ECU-ITAI-1-03

Software and Programming Fundamentals

Information Technology and Artificial Intelligence

Grade One

International Applied Technology Schools (IATS)

Thanks and Appreciation

USAID Workforce Egypt Project in partnership with the Ministry of Education and Technical Education extends its sincere thanks and appreciation to everyone who participated in the preparation, authoring, reviewing, and printing the content of this book, including the authors and publishers.

The content of this book including the competencies and students' assessment guides has been prepared by a large group of experts specialized in the field of information technology and artificial intelligence.

The project would sincerely like to thank the participating companies for their great efforts and transferring their professional knowledge and expertise in developing the content of this book. Companies that participated in the development of the book include:

Ahmed Diefalla Group, UMAMI For Development, Appsinnovate, Secured Smart Systems 3S, Technospace, DM Arts Academy, SOfCO, Ideas Gym, Silver Key, AWS, Integrated Knowledge Dynamics, BEDO Company, and Discovery Academy, Deep-Iris AI.

This is in addition to other companies operating in the field of information technology and artificial intelligence that took part in the development of the competencies and students' assessment guides of the specialization.

Special gratitude goes to "Nahdet Misr Publishing House" for preparing, authoring, revising, and printing the book.

All intellectual copy property rights for printing, copying, and distributing of this book by electronic and mechanical means belong to USAID Workforce Egypt Project in partnership with the Ministry of Education and Technical Education.

Disclaimer:

"This Publication is made possible by the support of the American People through the United States Agency for International Development (USAID) and does not necessarily reflect the views of USAID or the United States Government."

Table of Contents

Subject	Page
Terms	7
Software Development Life Cycle (SDLC)	12
Introduction	12
(1) Planning	14
(2) Defining Requirements	15
(3) Designing the Architecture	20
(4) Building the System	23
(5) Testing the System	23
(6) Deployment and Maintenance	24
(7) SDLC Models	25
(8) Defining Input, Processing, and Output Modules	28
(9) Case Study	30
Software Standards	32
Introduction	32
(1) What Is Meant by Software Standards?	32
(2) The Unified Modeling Language (UML)	32
(3) Unified Modeling Language (UML) Types	33
(4) Class Diagram (Structural)	35
(5) Use Case Diagram (Behavioral)	40

Table of Contents

Subject	Page
(6) Sequence Diagram (Behavioral)	42
(7) Case Study	46
Programming Paradigms	48
Introduction	48
Structured Programming	49
Software Planning	52
(1) What Is a Flowchart?	52
(2) Flowchart Symbols	52
(3) Flowchart Structures	53
Developing First Software Project	68
Introduction	68
Introduction to Python	69
(1) What Is Python?	69
(2) Why Python?	69
(3) Setting Up a Python Environment	70
(4) Python Program Structure	71
(5) Comments	72
(6) Variables	73
(7) Output	74
(8) Escape Sequence	75

Table of Contents

(9) User Input	79
(10) Data Types	80
(11) Type Conversions	81
(12) Python Operators	83
(13) Conditional Statements	94
(14) Loops	101
(15) Object-Oriented Programming	105
(16) Functional Programming	111
(17) Lists	115
(18) Tuples and Dictionaries	119
(19) Case Study	123
Verifying That the Developed Software Meets Its Intended Purpose	164
Introduction	164
(1) Verifying That the Program Is Error-Free	164
(2) Verifying That the Program Performs the Required Task	173
(3) Altering the Program Based on Requirements	176
Final Assessment	184
References	190

Main Competence

Developing a simple program.

Competence Code: ECU-ITAI-1-03

Learning Outcomes

Upon completion of this course, students will be able to:

1. Prepare a plan for simple software that meets specific tasks.

- Plan to meet certain specifications within a specified period of time for the tasks assigned to them.
- Define a design as the basis for the work of the code that will be implemented.
- Submit a report on the plan and a design in such a way that it can be consulted again.

2. Write code for the previously designed software program.

- Write code according to the design prepared.
- Address software issues and errors that appear during the development process.
- Use software language properties during the development process.
- Abide by the rules of naming conventions in accordance with its organization.
- Add sufficient comments to each part of the code to clarify its purpose.

3. Ensure the usage of source control, for example (GIT).

- Create storage space for the project (Source Code Repository).
- Copy the project from the cloud (Clone).
- Document each stage of the project (Commit).
- Pull other updates from the cloud (Pull).
- Upload the latest update on the cloud (Push).

4. Verify that the developed software meets its intended purpose.

- Test the program to make sure it is free of software errors.
- Test the program to ensure that its purpose is achieved.
- Adjust the program as needed.

TERMS

No.	Terms	Description
1	**Software Development Life Cycle (SDLC)**	SDLC is a process of sequential activities in a software development project to design, develop, and test high-quality software .
2	**Waterfall Model**	It is the classical method of software development techniques that depicts the software development process in a linear, sequential flow.
3	**Unified Modeling Language (UML)**	UML is a standard language of communication among software engineers throughout the world.
4	**Class Diagram**	It is one of the most popular UML diagrams that shows a static representation of the project, covering all relationships and functionalities over methods. It uses the concept of classes, in which each class represents an object within the system.
5	**Composite Structure Diagram**	It provides a logical representation of internal components for each part of the system.
6	**Object Diagram**	It shows a complete view of any object from the project at a specific time. The viewing process includes the structure of the object, the data, and the relations.
7	**Package Diagram**	It is considered a simple image of the class diagram, which could collect each group of classes into a package.
8	**Sequence Diagram**	It shows how each object in the project interacts with the other objects in the system in order.
9	**State Machine Diagram**	It refers to the action of an object in the same event, depending on the object's state.
10	**Interactions Overview Diagram**	It has a high level of interaction diagrams and is capable of showing complex interactions via frames.

11	**Activity Diagram**	It shows the flow control from the starting point to the end for all possible paths.
12	**Timing Diagram**	It focuses on the events that change during the lifeline time.
13	**Use Case Diagram (Behavioral)**	It is one of some UML diagrams that clarify the dynamicity of the project. Unlike the class diagram, in addition to this, it is characterized by explaining the probable user interactions, which may be internal or external requirements for the system.
14	**Sequence Diagram (Behavioral)**	It is one of the behavioral UML diagrams. It shows the communication between the system and the users, or between the system items.
15	**Imperative Paradigm**	It is a paradigm that makes the programmer teach the machine how to alter its condition.
16	**Declarative Paradigm**	It is a paradigm that makes the programmer just declare the properties of the program without explicitly computing them.
17	**Object-Oriented Programming (OOP)**	OOP is a programming paradigm that relies on object concepts. Each object comprises data and code.
18	**Object**	An object in the OOP is a thing that has features and functionality. In other words, the object is the stone that is used in building an application.
19	**Class**	The class in the OOP is a user-defined type that represents an object. A class description consists of attributes and methods of objects.
20	**Inheritance**	Inheritance features mean that a class can use code from another one. Inheritance in programming is very likely to be inherited in real life.
21	**Polymorphism**	It is one of the fundamental concepts in the OOP. Polymorphism describes a situation in which different classes can perform the same function in different ways.

22	**Abstraction**	Abstraction in the OOP means that objects share only necessary data and functionality with the rest of the code.
23	**Encapsulation**	Encapsulation in the OOP means that all the information is contained in an object, and only a set of information can be shared with the rest of the code.
24	**Functional Programming**	It is a programming paradigm where programs are developed by using and organizing functions.
25	**Flowchart**	It is a diagram that outlines the steps in a process. It began as a tool for describing algorithms and programming logic in computer science but has since expanded to include all types of processes.
26	**Comments**	They are human-readable texts. The compiler is ignoring any comments during the execution of the code.
27	**Variables**	A variable is the name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.
28	**Arithmetic Operators**	These operators are responsible for any arithmetic operations like addition, subtraction, multiplication, division, and so on.
29	**Assignment Operators**	The assignment operators are responsible for assigning values to the left-hand side of these operators.
30	**If-else Statements**	If Statement is used to check some given conditions and perform some operations depending on the correctness of those conditions.
31	**Working Directory**	It is the place of the local version of the source code, which may be our personal computer (PC) or laptop. Any changes in this state don't reflect on the main repository source code.
32	**Staging Area**	It is an intermediate state between the working directory (the local copy) and the repository (the main remote shared copy).
33	**Repository**	It is the local place of the source code that has the committed changes.

34	**Git Software Tool**	Git is a free and open-source distributed version control system (DVCS). Version control is a way to save changes over time without overwriting previous versions.
35	**Syntax Errors**	They are errors in programming language rules. Syntax errors can be detected and uncovered by a compiler or interpreter.
36	**Logic or Meaning Errors**	They are errors that indicate the logic used when coding the program failed to solve the problem.
37	**Runtime Errors (Exceptions)**	They happen when the code does something illegal when it is run, like dividing by zero. To verify that the program is syntax error-free, it should be compiled with zero syntax errors.

1st Learning Outcome

Preparing a plan for simple software that meets specific tasks.

Evidence and Proof Requirements

After completing this section, students should be able to:

- Plan to meet certain specifications within a specified period of time for the tasks assigned to them.
- Define a design as the basis for the work of the code that will be implemented.
- Submit a report on the plan and a design in such a way that it can be consulted again.

Software Development Life Cycle (SDLC)

Introduction

Let's imagine that you need to build a project for the nearest supermarket to your home. It's not an easy task to start from because you have multiple subsystems working together to build up the whole (overall) system. Let's talk through them, any supermarket even if it was a small or a big one with a brand name consisting of many entities including inventory, suppliers, customers, products, sale/purchase receipts, workers, etc. Every entity has its own functions which facilitate the task of another entity. Combining the tasks of the related entities is building up subsystems within the overall system. For example, the supermarket is with inventory control, checkout, etc., so before designing or analyzing any system, have a clear view of the whole system first and then focus on some specific area. Remember that you must be specific. So, before designing or planning any project, put yourself in the shoes of your teammates. Collect information about what this change relates to, what prompted it, what has already been done, what is left to do, and any specific asks for help or reviews. Include links to relevant work or conversations. So, we will discuss in this outcome how to plan to start applying a software program about learning the software development life cycle (SDLC). What is meant by SDLC in the software world?

What is the Software Development Life Cycle?

Software Development Life Cycle is known as SDLC, and it is a set of business rules followed to design software systems. Following the software development, life cycle approaches guarantee meeting the highest software quality and customer expectations within time and cost boundaries. SDLC comprises six main stages: Planning, Defining, Designing, Building, Testing, and Deployment as we see in figure (1):

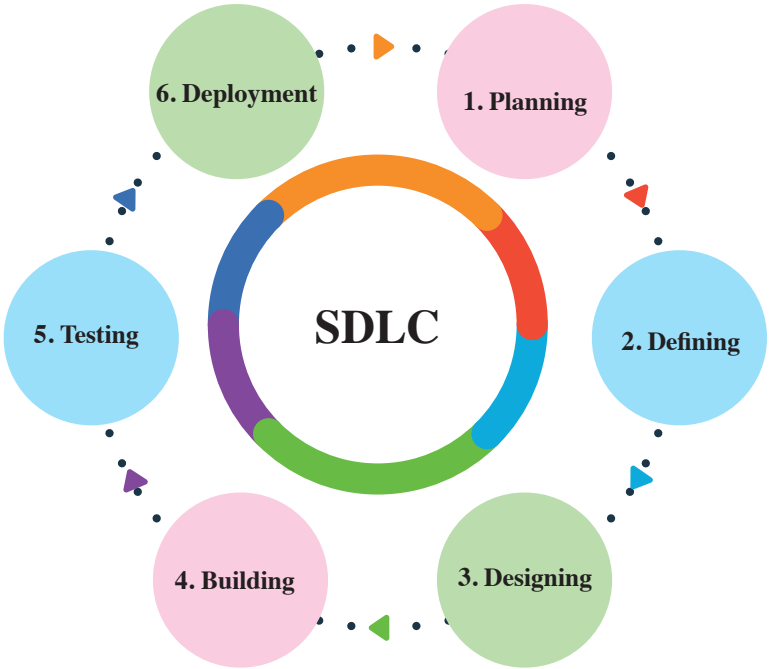


Figure (1): Software development life cycle phases

The standard software development life cycle consists of six main stages. Next, we will explain each one of them briefly.

1. Planning.
2. Defining requirements.
3. Designing the architecture.
4. Building the system.
5. Testing the system.
6. Deployment and maintenance.

Exercise (1)

Simulate with your group a meeting with a customer who is willing to develop a website for selling dairy products. The website is related to the dairy products factory. The website is also related to the application where the users can order their needs.

1 Planning

The planning and requirements analysis phases are very important steps in the software development life cycle. This step is usually done by the project managers and team leaders. This step is usually done in a form of a meeting that gathers all the teams in the company including the development team, marketing team, sales team, and operational team. The collected information from this meeting is used to form the plan of project flow. The planning phase also includes deciding the required count of team members, defining the team structure, calculating costs, and making a timetable with a sub-project delivery plan.

The planning phase includes several vital submodules:

1. Defining objectives and goals.
2. Defining deliverables and approvals.
3. Defining roles.
4. Defining tasks and activities.
5. Defining the timeline of the whole project.

Note (1)

The planning and requirements analysis phase answers the question of what the problem is and what the customer needs to solve it.

Example

Assuming that a customer is willing to develop a website connected to an application. This task requires a front-end and back-end developer or team based on the scale.

Exercise (2)

1. What is the planning phase in SDLC?
2. What are the submodules included in the planning phase?

Exercise (3)

In reference to Exercise (1) on Page 13, perform the planning phase with your colleagues in your group.

2 Defining Requirements

In the first line: Defining the requirements phase is sometimes considered a part of the planning and requirement analysis phase. In this phase, the requirements are explicitly documented and approved by the customer. Usually, the software requirement specification (SRS) form is used and signed by the project manager and the customer as the approved list of requirements. Defining the requirements phase also includes documenting the required tools, labor, and sometimes components for developing the required solution. The most common form of SRS is shown in table (1).

Example

If we build a hospital system, having a hospital should be documented as a requirement.

Requirements are classified into functional and non-functional.

Functional requirements: these are the requirements that the user specifically needs as features in the system. All these functionalities need to be part of the system. These are represented or stated in the form of input given to the system, the operation performed, and the expected output. They are the requirements stated by the user, which one can see directly in the final product, unlike the non-functional requirements.

Example

If the customer requires the user to be able to log into the system, this feature is considered to be a functional requirement.

Exercise (4)

What is meant by functional requirements?

Non-functional requirements: these are the rules or restrictions that the system satisfies. Non-functional requirements are also known as non-behavioral requirements. They deal with issues such as:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Example

If the customer requires the user to be able to request a feature, the maximum waiting between the request and the response is considered to be a non-functional requirement.

Table (1): SRS simplified form

Introduction	
Purpose	.....
Product scope	.....
Intended audience	.....
Intended use	.....
Definitions and acronyms	.....
System Requirements	
Functional requirements	
Non-functional requirements	
System features	
External Interface Requirements	
User interface	
Hardware interface	
Communication interface	

Exercise (5)

What is meant by non-functional requirements, and what are the issues that deal with them?

Example

Assuming that a customer is willing to develop a company productivity tracking system. The system should be able to track the productivity and the attendance of the employees. It also should be able to analyze the time that an employee wastes during the day. The system should also track suspicious behaviors in the company and give an early alarm in case of an emergency. Based on the given description, fill in the following SRS form.

