original design without changes
   b. It helps in making decisions about scaling the product
   c. It drives continuous improvement and informs necessary changes
   d. It is collected but rarely used for making product enhancements
4. Which of the following is NOT a key aspect of continuous improvement in product development?
   a. Feedback-driven adjustments
   b. Long-term, static product design
   c. Iterative cycles or sprints
   d. Data-driven decision-making
5. What should be considered when scaling an MVP to a broader audience?
   a. Maintaining the product's simplicity and avoiding feature bloat
   b. Avoiding any changes to the MVP based on user feedback
   c. Investing heavily in new features without assessing user needs
   d. Ignoring performance optimization and focusing solely on marketing
6. What is a crucial method for collecting user feedback during the MVP phase?
   a. Ignoring user feedback and focusing on internal opinions
   b. Implementing feedback mechanisms like in-app surveys and direct contact options
   c. Conducting feedback sessions only after a year of product launch
   d. Using only external market research without user input
7. When assessing prototypes and MVPs, what criterion is used to evaluate how well the product addresses user needs?
   a. The complexity of the product's features
   b. The alignment with user needs and effectiveness in solving the problem
   c. The number of features included in the prototype
   d. The frequency of updates and releases

8. What is one of the primary benefits of conducting user testing for a prototype or MVP?
   a. It delays the product launch to add more features
   b. It ensures that the product remains static and unchanged
   c. It helps validate assumptions and gather feedback to refine the product
   d. It avoids the need for future iterations and improvements
9. In the context of product development, what is meant by "scalability"?
   a. The ability to add new features without affecting the existing ones
   b. The capability of the product's infrastructure to handle increased demand
   c. The process of reducing the product's features to simplify it
   d. The speed at which the product can be developed initially
10. What is the purpose of a soft launch in the MVP launch strategy?
   a. To make the product available to everyone immediately
   b. To gather feedback from a smaller user base before a broader release
   c. To launch the product with all features fully developed
   d. To avoid collecting user feedback altogether
11. Which of the following best illustrates the use of targeted marketing for an MVP?
   a. Launching the product worldwide without prior research
   b. Advertising to all age groups equally
   c. Focusing on new mothers for a postpartum fitness app
   d. Offering free subscriptions to random users
12. What is the role of analytics in MVP development?
   a. To create advertisements for the product
   b. To enhance product design without user input
   c. To track user behavior and make informed improvements
   d. To reduce the number of users on the platform

Give short answers to the following Short Response Questions (SRQs).

1. What is the primary purpose of a Minimum Viable Product (MVP) in the lean start-up process?
2. Name two key strategies for launching an MVP effectively.
3. How does user feedback contribute to the iterative development process?
4. What is one important consideration when scaling an MVP to a larger audience?
5. Describe one method for collecting user feedback during the MVP phase.
6. Why is simplicity emphasized when developing an MVP?
7. How can customer feedback be effectively collected and used during the MVP phase?

Content Authors

Mohammad Sajjad Heder has done Master (MS) in Computer Science from George Mason University, Virginia, USA and B.E. Mechanical Engineering from Kim Chaek University of Technology, Korea. He has 25 years' experience of teaching the subject of Computer Science to students of SSC and HSSC. He is a highly experienced author who has written many textbooks of Computer Science.

Lubna Kausar Janjua is an educationist and IT professional, with more than 20 years of experience. She received her MCS degree from IIUI and completed MS coursework from NUST with a specialization in Intelligent Information System. She worked as a software engineer for a multinational company and is currently serving as a Computer Science professor.

Shah Islam Khan, a distinguished author renowned for his groundbreaking 'IQ Series' books. His academic journey boasts a breadth of qualifications, from an MPhil in Higher Education to a range of degrees including BS(CS), DIT, MEd, and BEd. His diverse expertise enriches his literary contributions, promising readers a depth of knowledge and insight unparalleled in his field. Prof Shah Islam has been working in OPF BOYS COLLEGE ISLAMABAD for the last twenty years.

Mohammad Khalid, a luminary in the field of Computer Science and Assistant Professor at OPF Boys College. Professor Khalid's rich pedigree, boasting an M.Sc. in Computer Science and B.Ed. qualifications, infuses every page of these meticulously crafted textbooks for Grades IX to XII. Collaborating with the prestigious National Book Foundation (NBF), Professor Khalid's work epitomizes innovation, excellence, and pedagogical mastery. Dive into a realm of unparalleled insight, where theoretical depth converges with practical application, propelling students towards digital fluency and technological prowess.

Dr. Saleem Iqbal received both his BS and MS in Computer Science from the COMSATS, Pakistan and PhD from UTM, Malaysia in 2015. He has a diversified experience of over 18 years comprising academic, development, research and administrative. Currently, he is an Associate Professor and Chairman of Computer Science Department at Allama Iqbal Open University. Previously he was with PMAS-Arid Agriculture University Rawalpindi and COMSATS, Islamabad .

Dr. Faraz Ahsan is an Educational and IT Specialist with more than 20 years of experience of both academia and industry. He worked as a Software Engineer and Network Administrator in the industry. Later, he completed his PhD from COMSATS Institute of Information Technology and specialized in the field of Network Security. He is currently associated as Consultant with a USA based firm.

Dr. Mian Hamid Hassan is an Educational Specialist with more than 25 years of diversified experience in the field of Education specially teachers training and education. He did Masters in Technology Education from University of the Punjab with distinction and was awarded University Grants Commission's research scholarship for Ph.D. studies. Dr. Hassan is working as an Associate Professor at Federal College of Education and is involved in Teachers Training (In/ Pre - Service) especially in the Field of Technical, Elementary, Secondary and Science education.

Approved by National Curriculum Council, Secretariat
Ministry of Federal Education & Professional Training
vide letter No. English, Dated:

پاک سَر زمین شاد باد! کشورِ حسین شاد باد!
تو نِشانِ عزمِ عالی شان ارضِ پاکستان
مرکزِ یقین شاد باد!

پاک سر زمین کا نِظام قُوّتِ اُخوّتِ عوام
قوم ، مُلک ، سلطنت پائندہ تابندہ باد!
شاد باد منزلِ مُراد!

پرچَمِ سِتارہ و ہِلال رہبرِ ترقی و کمال
ترجُمانِ ماضی، شانِ حال جانِ اِستقبال
سایۂ خُدائے ذُوالجلال!

National Book Foundation
as
Federal Textbook Board
Islamabad