Table (2): Suggested form

Introduction	
Purpose	○ Track the productivity and security of a company.
Product scope	○ Attendance tracking. ○ Productivity tracking. ○ Behavior tracking.
Intended audience	○ Security and productivity tracking. ○ Weekly, monthly, and annual reporting to support decision making.
Definitions and acronyms	○ **Tracking:** monitoring, recording, analyzing, and reporting aimed phenomena. ○ **Attendance tracking:** monitoring, recording, analyzing, and reporting the entering, leaving, and midday leaving time. ○ **Productivity tracking:** monitoring, recording, analyzing, and reporting the behavior of the employee during the day in terms of the amount of time wasted, not in the working area, the amount of time working, and other behaviors. ○ **Suspicious behavior:** it is the existence of an employee in a place that should not be. For example, the existence of an employee in the server or network room. ○ **Tracking suspicious behavior:** monitoring, recording, analyzing, and reporting the behavior of an employee that should not happen. For example, reporting how many times an employee is trying to enter a server room when he should not and there is no need to be around the server room. ○ **Early alarm:** Giving an early indication in the form of an email to the direct manager of the employee by the suspicious behavior detected was supported with visual evidence.

System Requirements	
Functional requirements	O Employee - computer - track O Attendance - track O Productivity - daily - track O Productivity - weekly - track O Productivity - monthly - track O Productivity - annual - track O Detect - suspicious - behavior O Report - suspicious - behavior O Give - an alarm
Non-functional requirements	O The system should be fast: each tracking function should respond in less than 1 second. O The system should be able to handle around 5000 employees. O The system should be simple and easy to use.

External Interface Requirements	
User interface	O The system is expected to have a main page for the authorized person login. O The system is expected to have a page for each employee with his historical data. O The system should present the data to the authorized person only.
Hardware interface	O The system should be interfaced with the required cameras in the company.
Communication interface	O The system should be able to connect all departments together. O The system should store the recorded data on the server. O The system should have 3 backups of the data on the cloud.

Next are the main differences between functional and non-functional requirements as shown in table (3).

Table (3): Functional vs. non-functional requirements

Functional Requirements	Non-functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?"
It is specified by the user.	It is specified by technical people e.g. architects, technical leaders, and software developers.
It is mandatory.	It is not mandatory.
It is captured in the use case.	It is captured as a quality attribute.
It is defined at a component level.	It is applied to a system as a whole.
It helps you verify the functionality of the software.	It helps you to verify the performance of the software.
Functional Testing like Systems, Integration, End to End, API Testing, etc. is done.	Non-functional Testing like Performance, Stress, Usability, Security Testing, etc. is done.
It is usually easy to define.	It is usually more difficult to define.
Examples: 1. Authentication of the user whenever he/she logs into the system. 2. System shutdown in case of a cyber-attack. 3. A verification email is sent to the user whenever he/she registers for the first time on some software systems.	Examples: 1. Emails should be sent with a latency of no greater than 12 hours from such an activity. 2. The processing of each request should be done within 10 seconds. 3. The site should load in 3 seconds when the number of simultaneous users is $>$ 10000.

Note (2)

Functional requirements are represented or stated in the form of input to be given to the system, the operation performed, and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Exercise (6)

1. Differentiate between functional and non-functional requirements.
2. A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs. Based on the previous details, list some functional and non-functional requirements for the system.
3. Fill in the SRS form shown in table (1).

3 Designing the Architecture

Based on the documented list of requirements in the defining requirement phase, a design architecture or more is proposed by the development team. The proposed architecture comprises several components like architecture, user interface, platforms, programming, communication techniques, and the required security. Designing the architecture phase includes defining all the internal modules of the system which should be documented in the design document specification (DDS). Next, a brief description of the modules covered in the architecture designing phase will be listed:

a Architecture

The architecture defines the whole system with its components and the relationships among these components. The system architecture can be visualized by an architecture diagram. An architectural diagram is a diagram that abstracts the overall software, the system, the relationships, and constraints as shown in figure (2). It is a vital step as it affords an overall view of the software system and its roadmap as shown in figure (3).

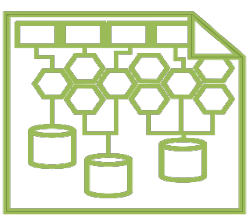


Figure (2): System architecture

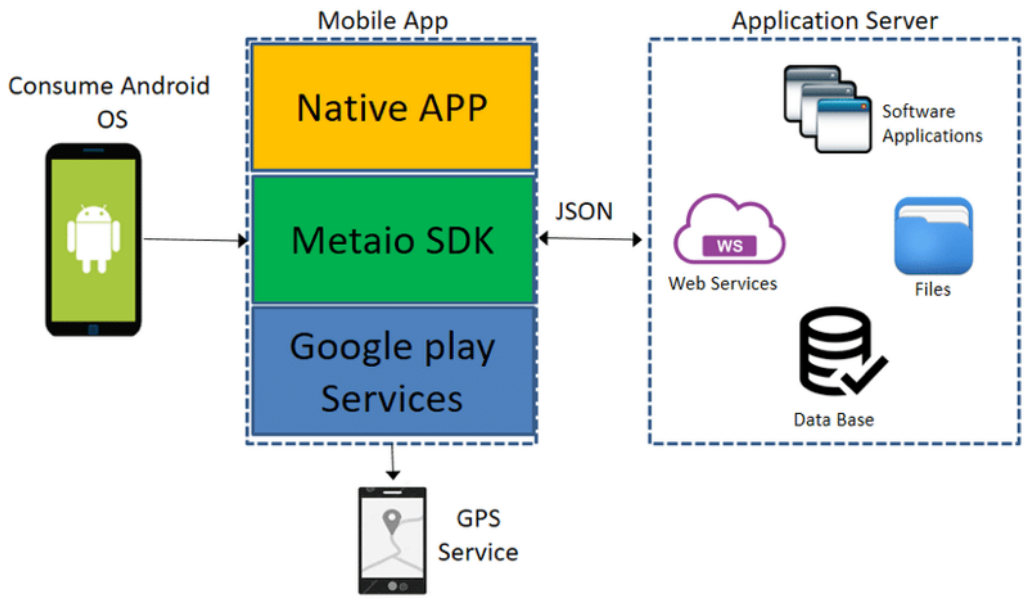


Figure (3): System architecture sample

Note (3)

1. Designing the system architecture is usually visualized in the system architecture diagram.
2. The system architecture diagram visualizes the structure relationship and the views of the system.

Exercise (7)

The requirements are listed in ..... form while the system architecture is documented in ..... form. (Fill in the blanks.)

b User Interface

The user interface defines the way that the user communicates with the system and how the system into acts in turn with the user as shown in figure (4).

Figure (4): User interface

c Platforms

Platforms define the tools that the system will be used on, for example, whether the system will run on personal computers or mobile phones and which type of each of them as shown in figure (5).

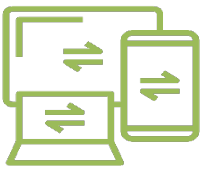


Figure (5): Platforms

Example

Booking a doctor's appointment should run on any platform. However, retrieving the number of available medicines should be run on hospitals' computers only.

d Programming

Programming defines the programming paradigms of the system. It is not only the programming language but also, the used technique in programming and solving the problem.

Figure (6): Programming

e Communication Techniques

Communication techniques define the protocols needed for the system modules to communicate with each other and the way that the system will communicate with the data center (if needed).

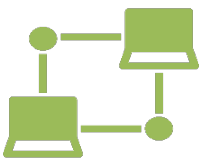


Figure (7): Communication techniques

f Security

Security defines the security techniques needed to protect an application like encryption techniques or protection of the data centers.

Figure (8): Security

Note (4)

Systems should be developed securely and resistant to attacks.

Exercise (8)

1. List and explain the modules covered in the architecture designing phase.
2. A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs. Based on the previous details, and functional and non-functional requirements gathered, discuss the submodules of designing the architecture with your group.

4 Building the System

In this stage, the real development of the system begins. The requirement gathered and defined in the previous stages will be implemented based on the planned specifications. Small systems campaign is implemented using a single developer. However, in large projects, a development team cooperates to achieve the goal. Access control and source code management techniques are used to control the way that the development team cooperates. The development team is also responsible for documenting the code they wrote. In some cases, the development team is responsible for testing some modules of the system.

5 Testing the System

Testing the system is a critical step before deploying the system to users. The testing phase usually is associated with the testing team. The testing team is responsible for trying all the modules of the system, this thing is the overall system, reporting defects, and making sure that the system meets the defined requirements. In case the system is very large, the testing phase can be automated.

Note (5)

The testing phase aims to minimize the number of failures in the system. That leads to customer satisfaction and higher development quality.

Exercise (9)

1. Why is testing important in system development?
2. A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs. Based on the previous details, functional and non-functional requirements, and the submodules of designing the architecture, discuss the testing procedures expected to be performed with your group.

6 Deployment and Maintenance

Once the system is tested and ensured that there are no failures and the requirements are met, it can be deployed for the users. System deployment can be simple and done in a single stage like application upload on the App Store, or it can be complex and done in stages like updating distributed system database. Once the system is deployed, feedback can be received from the users and then maintenance can be done on the system. Maintenance means handling shortages did not cover through the testing phase, or handling cases were not taken into consideration in the development stage. The deployment and maintenance phase can be done several times based on the user's experience feedback.

Note (6)

The deployment stage is directly associated development, testing, and maintenance stages.

Example

1. Deployment of a hospital management system is done by installing the software on hospital computers.
2. Deployment of clinic reservation application is done by uploading it.

Exercise (10)

What is meant by system deployment?

7 SDLC Models

The Software Development Life Cycle has different models that follow the previous stages. These models are known as software development approaches models which follow a set of steps that ensures success in each process of software development. Now we show some of the most popular software development life cycle models.

- Iterative model.
- Waterfall model.
- Agile model.

Exercise (11)

List three different types of SDLC models.

a Iterative Model

The iterative model starts with a simple implementation of a set of modules of the system and iteratively improves these subsets until the system completes. The core idea of the iterative model is to develop the system through repeated iterations in an incremental manner as shown in figure (9). In an iterative model, one or more iterations of software development can take place simultaneously. The whole system is divided into sets of modules, and each module goes through the requirement design implementation, and testing phases. The iterative model enables sub-teams to work simultaneously which minimizes the development time of the system. However, on the other hand, it requires more resources to enable the teams to work simultaneously.

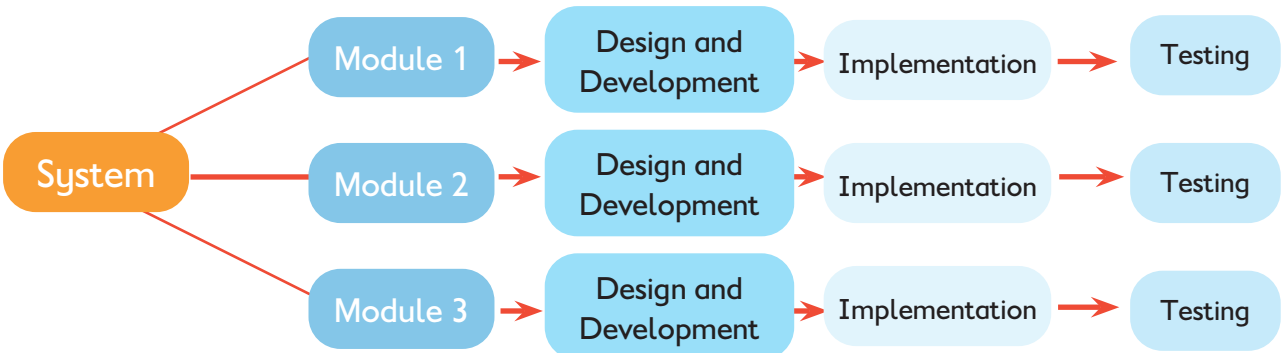


Figure (9): Iterative model

Exercise (12)

A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs.
Discuss the concept of the iterative model in the previous scenario with your group.

b Waterfall Model

The waterfall model is the classical method of software development technique.
In the waterfall model, the whole project is divided into separate phases.
Each phase should be finished before moving into the next phase as shown in figure (10).
The waterfall model is a sequential technique for software development. The benefit of the waterfall model is that it is simple and easy to use and manage. However, on the other hand, it's a very poor model for long and huge projects.

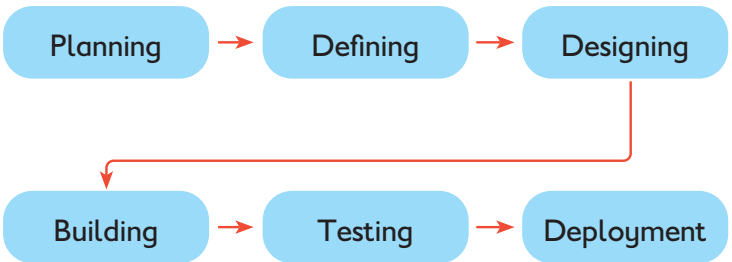


Figure (10): Waterfall model

Note (7)

1. The waterfall model is usually utilized when the requirements are very well documented clear and fixed.
2. The waterfall model is very difficult to move back to any phase; accordingly, it cannot handle requirements changing.

Exercise (13)

1. The waterfall model is used with dynamic and variable requirements. (True or False)
2. A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs.

Discuss the concept of the waterfall model in the previous scenario with your group.

c Agile Model

The agile model aims to divide the project into cycles that are called sometimes milestones. Each milestone has a specific duration with expected delivery. Each milestone in the agile model goes through the first five phases of the software development life cycle which are planning, defining, designing, building, and testing as shown in figure (11). One of the main characteristics of the agile model that is deliver working product very quickly. However, one of the main drawbacks of the giant model is to heavily include customer interaction which sometimes leads to technical contradictions.

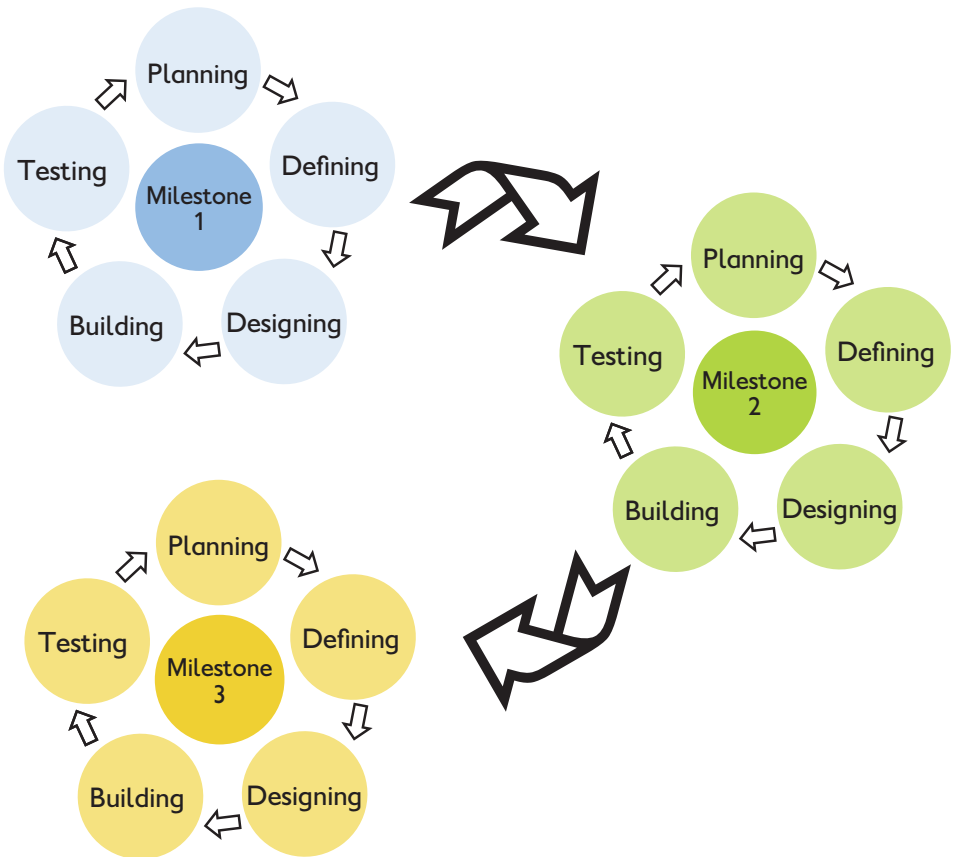


Figure (11): Agile model

Exercise (14)

A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs.

Discuss the concept of the agile model in the previous scenario with your group.

8 Defining Input, Processing, and Output Modules

To define input, processing, and output modules, you need to carefully read the problem description and analyze these modules.

Note (8)

Some programs may not include an input module.

Example

Define the input, processing, and output modules of the following problem:
Write an algorithm of a program that reads a number from the user and states whether it is a positive, negative, or zero number.

Solution:

Input: a number from the user.
Processing: the process of stating whether it is a positive, negative, or zero number.
Output: a statement of whether it is a positive, negative, or zero number.

Example

Define the input, processing, and output modules of the following problem:
Write an algorithm of a program that reads a number from the user and states whether it is an even or odd number.

Solution:

Input: a number from the user.
Processing: the process of stating whether it is an even or odd number.
Output: a statement of whether it is an even or odd number.

Example

Define the input, processing, and output modules of the following problem:
Write an algorithm of a program that prints even numbers between 0 and 100

Solution:

Input: none.
Processing: the process of stating whether the number is even or not.
Output: even numbers between 0 and 100

Example

Define the input, processing, and output modules of the following problem:
Write an algorithm of a program that reads three numbers and prints the maximum number among them.

Solution:

Input: three numbers.
Processing: the process of stating whether the number is even or not.
Output: the maximum number among the three numbers.

Exercise (15)

Define the input, processing, and output modules of the following problems:

1. Write an algorithm for a calculator that works on integer numbers. The user enters two numbers to perform only one of four basic arithmetic operations (+, -, *, and /).

The interaction with the user might look like this:

Enter your expression: 2 + 1
The result is 3

Hint: The user is allowed to do only one operation at a time.

2. Write an algorithm for a program that converts seconds to equivalent hours, minutes, and seconds.
3. Write a program to swap the values of two integers using a third variable.
4. Write an algorithm for computing factorial N (N!) Where N! = 3*2*1*. N.
5. Write an algorithm to find the sum of the first 50 natural numbers.
6. Write an algorithm for a program that reads N numbers. It displays the sum of even numbers only.
7. Write an algorithm that prompts the user to enter two positive integers and finds their greatest common divisor.
8. Write an algorithm to read the grades of 10 students and display the top student.
9. Write an algorithm for the electricity company, which charges customers according to their usage rank (1, 2, or 3) and reading (watt) if:
   - In rank 1, customers pay L.E. 1/100 watts with a minimum of L.E. 10
   - In rank 2, customers L.E. 2/100 watts but pay with a maximum of L.E. 400
   - In rank 3, customers pay L.E. 2.5/100 watts.

The program reads the usage rank and reading and displays the charged amount.

9 Case Study

A Temperature Conversion Project is a software system that is responsible for helping people in knowing the air temperature. The air temperature is measured in two units, either Celsius (C) or Fahrenheit (F). Some countries depend on the (F) when measuring the air temperature like the United States and other countries depend on the (C) like Egypt. The Celsius (C) and Fahrenheit (F) are not equal in value. So, this software system will help you if you left your country. You could know the air temperature regardless of the new location. This could occur by entering the temperature of the new country and asking for the temperature of your country.

Note (9)

Writing the code is the final step.

To start developing any software system like this, you should follow some rules:

1. Read the case study in detail to understand the requirements.
2. List the required functionalities of the software system.
3. Define the inputs and outputs of the system.

CELSIUS ⇔ FAHRENHEIT

C ⇨ F	F ⇨ C
$F = \frac{9}{5}(C) + 32)$	$C = \frac{5}{9}(F - 32)$

4. Draw the time plan.
5. Dividing the project into milestones depends on the required functionality.
6. Document all previous steps besides defining the programming language which you use.
7. Finally, start developing the software system by taking into account applying all previous steps. (Writing the Code).

Assessment

Q 1. What is the software development life cycle?

Q 2. What are the main phases of SDLC?

Q 3. What are the submodules in the planning phase of SDLC?

Q 4. Requirements can be classified into ..... and .....

Q 5. What is meant by functional requirements?

Q 6. What is meant by non-functional requirements?

Q 7. Differentiate between functional and non-functional requirements.

Q 8. State the input, processing, and output modules, for a program that reads a number and determines whether it is positive, negative, or zero.

Q 9. State the input, processing, and output modules, for a program that checks an input number within a specified range. For example, if the user enters range (20, 50) and queried number 34, the program should display "In range". On the other hand, if he/she enters 76, the program should display "Not in range".

Q 10. State the input, processing, and output modules, for a program that determines whether a baby's weight is normal or not. For girls, the normal weights are 2.5 to 4.5 kg. On the other hand, for boys, the normal weights are 4 to 5.5 kg.

Q 11. State the input, processing, and output modules, for the bank withdrawal process. First, the program reads the customer balance and withdrawal amount. The transaction is completed in case the balance covers the required amount. If not, display the message "Insufficient funds". Finally, display the customer balance.

Q 12. State the input, processing, and output modules, for an algorithm that represents a program that calculates the absolute difference between two numbers.

Q 13. State the input, processing, and output modules, for an algorithm that represents a program that calculates the net salary of an employee based on his/her job: 'M' for a manager and 'E' for an employee.

- The salary of the manager = salary + bonus
- The salary of the employee = salary – tax

Software Standards

Introduction

Since the beginning of the era of the software industry in the world, fixed rules and protocols should have been established for unifying the ways of using the software either among the users or among the software developers. The software developers thought about how there is a way to facilitate communication with each other. Accordingly, several standard protocols were unified to ease communication among software development teams.

1 What Is Meant by Software Standards?

James Rumbaugh, Grady Booch, and Ivar Jacobson created a language to be the standard language among software engineers. It was a simple visualization-based language, easy to understand, and applicable to be shared. This language had become the official method to document software systems. It had been called the Unified Modeling Language (UML).

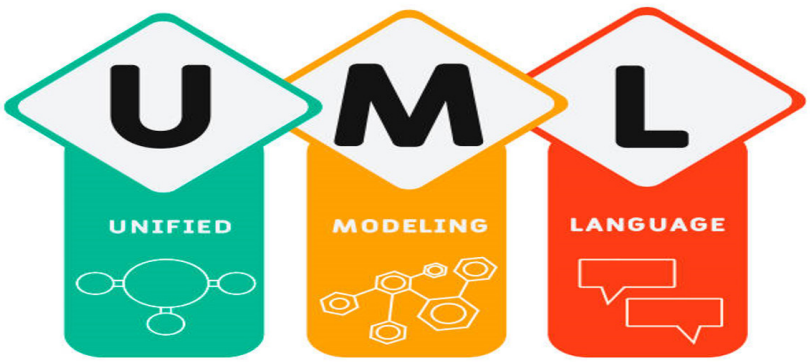


Figure (12): Unified Modeling Language (UML)

2 The Unified Modeling Language (UML)

The UML stands for the Unified Modeling Language which means a standard language of communication among software engineers throughout the world. It is a set of documents that are constructed based on drawing or visual perception. The UML has two different types of diagrams: either structural-based or behavioral-based. It is all about 13 different diagrams. In the following paragraph, we will discuss each type with its diagrams.

3 Unified Modeling Language (UML) Types

The UML types are structural or behavioral. The structural type contains 6 different diagrams and the behavioral type contains 7 different types.

a UML Structural Diagrams

These types of diagrams focus on describing the software system project by displaying the different components of this project such as inputs, building blocks, and outputs. This type of diagram contains 6 different types; each with its characteristics:

- Class Diagram.
- Composite Structure Diagram.
- Object Diagram.
- Component Diagram.
- Deployment Diagram.
- Package Diagram.

In the following section, you could look at the definition for each type of these diagrams:

Class Diagram:

It is one of the most popular UML diagrams that shows a static representation of the project for covering all relationships and functionalities over methods. It uses the concept of classes in which each class represents an object within the system. For example, in a factory, we can assume the product A is a class. Also, customers can be assumed as a class.

Component Diagram:

It is like a class diagram about the executable aspects of component-based projects. The component diagram is responsible for visualizing the system and relationships among modules within the system.

Composite Structure Diagram:

It provides a logical representation of internal components for each part in the system.

Deployment Diagram:

It shows the relationship between the software and the hardware components of the project.

Object Diagram:

It shows a complete view of any object from the project at a specific time. The viewing process includes the structure of the object, the data, and the relationships.

Package Diagram:

It is considered a simple image of the class diagram which could collect each group of classes into a package.

Exercise (16)

1. List some of the structural UML diagrams and mention their core usages.
2. Why are the structural diagrams used?

b UML Behavioral Diagrams

These types of diagrams show the interactions between the different objects in the software program. This may depend on the time. This type contains 7 diagrams:

- Use Case Diagram.
- Sequence Diagram.
- State Machine Diagram.
- Interactions Overview Diagram.
- Activity Diagram.
- Timing Diagram.
- Communication Diagram.

In the following section, you could look at the definition for each type of these diagrams:

Use Case Diagram:

It shows a visualized and dynamic representation of each functionality in the project.

Sequence Diagram:

It shows how each object in the project interacts with the other objects in the system in order.

State Machine Diagram:

It refers to the action of an object of the same event depending on the object state.

Interactions Overview Diagram:

It is a high level of interaction diagrams and is capable of showing complex interactions via frames.

Activity Diagram:

It shows the flow control from starting point to the end for all possible paths.

Timing Diagram:

It focuses on the events that change during the lifeline time.

Communication Diagram:

It is similar to the sequence diagram which monitors the interactions between the objects regarding the sent and the received messages.

After we mentioned the definition of each type from the UML different types, let us explain the most common diagrams of them in more detail.

In the next sections, we will study the class diagram, use case diagram, and sequence diagram.

Exercise (17)

1. List some of the behavioral UML diagrams and mention their core usages.
2. Why are the behavioral diagrams used?

4 Class Diagram (Structural)

The class diagram is considered one of the simplest and most popular UML diagrams. Its simplicity was coming from the ease of drawing and understanding. The class diagram could make you think about the software code as an object-oriented paradigm.

In the next few points, we will know more about the class diagram:

a Why Do We Use the Class Diagram?

1. It is simple.
2. It is a static representation of the software features (functions).
3. It is a visualized diagram.
4. It represents the relationship between the different objects in the project.
5. It lets you write your own software code functions.
6. It lets you identify the functionality of the software project.

b Class Diagram Components

1. Class:

It represents the different types of objects in the project as shown in figure (13).

Note (10)

Each type of object in the software project is a class.

2. Attribute:

It represents the information of each class as shown in figure (13).

3. Methods:

They represent the functionalities of the class that it is shown in figure (13).

4. Relationships:

They represent the communications between the classes as shown in figure (14).

A class diagram has several relationship types such as association, directed association, reflexive association, multiplicity, aggregation, composition, inheritance/ generalization, and realization as shown in figure (14).

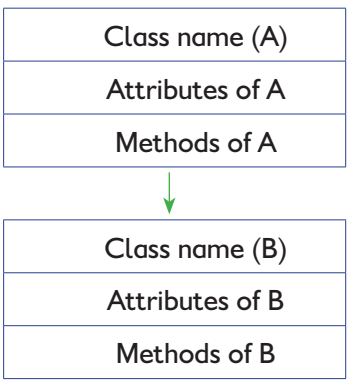


Figure (13): Class diagram

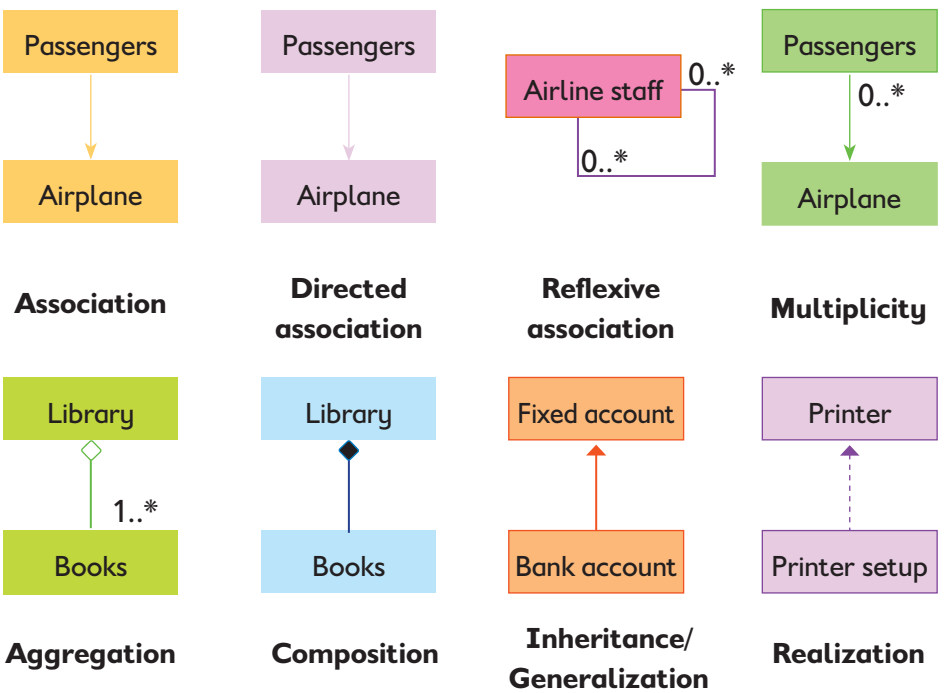


Figure (14): Class diagram relationship types

Note (11)

Each class in the class diagram should contain attributes and methods for this class.

Association:

It is a relationship that refers to any logical connection or relationship between classes as shown in figure (15). For example, passengers and airlines may be linked as above.

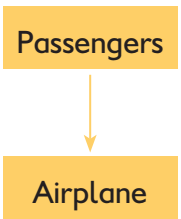


Figure (15): Association relationship

Directed association:

It is a directional relationship that usually represents a container-contained relationship as shown in figure (16). For example, an airplane (container) contains passengers.

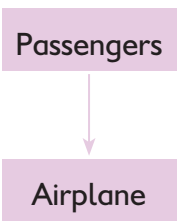


Figure (16): Directed association relationship

Reflexive association:

It is a relationship between a class to itself. In other words, a reflexive association is an association between instances of the same class as shown in figure (17). A reflexive association relationship is used when a class has several responsibilities. For example, an airplane staff member can be a pilot, ticket writer, security, flight engineer, or maintenance crew member.

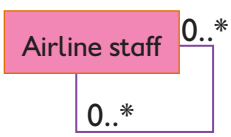


Figure (17): Reflexive association relationship

Exercise (18)

With your colleagues, give an example of the following relationships:

1. Association. 2. Direct association. 3. Reflexive association.

Multiplicity:

It is a relationship that defines the number of participating objects of each class in the relationship as shown in figure (18). In other words, it defines how many objects of one class are related to the instance of the second class. For example, one airplane may contain zero to many passengers. The notation 0..* in the diagram means "zero to many".

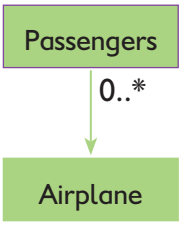


Figure (18): Multiplicity association relationship

Aggregation:

It is a relationship that defines how a class is defined by aggregating other objects of another class as shown in figure (19). For example, the class "library" is made up of one or more books. In aggregation, the objects aggregated to define a class are not strongly dependent on the lifecycle of the container class. In the same example, books will remain so even when the library is dissolved.

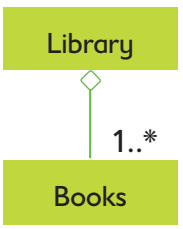


Figure (19): Aggregation relationship

Composition:

Is a relationship that represents an all-part relationship. A composition relationship is a form of aggregation relationship; however, it emphasizes the dependence of the contained class on the container class as shown in figure (20). For example, the library's existence is fully dependent on books existence.

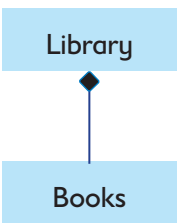


Figure (20): Composition relationship

Exercise (19)

With your colleagues, give an example of the following relationships:

1. Multiplicity.
2. Aggregation.
3. Composition.

Inheritance/Generalization:

It is a relationship that represents the concept of inheritance, in which one class is a child of another class as shown in figure (21). In other words, the child class is a class that has all the features of the parent class with some new features. For example, a bank account is a special type of fixed account with other features.

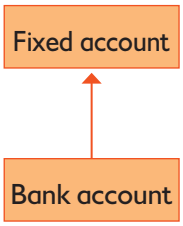


Figure (21): Inheritance relationship

Realization:

It is a relationship that represents the implementation of the functionality of a class in another class as shown in figure (22). For example, the printing setup function is implemented by the printer.

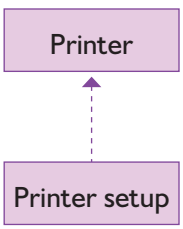


Figure (22): Realization relationship

Exercise (20)

With your colleagues, give an example of the following relationships:

1. Inheritance.
2. Realization.

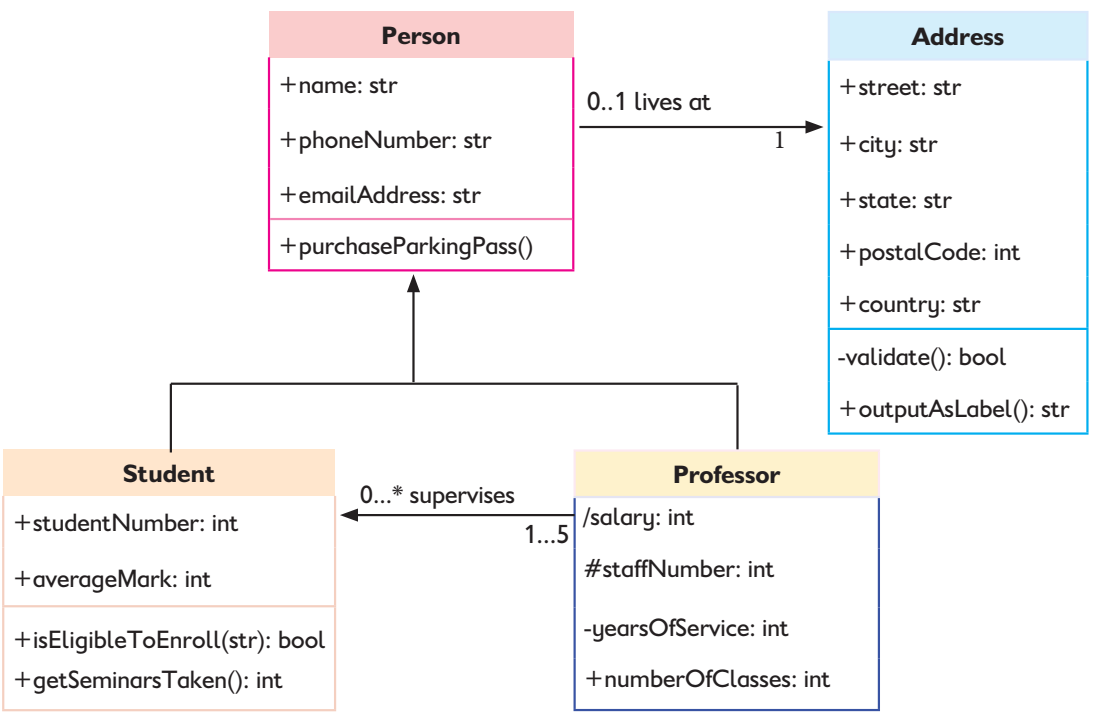


Figure (23): Class diagram example

Example

If a system has customers, a customer class should be created. A customer class should contain customer attributes which are the details of this customer and customer methods like buying products, viewing products, or confirming orders.

Exercise (21)

1. Why is the class diagram used?
2. A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs. Based on the previous details, discuss what the needed classes in the class diagram would be with your group.

5 Use Case Diagram (Behavioral)

The use case diagram is one of some UML diagrams that clarify the dynamicity of the project. Unlike the class diagram. In addition to this, it is characterized by explaining the probable user interactions which may be internal and external requirements for the system. Each requirement of the project is expressed as a single-use case diagram. So, in the next you could detect:

a Why Do We Use the Use Case Diagram?

1. It is dynamic.
2. It identifies and describes each interaction (visually) in the project.
3. It enables the project owner to preview the roles of the project in the graphical representation besides the documentation.
4. You could see the different types of users of the project.
5. Because of the relationship between the different users.

After defining the use case diagram and its importance, we could start showing the different components of the use case diagram.

b Use Case Diagram Components

1. Actors:

They represent the project users to apply for some project roles as shown in figure (24).

2. Relationships:

They exist in different types like Include, Extend, and Generalization. They are used to connect the use cases as shown in figure (25).

3. Use Case:

It represents one functionality in the project.

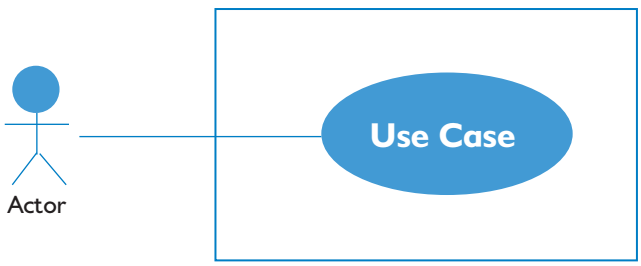


Figure (24): Use case diagram

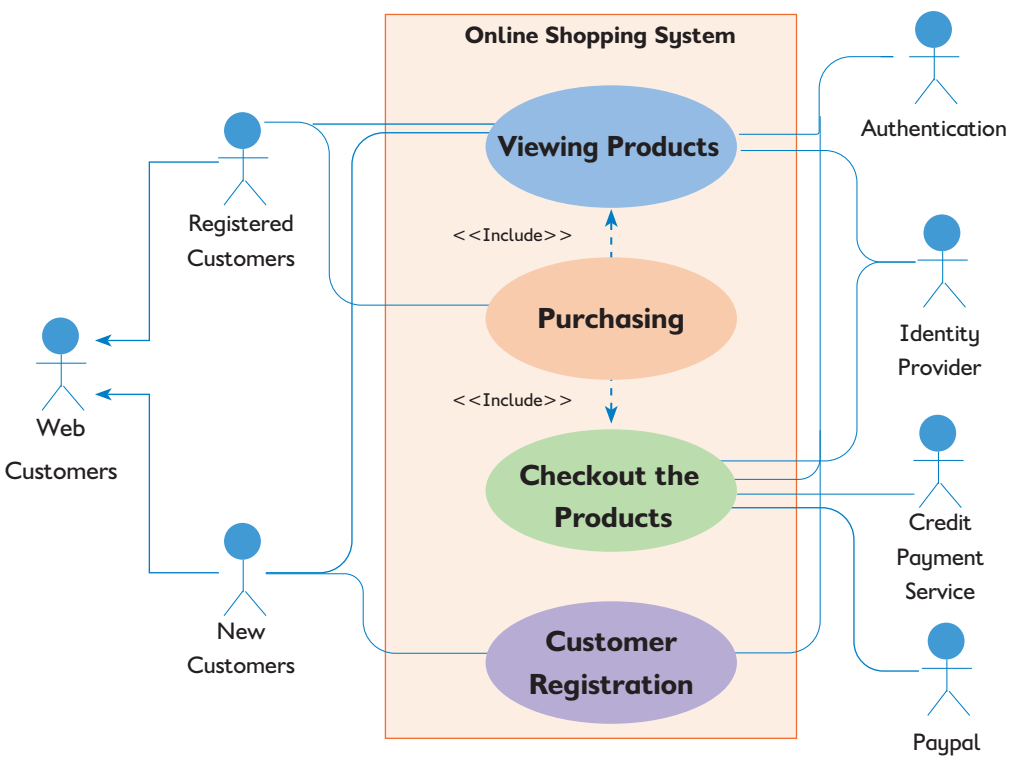


Figure (25): Sample use case diagram with relationships

Exercise (22)

Why is the use case diagram used?

Note (12)

Each user dealing with the system is considered an actor in the use case diagram.

Example

If a system has customers, a customer is represented as an actor and all the functionalities that the customer can perform should be represented as a use case.

Exercise (23)

A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs. Based on the previous details, discuss who the actors in the use case diagram would be, what the used cases for each actor would be, and what the relationships among the actors within the system would be with your group.

6 Sequence Diagram (Behavioral)

The sequence diagram is one of the behavioral UML diagrams. It shows the communication between the system and the users or between the system items. Its name is 'sequence' which focuses on showing the sequence of interactions between the different sub-systems during some actions in the project. It has a set of advantages that could be shown by answering the following question:

a Why Do We Use the Sequence Diagram?

1. Because of dynamicity and time.
2. It shows the event across its lifetime of it.
3. It displays the different messages during the event's lifetime.
4. It could be used by the software developers or the project stakeholder.

In the following, we will show the different components of the sequence diagram:

b Sequence Diagram Components

1. Actors:

They represent the external role of the system in which it interacts with the project and its objects.

2. Lifelines:

They are named entities that represent an entire object entire the system and it is at the top of the sequence diagram.

3. Guards:

They define the conditions and constraints of the project.

4. Messages:

There are several types of messages in the sequence diagram. Each type of them represents a specific condition within the scenario. Next, the types of messages in the sequence diagram will be overviewed.

a) Synchronous Message:

A synchronous message waits for a response before moving into the next interaction.

b) Asynchronous Message:

An asynchronous message does not wait for a response from the receiver. The interaction continues in proceeding regardless of whether the processing of the receiver for the previous message occurred or not.

c) Create a Message:

It is used when creating a new object in the sequence diagram.

d) Delete a Message:

It is used when you want to destroy an object.

e) Self-message:

It is used when the object in the sequence diagram wants to send a message to itself.

f) Reply a Message:

It is used to point out that the message receiver is done processing the message and is returning control over to the message caller.

g) Found Message:

Messages that are reached from a strange sender, or a sender not shown on the current diagram.

h) Lost Message:

Messages that are either sent but do not arrive at the intended receiver, or which go to a receiver not shown on the current diagram.

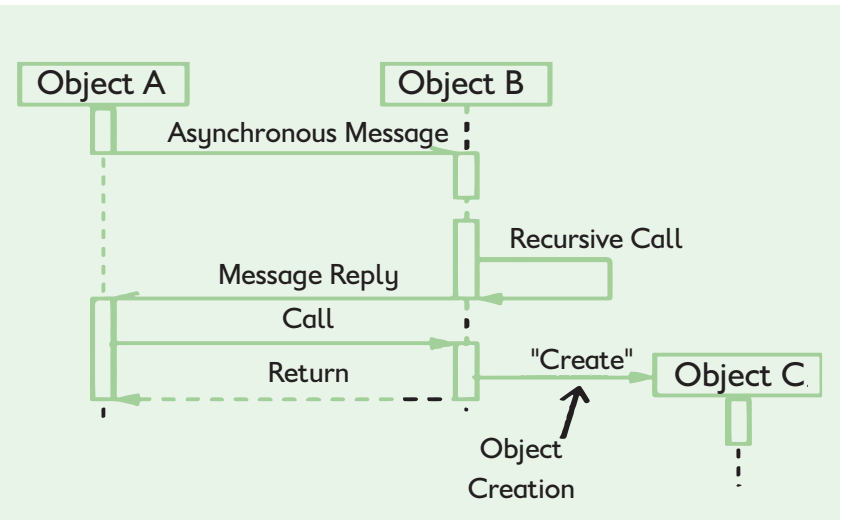


Figure (26): Sequence diagram

Exercise (24)

1. Why is the sequence diagram used?
2. List the types of messages in the sequence diagram and mention the case each one is used for.

Note (13)

The sequence diagram represents a timeline that begins at the top and descends gradually to mark the sequence of interactions.

Example

The online shopping process goes through a set of transactions. The proper way of representing the dependency among these transactions is by the sequence diagram.

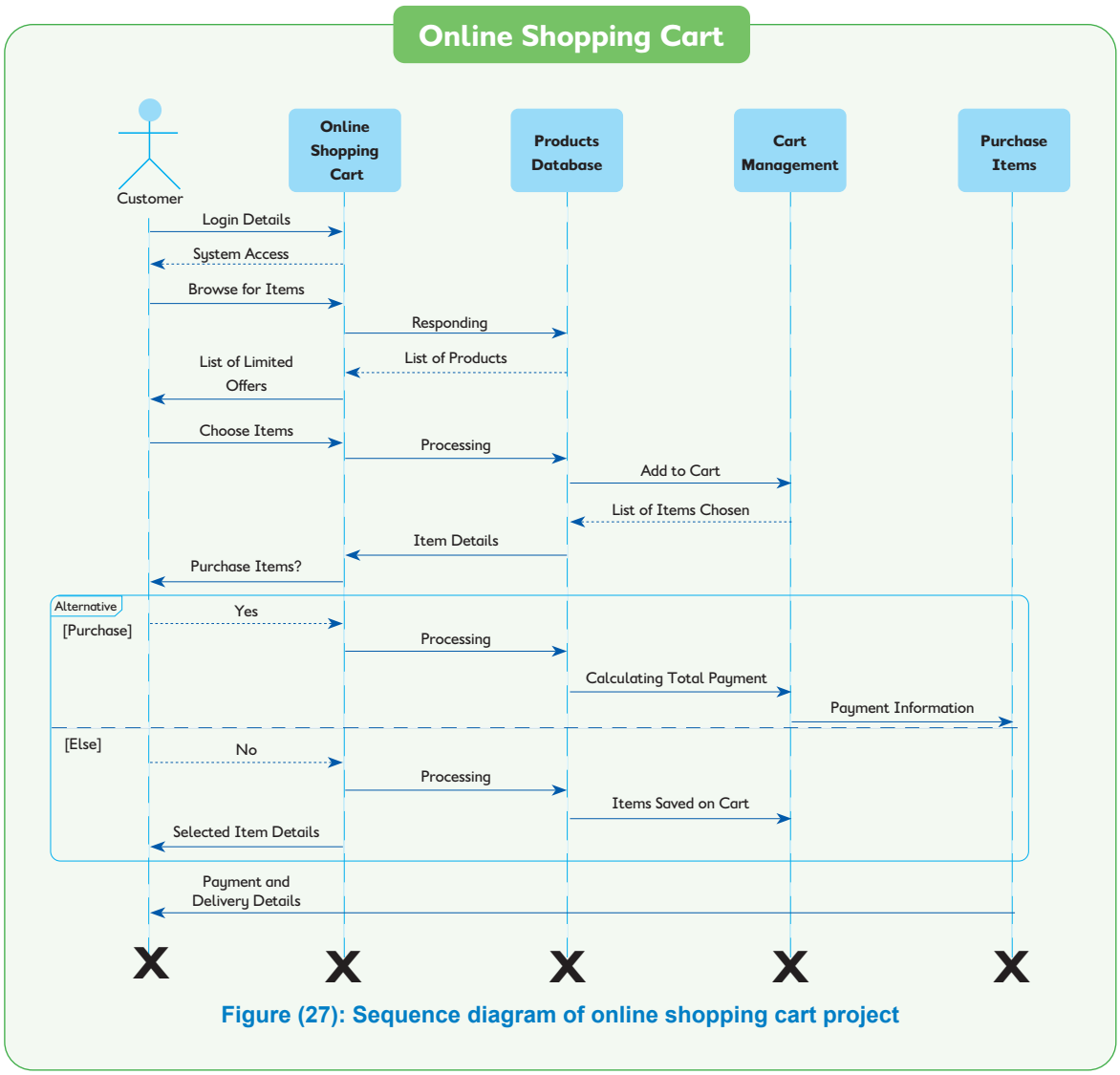


Figure (27): Sequence diagram of online shopping cart project

Exercise (25)

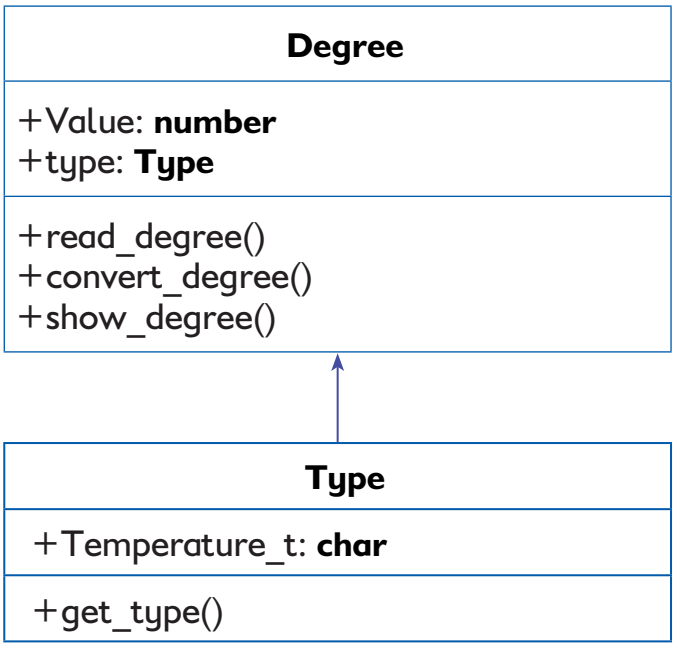
A customer is willing to develop a website for selling dairy products. The website should be related to the dairy products factory. And the website should also be related to the application where the users can order their needs.

Based on the previous details, discuss the sequence of transactions for one online process of buying a set of dairy products with your group and then clarify the type of messages used.

7 Case Study

Apply the most common UML diagrams which you learned like the class diagram, use case diagram, and sequence diagram to the same temperature case study that had been mentioned in Outcome 1. (We will solve the class diagram and you will solve the use case diagram and the sequence diagram).

Class Diagram:



Assessment

Q 1. What does UML stand for?

Q 2. What is meant by Unified Modeling Language?

Q 3. What are the main types of UML?

Q 4. List the types of UML structural diagrams.

Q 5. List the types of UML behavioral diagrams.

Q 6. Differentiate between structural and behavioral diagrams.

Q 7. Draw a class diagram for a veterinary clinic.

Q 8. Draw a class diagram for a restaurant.

Q 9. Draw a class diagram for a delivery system (like Mrsool).

Q 10. Draw a class diagram for a factory.

Q 11. Draw a sequence for the following:

a) In a restaurant, kitchen items came to life. Draw a sequence diagram for making a peanut butter and jelly sandwich if the following objects are alive: knife, peanut butter jar (and peanut butter), jelly jar (and jelly), bread, and plate. I may or may not want the crusts cut off. Do not forget to open and close things like the jars, put yourself away, clean up, etc.

b) Getting on a flight. Start at home, check in at the counter, go through security and end up at the gate. (If you have time during the exercise, get yourself to your seat.) (You may get searched in security.)

c) Getting money from the ATM machine

- Treat each part of the ATM as a class (Money dispenser, Screen, Keypad, Bank computer, etc.)

Programming Paradigms

Introduction

The programming paradigm is the way that a methodology of classification programming language is based on some features in it. There are various programming paradigms, some are concerned with the structure of the code written, while others are concerned with the execution sequence of the code. To summarize, the programming paradigm is a technique to solve problems using programming language rules.

Usually, programming paradigms are classified into imperative and declarative paradigms.

- **Imperative Paradigm:** It is a paradigm that makes the programmer teaches the machine how to alter its condition. The imperative paradigm includes but is not limited to procedural and object-oriented programming paradigms.
- **Declarative Paradigm:** It is a paradigm that makes the programmer just declare the properties of the program without explicitly computing it. In other words, declarative programming declares the output result instead of computing how it is processed.

There are several programming paradigms including but not limited to logic, functional, and database programming.

Exercise (26)

Differentiate between imperative and declarative programming paradigms.

In this lesson, two imperative paradigms and one declarative paradigm will be covered as shown in figure (28). The most common three programming paradigms are structured programming, object-oriented programming, and functional programming. Structural and object-oriented programming is two types of imperative programming.

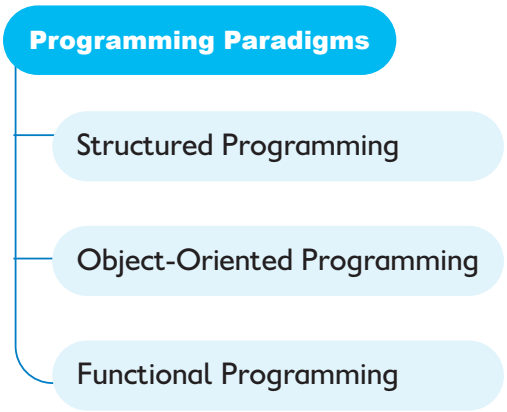


Figure (28): Covered programming paradigms

Note (14)

The programming paradigm states the style of programming.

It does not state a specific programming language but the way that the developer solves the problem.

Exercise (27)

1. List three types of programming paradigms.
2. Programming paradigms refer to a specific programming language. (True or False)

Structured Programming

The first paradigm to be applied was structured programming (but not the first invented). Structured programming was founded by Edsger Wybe Dijkstra in 1968. Structured programming enhances the code flow and quality by utilizing control flow, repetition, and subroutine structures.

Structured programming emphasizes:

- The total logic of an application program.
- What the application program does.
- What the logical flow through the application program should be.

Exercise (28)

Based on your knowledge, define structured programming.

a Structured Programming Concepts

Control Structures:

The software consists of a set of ordered statements. In some cases, one or more statements should be executed based on the state of part of the software. This is known as a conditional statement that should have one valid flow that the program should follow. This is usually known as if statements in programming languages.

Example

Assuming that you want to make a piece of code that draws a happy face if the user entered H for happy, or a sad face if the user entered S for sad. The suitable way to represent these cases is by using control structures.

Repetition:

In some cases, a statement or a set of statements should be repeated until a specific condition happened in the program. This is known as loops in programming.

Subroutines:

Subroutines are a piece of code that can be used several times. Subroutines are also known as procedures, methods, or functions. Applying subroutines in development enhances code quality and reusability.

Exercise (29)

1. Subroutines in structured programming are also known as ....., ..., or ..... .
2. List structured programming concepts.
3. Use a search engine to help you how to implement the three concepts of structured programming in C++.

b Advantages of Structured Programming

1. Application programs are easier to read and understand.
2. Higher productivity during application program development.
3. Errors are more easily found.
4. Improved application program design.
5. Application programs are less likely to contain logic errors.
6. Application programs are more easily maintained.

Exercise (30)

List the advantages of structured programming.

Assessment

Q 1. What are the main two types of programming paradigms?

Q 2. Fill in the blanks.

a) Structured programming enhances the code flow and quality by utilizing ....., ..., and ..... .

b) Subroutines in structured programming are also known as ....., ..., or ..... .

c) Object-oriented programming follows a set of concepts which are ....., ..., ..., and ..... .

d) ..... is a template or blueprint from which objects can be instantiated from.

e) ..... is a subroutine defined within a class to implement a behavior.

Q 3. What are the main points that structured programming emphasizes?

Q 4. What are the structured programming concepts?

Q 5. List the advantages of structured programming.

Q 6. What is meant by inheritance in the OOP?

Q 7. What is meant by abstraction in the OOP?

Q 8. What are the advantages of functional programming?

Q 9:

1. Here is a function, double getNum (int x), what is the name of this function?

a) double. **b) x** **c) getNum.**

2. Say we have a function, double subtract (double a, double b), what is the correct way to call this function in the main program?

a) subtract (a). **b) subtract (b).** **c) subtract (a, b).**

Software Planning

1 What Is a Flowchart?

A flowchart is a diagram that outlines the steps in a process. It began as a tool for describing algorithms and programming logic in computer science but has since expanded to include all types of processes. A flowchart is a visual illustration of a series of steps. It is commonly used to illustrate the flow of algorithms, workflows, or processes in

sequential order. A flowchart often displays the processes as various types of boxes, with arrows connecting them in order.

2 Flowchart Symbols

Special shapes are used in flowcharts to describe different kinds of actions or steps in a process. Lines and arrows show the order of the steps and their interactions. Each flowchart symbol has a special meaning. The most common flowchart symbols are represented in the following table (4) and figure (29). Flowchart symbols include:

Table (4): Flowchart symbols

Symbol Name	Symbol Shape
Rectangle shape – it represents a process. A box indicates some particular operations.	
Oval or Pill shape - it represents the start or end. The terminator symbol represents the starting or ending point of the system.	
Diamond shape - it represents a decision. A diamond represents a decision or branching point. Lines coming out from the diamond indicates different possible situations, leading to different sub-processes.	
Parallelogram - it represents input/output. It represents information entering or leaving the system. An input might be an order from a customer. Output can be a product to be delivered.	
Flow: Lines represent the flow of the sequence and direction of a process.	
Connectors: the connector indicates that the second flowchart segment begins where the first segment ends.	
Subroutines (Modules): A module is represented by a special symbol. The position of the module symbol indicates the point at the module is executed. A separate flowchart can be constructed for the module.	

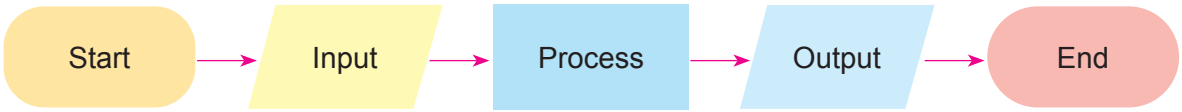


Figure (29): Flowchart symbols

3 Flowchart Structures

There are four main flowchart structures as shown in figure (30).

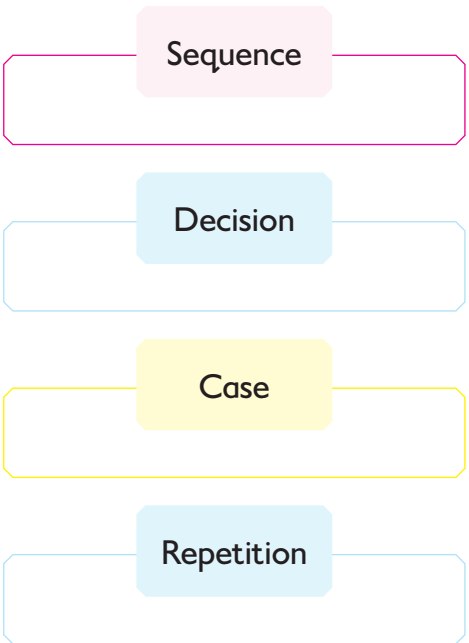


Figure (30): Flowchart structures

Sequence Structure:

A series of actions are performed in sequence as shown in figure (31).

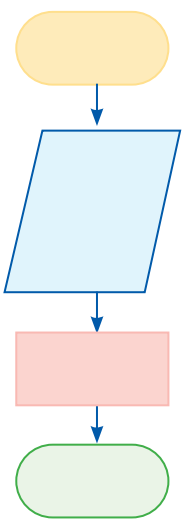


Figure (31): Sequence structure

Decision Structure:

In decision structure, one of two possible actions is taken, depending on a condition as shown in figure(32). The diamond symbol, indicates a yes/no question. If the answer to the question is yes, the flow follows one path. If the answer is no, the flow follows another path.

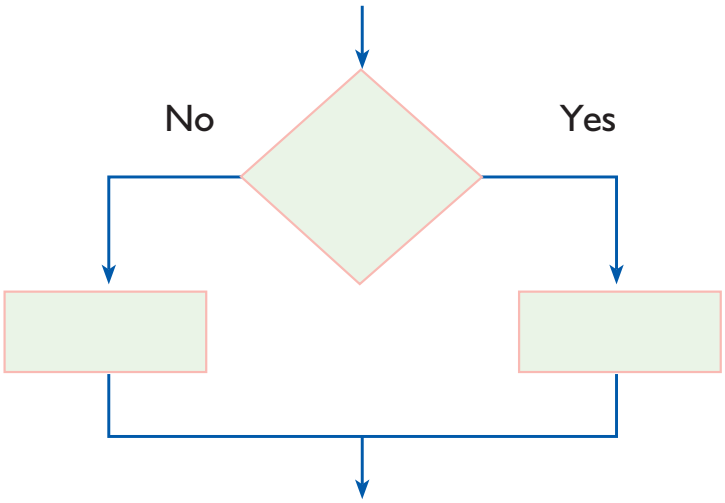


Figure (32): Decision structure

In figure (33), the question "Is x < y?". If the answer is no, then Process A is performed. If the answer is yes, then Process B is performed.

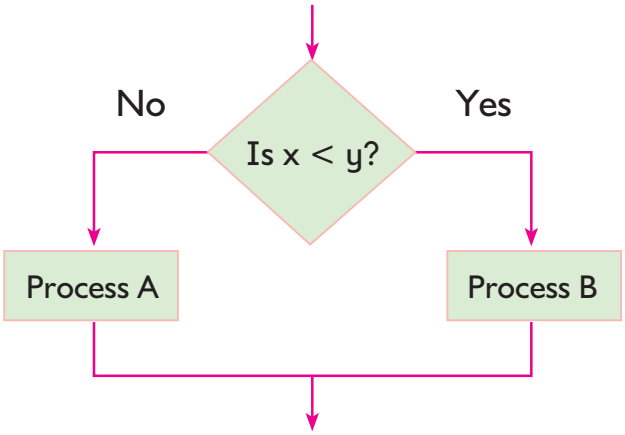


Figure (33): Decision structure example

Case Structure:

In case structure, one of several possible actions is taken, depending on the contents of a variable as shown in figure (34).

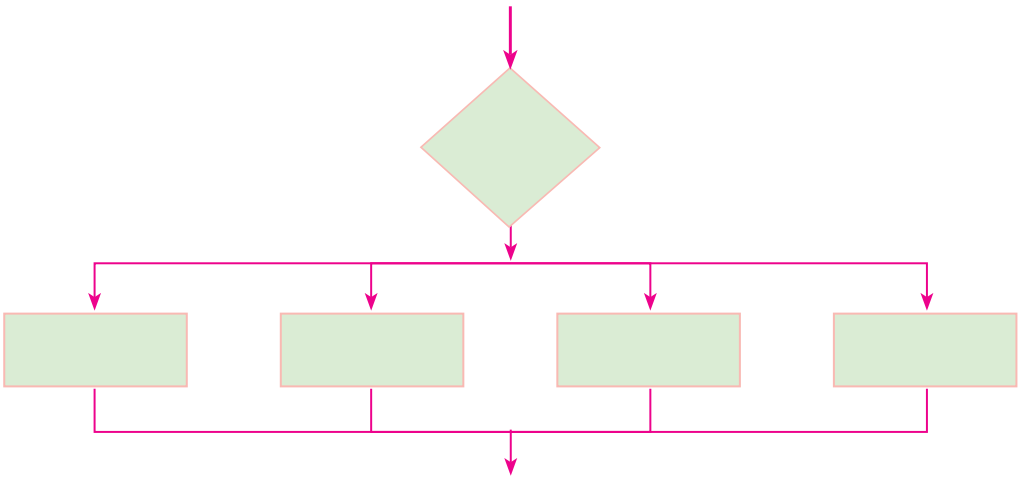


Figure (34): Case structure

Figure (35) indicates actions to perform depending on the value in years_employed.

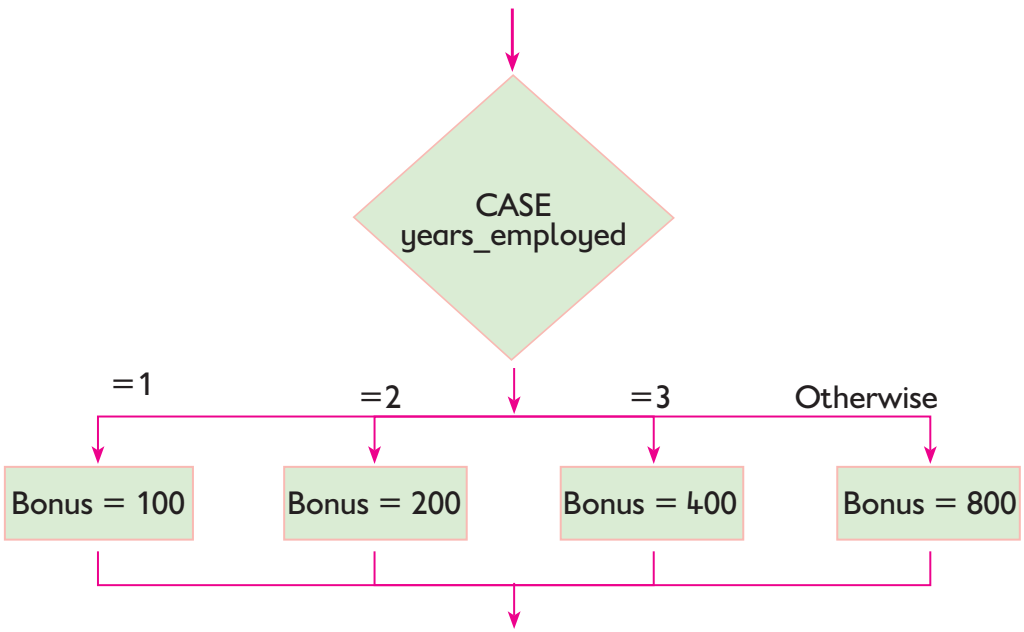


Figure (35): Case structure example

Repetition Structure:

The repetition structure represents part of the program that repeats as shown in figure (36). This type of structure is commonly known as a loop. Notice the use of the diamond symbol. A loop tests a condition, and if the condition exists, it acts. Then it tests the condition again. If the condition still exists, the action is repeated. This continues until the condition no longer exists.

Figure (36): Repetition structure

In figure (37), the question "is x < y?". If the answer is yes, then Process A is performed. The question "is x < y?" is asked again. Process A is repeated as long as x is less than y. When x is no longer less than y, the repetition stops and the structure is exited.

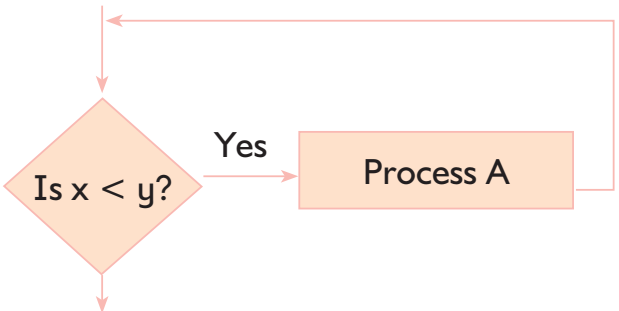


Figure (37): Repetition structure example

Figure (38) below shows a repetition structure that increments x by 1 as long as x is less than y.

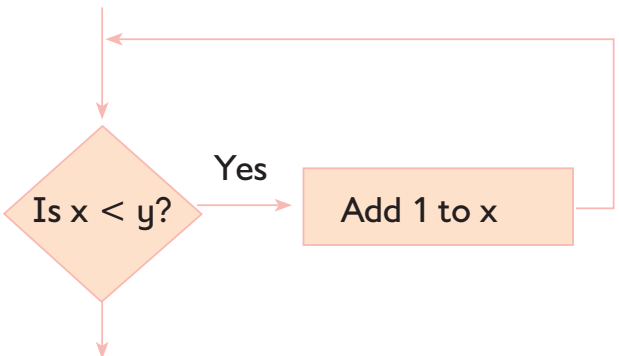


Figure (38): Increment by 1 repetition structure example

Types of Repetition Structure:

There are two main types of repetition structures which are pre-test and post-test structures.

1. Pre-test Repetition Structure:

In the pre-test repetition structure, the condition is tested BEFORE any actions are performed as shown in figure (39). In a pre-test repetition structure, if the condition does not exist, the repetition will never begin.

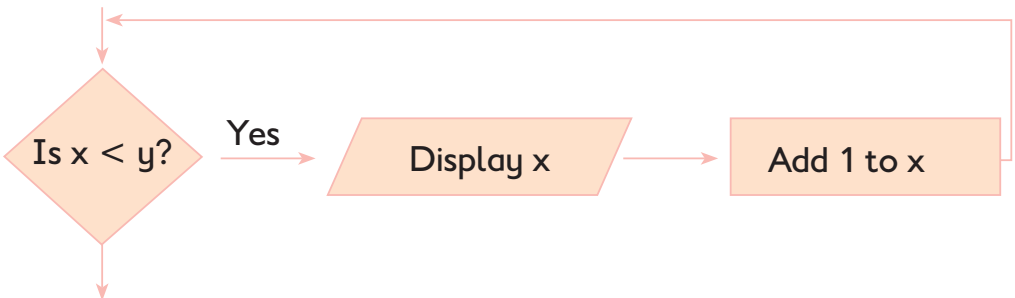


Figure (39): Pre-test repetition structure example

2. Post-test Repetition Structure:

In the post-test repetition structure, the condition is tested AFTER the actions are performed as shown in figure (40). A post-test repetition structure always performs its actions at least once.

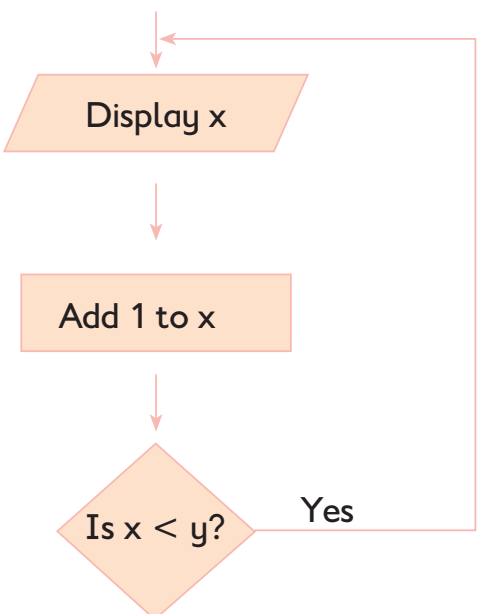


Figure (40): Post-test repetition structure example

Flowchart Examples:

Regardless of the complexity of the algorithm, a flowchart can be utilized to visualize it. Here's an example of how to use a flowchart to represent a simple summation process.

Example

Find the summation of 529 and 256. Add two numbers A and B, then add the two numbers 529 and 256 to produce the sum as 785.

Solution of Example (1):

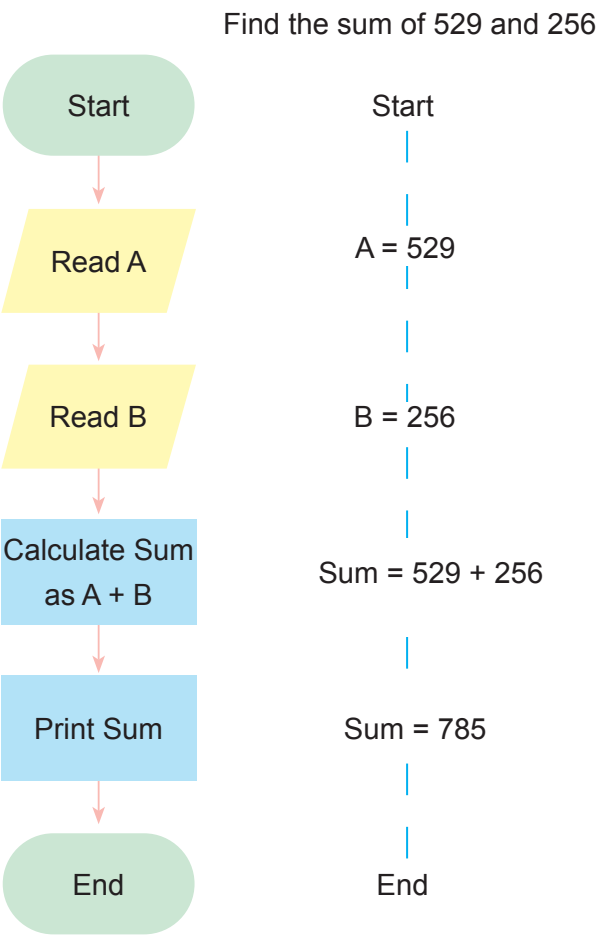


Figure (41): Flowchart of Example (1)

Example

The flowchart example below shows how profit and loss can be calculated.

Solution of Example (2):

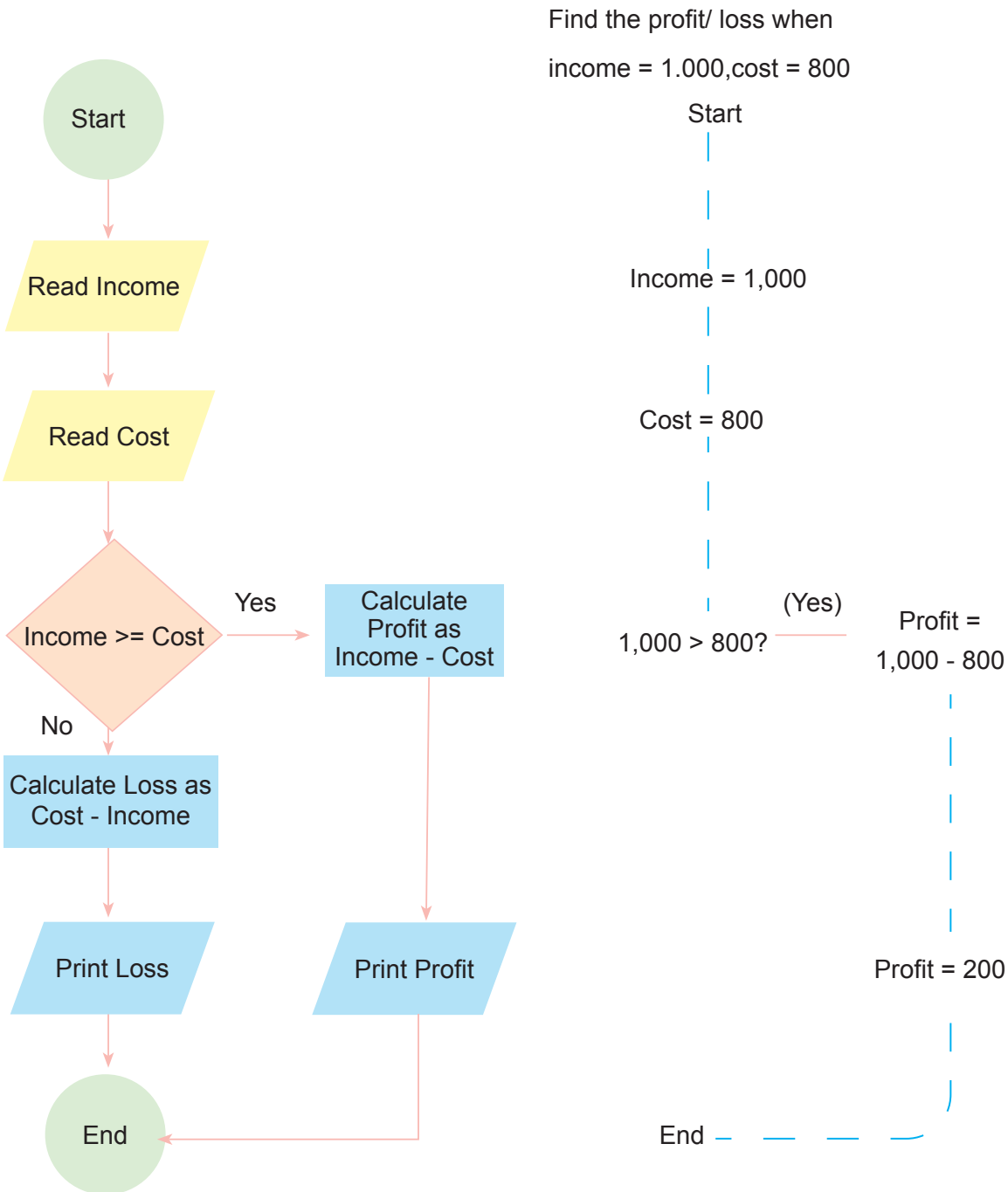


Figure (42): Flowchart of Example (2)

Example

What does the following flowchart shown in the following diagram perform?

Solution of Example (3):

The flowchart shows how a decision structure multiplies x by y if x is less than y. Otherwise, it adds x to y.

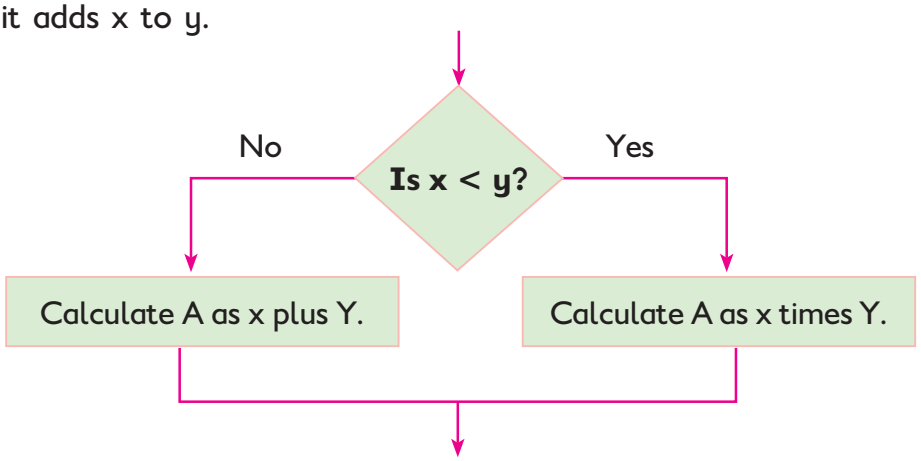


Figure (43): Flowchart of Example (3)

Example

The program increments an integer by one and displays the new value as long as it is less than 10 by the pre-test structure.

Solution of Example (4):

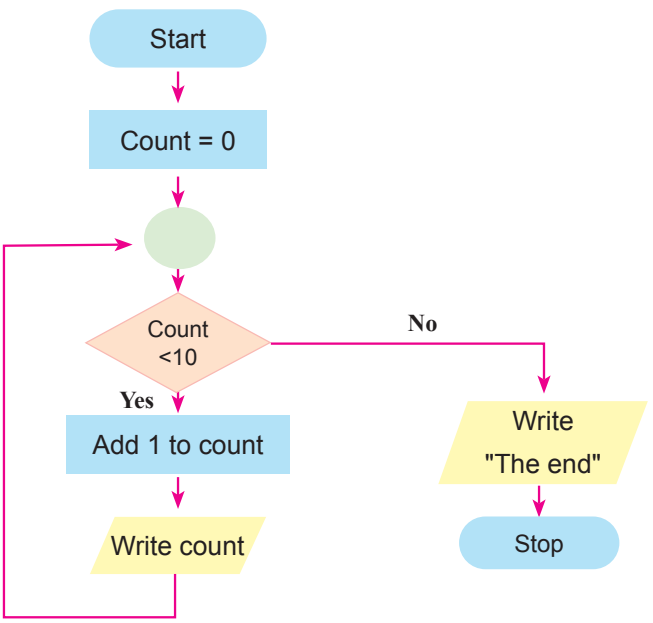


Figure (44): Flowchart of Example (4)

Example

The program increments an integer by one and displays the new value as long as it is less than 10 by the post-test structure.

Solution of Example (5):

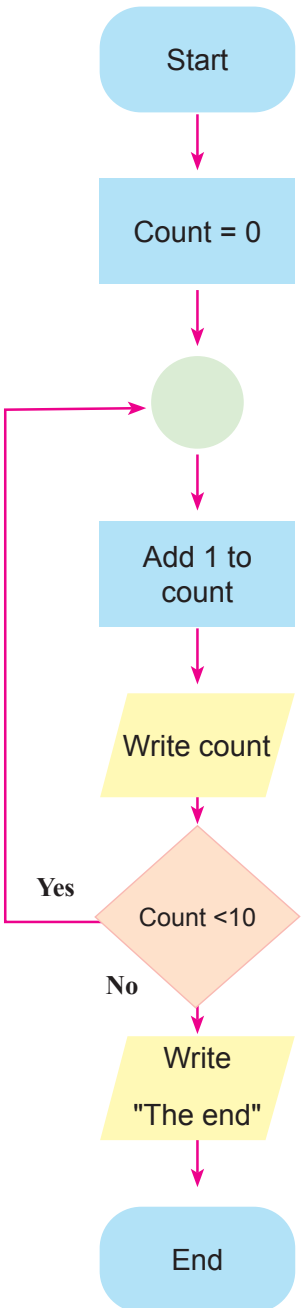


Figure (45): Flowchart of Example (5)

Example

What does the following flowchart shown in the following diagram perform?

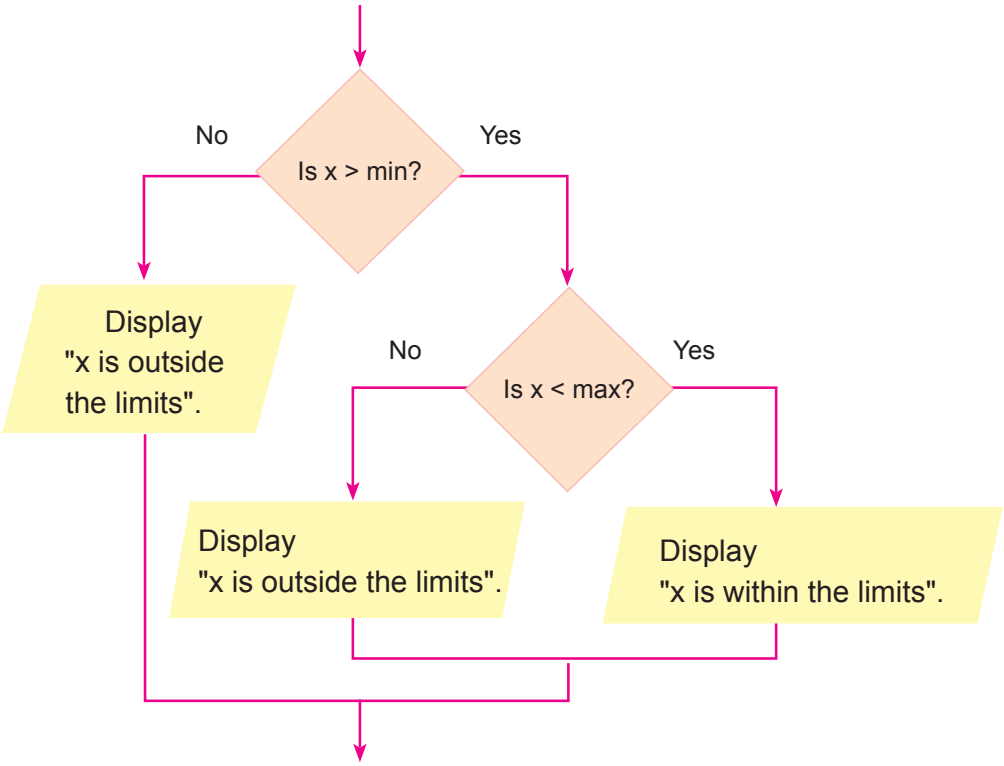


Figure (46): Flowchart of Example (6)

Solution of Example (6):

The previous flowchart checks whether x is within the limits between min and max or not and displays the decision on the screen.

Report plan and design

As a software engineer, I understand how crucial it is to carefully organize a software development project, and how challenging this may be. But in software development, if you don't plan, you're planning to fail - or at least to extend the timeline, increase the expense, and aggravate the team. The fundamentals of software project planning are outlined here.

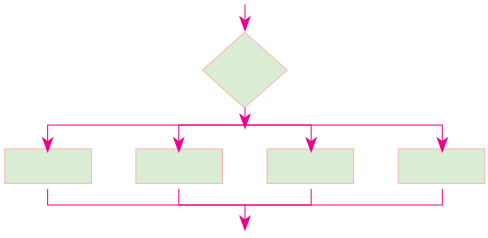
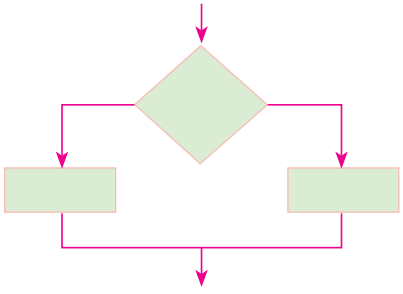
Table (5): Report plan and design

1.	Gola and Scope	Having a clear concept of where you want to end up and what you want to accomplish may be greatly aided by outlining your project's objectives and scope.
2.	What you need?	After settling on certain objectives, you may calculate the means necessary to achieve them. Your needs will define the other components of your project, such as the personnel, resources, timeline, expertise, and financial allocation. Your project's scope and objectives will determine which factors are most important. If you feel you need specialist guidance in this area, a project manager is a good resource to tap into for assistance.
3.	Programming Language	The next stage is to settle on a development language and set of tools. Pick a language to study based on your interests. If you can't make up your mind, think about which languages might serve the project best. It's true that certain languages are more optimal for specific jobs than others.
4.	Cost	A software development project's budget is one factor that must be considered. You may get a better idea of how much money will be needed for your project if you take the time to make a detailed list of all the things you'll need. In order to find a happy medium between spending less money and getting subpar outcomes, many businesses are turning to nearshore software developers.
5.	Timeline	After you have established these fundamental aspects of your software development strategy, you can go on to creating a timetable of the activities you will do. You may roughly estimate how long your project will take by breaking it down into smaller jobs and taking a look at your available resources like money and people. While software development has come a long way, there is still a lot of room for error. If you want your project to be completed on schedule, you need either give yourself more time to do everything or hire a project manager.
6.	List all features and entities	The next step is to sit down at a computer and make a list of every function your program will have. Future phases of the project will consist of these. In the meanwhile, make a list of everything that must be included and everything that would be nice to have. Say, for instance, you're interested in developing an account-based website. A profile image option is nice to have, but not required, and a login and password management system are must.

7.	Map the project architecture	We will now create a flowchart for our capstone project. If you want to rearrange the components, post-it notes or a digital version are your best bet. Sort the features into separate boxes and hang them all together. The next step is to draw a line from each characteristic to the ones that matter. These elements may be implemented simultaneously or at a later point in the program's timeline.
8.	Build Program	The student will build the program.
9.	Progress and quality	The planning process does not end when a project is set up and ready to begin. Testing for quality and checking in on progress should both be part of your strategy. The software development process is not complete until quality assurance testing has been accounted for. Working with an experienced software development outsourcing business helps simplify the process of planning a software development project. To hasten the completion and success of your project, consider hiring a firm to offer a project manager, software development team, recruiting support, and quality testing.

Assessment

Q 1. What type of structure is this?

	Structure	Type
1.		
2.		
3.		
4.		

Q 2. Write a flowchart for a program that reads a number and determines whether it is positive, negative, or zero.

Q 3. Write a flowchart for a program that checks whether an input number lies within a specified range. For example: If the user enters a range (20, 50) and queried number34, the program should display "In range". On the other hand, if he/she enters 76, the program should display "Not in range".

Q 4. Write a flowchart for a program that determines whether a baby's weight is normal or not. For girls, the normal weights are 2.5 to 4.5 kg. On the other hand, for boys, the normal weights are 4 to 5.5 kg.

Q 5. Write pseudocode and draw a flowchart for a program that reads a number and determines whether it is positive, negative, or zero.

Q 6. Write pseudocode and draw a flowchart for a program that checks an input number that lies within a specified range. For example, if the user enters range (20, 50) and queried number 34, the program should display "In range". On the other hand, if he/she enters 76, the program should display "Not in range".

Q 7. Write pseudocode and draw a flowchart for a program that determines whether a baby's weight is normal or not. For girls, the normal weights are 2.5 to 4.5 kg. On the other hand, for boys, the normal weights are 4 to 5.5 kg.

Q 8. Write pseudocode and draw a flowchart for the bank withdrawal process. First, the program reads the customer balance and withdrawal amount.
The transaction is completed in case the balance covers the required amount.
If not, display the message "Insufficient funds". Finally, display the customer balance.

Q 9. Write pseudocode and draw a flowchart for an algorithm that represents a program that calculates the absolute difference between two numbers.

Q 10. Write pseudocode and draw a flowchart for an algorithm that represents a program that calculates the net salary of an employee based on his/her job: 'M' for a manager and 'E' for an employee.

- The salary of the manager = salary + bonus
- The salary of the employee = salary – tax

2nd Learning Outcome

Writing a code for the previously designed software program.

Evidence and Proof Requirements

After completing this section, students should be able to:

- Write a code according to the design prepared.
- Address software issues and errors that appear during the development process.
- Use software language properties during the development process.
- Abide by the rules of naming conventions in accordance with its organization.
- Add sufficient comments to each part of the code to clarify its purpose.

Developing First Software Project

Introduction

To start developing any software program, you should know how the computer can understand your block of code. You could imagine the computer as a human with a different language. So, you need to learn its language. A computer is a hardware component; therefore, it can deal only with signals in binary values zeros (0's) and ones (1's).

The binary language is very different from the nature of human languages. So, some scientists have thought about how to make the computer understand human language characters. These scientists have developed some programming languages to be intermediate languages between the human language and the computer language.

The programming language is a human language written with special syntax so that the computer can understand it. Moreover, an interpreter is required to be an intermediate layer between the programming language and the computer binary language. This interpreter is sometimes called a compiler. The role of the compiler is to translate the programming language from the human language into the computer language (binary values 0's and 1's).

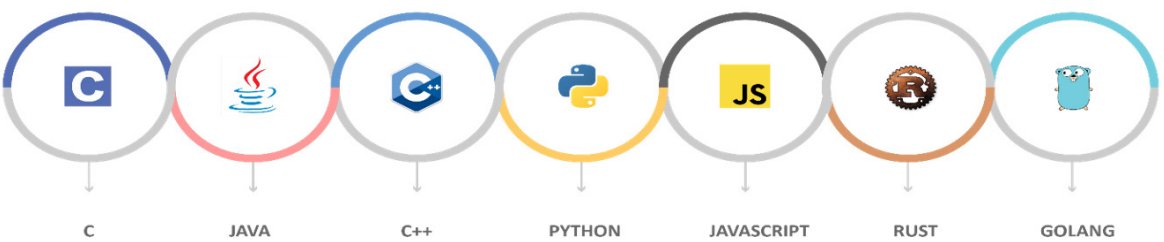


Figure (47): The most common programming languages

There are a lot of programming languages in the world, as shown in figure (47), each of them with its features that make it different from the others. These programming languages such as C, Java, C++, Python, JavaScript, Rust, Golang, etc.

Introduction to Python

1 What Is Python?

Python is a high-level, versatile, and easy-to-learn programming language known for its simplicity and readability. Developed by Guido van Rossum and first released in 1991, Python has gained immense popularity in various domains, including web development, data science, scientific computing, artificial intelligence, and more.

Key Features of Python:

- **Readability:** Python's syntax is clear and expressive, making it easy for programmers to write and understand code.
- **Versatility:** Python supports a wide range of applications, from web development to data analysis and automation.
- **Large Standard Library:** Python comes with an extensive standard library that simplifies common programming tasks.
- **Community Support:** Python has a vibrant and active community that continually develops libraries and frameworks, making it suitable for various projects.
- **Cross-Platform:** Python is available on multiple platforms, including Windows, macOS, and Linux.

2 Why Python?

Python's popularity stems from its numerous advantages and applications:

1.2.1: Ease of Learning

Python's simple and readable syntax is ideal for beginners. It emphasizes code readability and reduces the cost of program maintenance.

1.2.2: Versatility

Python is versatile, making it suitable for various applications, including web development, data analysis, scientific computing, machine learning, and more.

1.2.3: Abundant Libraries

Python's extensive standard library and third-party packages provide pre-built functionality for a wide range of tasks, reducing development time.

1.2.4:- Community and Support

Python's strong community fosters collaboration, provides documentation, and offers support through forums and online resources.

1.2.5: Career Opportunities

Proficiency in Python opens doors to a wide array of career opportunities in the tech industry, given its relevance to modern technology.

3 Setting up a Python Environment

Before you start writing Python code, you need to set up your development environment.

Here are the essential steps to get started:

1.3.1: Installing Python

To install Python on your computer, follow these steps:

1. Visit the official Python website at python.org.
2. Download the latest Python version for your operating system (e.g., Windows, macOS, or Linux).
3. Run the installer and follow the installation instructions.

1.3.2: Interactive Mode (IDLE)

Python comes with an interactive mode called IDLE, which allows you to experiment with Python code interactively. You can open IDLE from your computer's applications or by running "python" in the terminal or command prompt.

1.3.3: Integrated Development Environments (IDEs)

While IDLE is a basic environment, many developers prefer using integrated development environments (IDEs) for Python development. Some popular Python IDEs include:

- PyCharm
- Visual Studio Code
- Jupyter Notebook
- Spyder

Choose an IDE that suits your needs and preferences.

1.3.4: Writing Your First Python Program

Let's start with a simple Python program to get a feel for the language. Open your chosen development environment and create a new Python file. In your file, you can write:

```
print("Hello, Python!")
```

Figure (48): Printing a greeting

In Figure (48), save the file with a `.py` extension (e.g., `hello.py`) and run it. You should see the text "Hello, Python!" printed to the console.

Congratulations! You have successfully set up Python and written your first program. In the following chapters, we will explore Python's fundamental concepts and dive deeper into its capabilities.

4 Python Program Structure

A Python program consists of a sequence of statements that are executed one by one. Here's a basic program structure:

```
# This is a comment

# Import libraries/modules (if needed)
import module_name

# Define functions (if needed)
def my_function():
    # Function code here

# Main program logic
if __name__ == "__main__":
    # Your code here
```

Figure (49): Organizing your Python program

In Figure (2x)

• **Comments:** Python allows you to add comments using the `#` symbol. Comments are ignored by the Python interpreter and are used for documentation and readability.

• **Importing Modules:** You can import external modules or libraries to extend Python's functionality. Import statements usually appear at the beginning of your program.

• **Defining Functions:** Functions are reusable blocks of code. You can define your functions to organize your code logically.

• **Main Program Logic:** The `if __name__ == "__main__":` block ensures that the code within it is only executed when the script is run directly (not when it's imported as a module).

Note (15)

In Python, each statement is typically written on a separate line, and the interpreter understands the end of a statement based on line breaks and proper indentation. There's no need for semicolons in Python.

```
a = 42
b = 3.14
c = 'X'
```

Figure (50): End of a statement

5 Comments

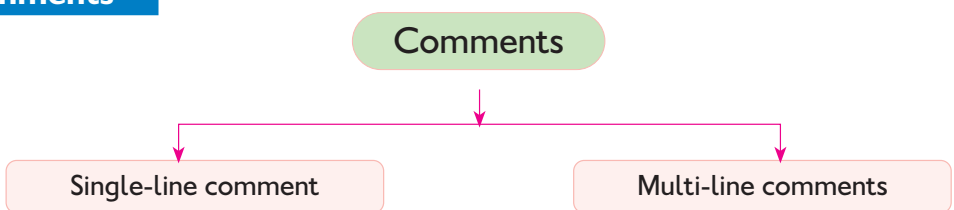


Figure (51): Types of comments in Python

Comments are essential for documenting your code and making it more understandable. In Python, you can add comments using the `#` symbol:

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

Figure (52): Single-line comment

In Figure (52): A single-line comment is denoted by a `#` symbol. Any text following the `#` symbol on the same line will be ignored by the Python interpreter. You can place single-line comments before or after the code.

```
"""
This is a
multi-line comment
```

Figure (53): Multi-line comments

In Figure (53), there are are multi-line comments that can be created using triple-quotes, either single `(''')` or double `(""")`. To begin a multi-line comment, you start with three quotes, and to end it, you also use three quotes.

6 Variables

Understanding Variables and Their Naming Rules:

Variables are used to store data. In Python:

- Variable names are case-sensitive (`my_variable` is different from `My_Variable`).
- Variable names can consist of letters, digits, and underscores but cannot start with a digit.
- Variable names should be descriptive and follow a naming convention (e.g., `my_variable` or `user_input`).

```
my_variable = 42  # Integer
another_variable = 3.14  # Float
text_variable = "Hello, Python!"  # String
```

Figure (54): Variable declaration

In Figure (54), the variable declaration in Python typically consists of three key components. The first part, which will be covered in the next section, is the data type. The second part is the variable name, which should be chosen meaningfully and follow specific rules for definition, as outlined later. Finally, in contrast to some other languages, there is no need for a semicolon (;) at the end of a variable declaration in Python.

Example

Determine the valid and invalid variables for each of the following:

•string num;	Valid
•int _x;	Valid
•int ab;	Valid
•int 2;	Not valid
•int a;	Valid
•int ab;	Valid
•int float;	Not valid
•int a30;	Valid

In this example, variables 1, 2, and 4 are valid, while variables 3, 5, 6, and 7 are invalid in terms of variable names because they don't follow the naming rules.

7 Output

In Python, it's not just about taking input from users; it's also important to provide meaningful output to users. Output in Python is typically done through the use of the `print()` function.

The "print()" Function:

The `print()` function is a versatile tool for displaying information to the user. You can pass various types of data to `print()` to display text, numbers, or the results of calculations.

Here's a basic example:

```
name = "Alice"
age = 30
print("Hello, " + name + "! You are " + str(age) + " years old.")
```

Figure (55): User output

Figure (55) shows the program that prints "Hello World!" on the console screen. The syntax of cout is for writing the first two lines of #include<iostream> and using namespace std;. So, we could use the cout object for writing the cout keyword followed by the << operator, and then the output that you could print on the console in our case is the string "Hello World!" ending with a semicolon.

Formatting Output:

Python provides several ways to format and control the output:

1. **String Formatting with f-strings:** You can use f-strings (formatted string literals) to embed expressions inside string literals. It allows for more readable and concise output formatting.

```
name = "Alice"
age = 30
print(f"Hello, {name}! You are {age} years old.")
```

Figure (56): String formatting with f-strings

2. **String Concatenation:** You can concatenate strings and variables using the + operator, as shown in the previous example.
3. **String Formatting with "str.format()":** Another way to format strings is using the `str.format()` method:

```
name = "Alice"
age = 30
message = "Hello, {}! You are {} years old.".format(name, age)
print(message)
```

Figure (57): String formatting with `str.format()`

Output: Hello, Alice! You are 30 years old.

8 Escape Sequence

In Python, an escape sequence is a combination of characters that represents a special character or control sequence. Escape sequences are typically used within strings to represent characters that cannot be easily typed or represented directly. They start with a backslash (\) followed by one or more characters.

- **Using Escape Sequences:** You can include special characters in your output using escape sequences. For example, `\n` represents a newline character, and `\t` represents a tab character.

```
print("This is a sentence.\nThis is on a new line.")
```

Figure (58): Escape sequences

Output: This is a sentence.

This is on a new line.

Here are some commonly used escape sequences in Python:

Table (6): Escape sequences

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark

- **Printing Variables and Expressions:**

You can directly print variables and the results of expressions without the need for explicit concatenation or formatting. Python will convert them to strings automatically:

```
x = 10
y = 20
result = x + y
print(x)      # Output: 10
print(y)      # Output: 20
print(result) # Output: 30
```

Figure (59): Variables printing

Exercise (31)

Write a program whose output has to be like in figure (60).

```
Hello
   world!
```

Figure (60): Hello world! exercise output

To achieve the desired output "Hello\n World!" where "World!" is indented with spaces, you can use the following Python program:

```
print("Hello")
print("     World!")
```

Figure (61): Hello world! exercise output

This program consists of two `print()` statements. The first `print ("Hello")` prints "Hello" on one line, and the second "print (" World!")" prints " World!" with leading spaces, creating the indentation.

Exercise (32)

Write a program to print the output in figure (62). You have to use escaping sequences only.

Figure (62): Escaping sequence exercise output

Answer:

```
print("*")
print("*\t*")
print("*\t*\t*")
print("*\t*")
print("*")
```

In this program, we use the escape sequence \t (tab) to create the desired pattern. When you run this code, it will produce the output as shown in Figure (62).

Note (16)

The "\t" (tab) character is used instead of "\n" (newline) to create a specific pattern of asterisks. The goal here is to create a triangular pattern with asterisks, and tabs are used to control the horizontal spacing between the asterisks.

Using "\n" would result in each asterisk being on a new line, which would create a vertical pattern, not the intended triangular shape.

Exercise (33)

Define Python variables to store the length of three sides of a triangle and a variable to store the triangle area.

Answer:

```
# Define variables to store the lengths of triangle sides
side_a = 5.0  # Length of side A
side_b = 7.0  # Length of side B
side_c = 8.0  # Length of side C

# Calculate the semi-perimeter (half of the perimeter)
semi_perimeter = (side_a + side_b + side_c) / 2

# Calculate the triangle area using Heron's formula
triangle_area = (semi_perimeter * (semi_perimeter - side_a) *

# Display the results
print("Length of side A:", side_a)
print("Length of side B:", side_b)
print("Length of side C:", side_c)
print("Triangle Area:", triangle_area)
```

In this Python code:

- We define variables `side_a`, `side_b`, and `side_c` to store the lengths of the three sides of the triangle.
- We calculate the semi-perimeter using the formula (side_a+ side_b + side_c) / 2`.
- We calculate the triangle area using Heron's formula, which requires the semi-perimeter and the lengths of the three sides.
- Finally, we display the lengths of the sides and the calculated triangle area using the `print()` function.

Understanding how to provide user-friendly output is crucial in Python programming. Whether you're displaying results, communicating with users, or debugging your code, the `print()` function and formatting techniques will be your allies in delivering clear and informative messages.

9 User Input

You can interact with users by taking input using the `input()` function:

```
user_input = input("Enter your name: ")
print("Hello, " + user_input + "!")
```

Figure (63): User input

Remember that `input()` returns a string, so you may need to convert it to the appropriate data type for calculations or comparisons.

Example

Read the input from the user and print it on the console.

```
# Get user input for a number
user_input = input("Type a number: ")

# Display the user's input
print("Your number is:", user_input)
```

The console output is:

```
Type a number: 22
Your number is: 22
```

When you run this program, it will prompt the user to enter a number. After the user enters a number, the program will display "Your number is:" followed by the entered number.

10 Data Types

Python supports various data types:

- **int:** represents integer values (e.g., `42`, `-10`).
- **float:** represents floating-point numbers (e.g., `3.14`, `-0.5`).
- **string:** represents text (e.g., `"Hello, World!"`, `'Python'`).
- **boolean:** represents true or false values (`True` or `False`).

You can use the 'type()' function to determine the data type of a variable.

In Python, you don't need to explicitly declare data types like in some other languages. Python uses dynamic typing, so you can simply assign values to variables and their data types are determined automatically. Here's how you can create variables with the equivalent data types in Python:

Example

Make variable declarations of the given datatypes below:

- Integer variable a
- Float variable b
- Character variable c

```
a = 42        # Integer
b = 3.14      # Float
c = 'X'       # String (single character)
```

Figure (64): Variable declaration example

In Figure (64):

- **"a"** is assigned an integer value.
- **"b"** is assigned a floating-point value.
- **"c"** is assigned a string containing a single character. Python treats single-character strings as regular strings.

Example

Define Python variables to store 2 numbers and the summation of these numbers, keeping in mind naming conventions.

As shown in the code below, we need three variables. Two of them are for the numbers, while the third one will be used to store their summation.

```
# Define variables to store two numbers and their summation
first_number = 10  # Using snake_case for variable names
second_number = 20  # Using snake_case for variable names
sum_of_numbers = first_number + second_number  # Using snake_case for variable names

# Display the results
print("First Number:", first_number)
print("Second Number:", second_number)
print("Sum of Numbers:", sum_of_numbers)
```

In this Python code:

We define three variables using snake_case naming conventions: `first_number`, `second_number`, and `sum_of_numbers`.

- We assign values to `first_number` and `second_number`.
- We calculate the sum of the two numbers and store it in the `sum_of_numbers` variable.

Finally, we display the values of all three variables using the `print()` function.

This program defines variables following naming conventions and performs the required operations.

11 Type Conversions

Type conversion, also known as type casting or type coercion, refers to the process of changing the data type of a variable or value from one type to another. Python provides several methods for performing type conversions to ensure compatibility and flexibility in your code. Here are the common types of type conversions:

1. Implicit Type Conversion:

Implicit type conversion, also known as coercion, occurs automatically when Python converts one data type to another without explicit instructions. This typically happens when performing operations between different data types.

Example

This example introduces the expected type conversion by type.

```
num_int = 10  # Integer
num_float = num_int  # Implicitly converts to float
```

Figure (65): Implicit conversion

2. Explicit Type Conversion:

Explicit type conversion occurs when you intentionally convert a value from one data type to another using built-in functions or casting.

2.1. Conversion Using Built-in Functions:

Python allows you to convert between different data types using casting. This powerful feature allows you to change the type of data held by a variable. Here are some common types of conversion functions:

- **"int()":** Converts to an integer.
- **"float()":** Converts to a floating-point number.
- **"str()":** Converts to a string.
- **"bool()":** Converts to a boolean.

Example

```
num_str = "42"
num_int = int(num_str)  # Converts the string "42" to an integer
```

Figure (66): Converting to an integer

2.2. Conversion Using the Cast Operator:

You can use the cast operator to explicitly convert a value to a specific data type.

Example

```
num_float = 3.14  # Float
num_int = int(num_float)  # Explicitly converts to integer
```

Figure (67): Cast operator

12 Python Operators

The Python programming language has five groups of operators, as shown in figure (68).

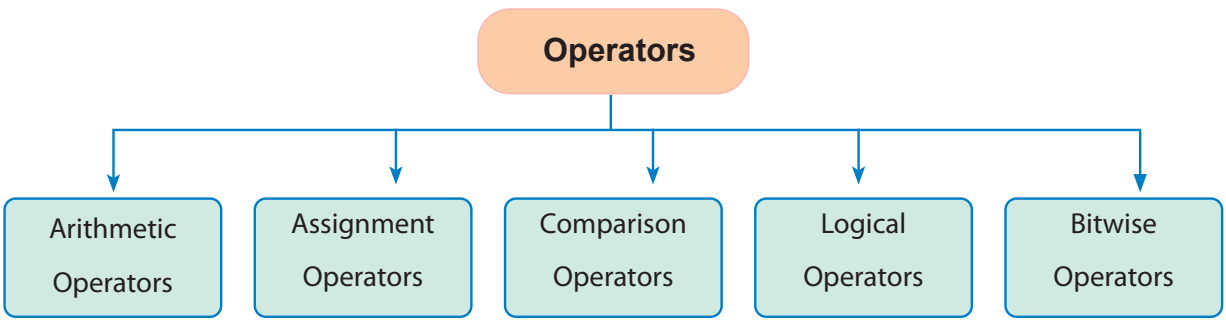


Figure (68): Python operators

Figure (68) shows the different categories of operators in Python. We will discuss each of these.

Arithmetic Operators:

These operators are responsible for any arithmetic operations such as addition, subtraction, multiplication, division, and so on. Table (7) shows the different Python arithmetic operators.

Table (7): Arithmetic operators

Operator	Name	Description	Example
+	Addition	It adds together two values.	x + y
-	Subtraction	It subtracts one value from another.	x - y
*	Multiplication	It multiplies two values.	x * y
/	Division	It divides one value by another.	x / y
%	Modulus	It returns the division remainder.	x % y
//	Floor Division	It divides the left operand by the right operand and returns the integer part of the result.	x//y
--	Exponentiation	It raises the left operand to the power of the right operand.	x**y

Example

```
a = 10
b = 3

sum_ab = a + b  # 13
difference_ab = a - b  # 7
product_ab = a * b  # 30
division_ab = a / b  # 3.3333...
floor_div_ab = a // b  # 3
remainder_ab = a % b  # 1
```

Figure (69): Arithmetic operators

Exercise (34)

Write a Python program that applies all the arithmetic operators to two numbers:

Answer:

```
# Define two numbers
num1 = 10
num2 = 4

# Addition
addition_result = num1 + num2
print(f"{num1} + {num2} = {addition_result}")

# Subtraction
subtraction_result = num1 - num2
print(f"{num1} - {num2} = {subtraction_result}")
```

```
# Multiplication
multiplication_result = num1 * num2
print(f"{num1} * {num2} = {multiplication_result}")

# Division
division_result = num1 / num2
print(f"{num1} / {num2} = {division_result}")

# Integer Division (Floor Division)
floor_division_result = num1 // num2
print(f"{num1} // {num2} = {floor_division_result}")

# Modulus (Remainder)
modulus_result = num1 % num2
print(f"{num1} % {num2} = {modulus_result}")

# Exponentiation
exponentiation_result = num1 ** num2
print(f"{num1} ** {num2} = {exponentiation_result}")
```

When you run this program, it will perform addition, subtraction, multiplication, division, integer division, modulus (remainder), and exponentiation on "num1" and `num2` and print the results.

Assignment Operators:

The assignment operators are responsible for assigning values to the left-hand side of these operators. The basic operators are the first six rows in table (8).

Table (8): Assignment operators

Operator	Example	Same as
=	x =5	x =5
=+	x + = 3	x = x + 3
=-	x - = 3	x = x - 3
=*	x * = 3	x = x * 2
=/	x / = 3	x = x / 3
=//	x // = 3	x = x // 3
=%	x % =3	Y =x
=**	x **= 3	x = x ** 3

Note (17)

Incrementing and Decrementing Variables in Python:

- In Python, there is no ++ operator for incrementing variables, nor is there i++ or ++i as seen in some other languages. To increment a variable by 1, you should use += 1, and to decrement it by 1, you should use - = 1
- Python's syntax is different from languages that employ ++ and -- operators. Understanding this distinction is essential when working with variables in Python.

Example

```
# Initialize a variable
count = 0

# Increment the variable by 1
count += 1

# Display the result
print("Incremented count:", count)

# Decrement the variable by 1
count -= 1

# Display the result
print("Decremented count:", count)
```

Comparison Operators:

The assignment operators are responsible for assigning values to the left-hand side of these operators. The basic operators are the first six rows in table (9).

Table (9): Comparison operators

Operator	Example	Same as
==	Equal to	x ==y
=!	Not equal	x ! = y
<	Greater than	x > y
>	Less than	x < y
=<	Greater than or equal to	x >= y
=>	Less than or equal to	x <= y

Example

```
x = 5
y = 10

is_equal = x == y  # False
is_not_equal = x != y  # True
is_less_than = x < y  # True
is_greater_than = x > y  # False
is_less_or_equal = x <= y  # True
is_greater_or_equal = x >= y  # False
```

Figure (70): Comparison operators

Logical Operators:

Logical operators in table (10) are responsible for the logical comparison, like in logic gates and, or, and not.

Table (10): Logical operators

Operator	Name	Description	Example
and	Logical and	It returns true if both statements are true.	x < 5 and x < 10
or	Logical or	It returns true if one of the statements is true.	x < 5 or x < 4
not	Logical not	Reverse the result, it returns false if the result is true.	(not(x < 5 and x < 10

Example

```
p = True
q = False

result_and = p and q  # False
result_or = p or q  # True
result_not = not p  # False
```

Figure (71): Logical operators

Bitwise Operators:

The operators in table (11) are working with the integer values only on the bit level.

Table (11): Bitwise operators

Operator	Description
&	Bitwise AND Operator
|	Bitwise OR Operator
^	Bitwise XOR Operator
~	Bitwise Complement Operator
>>	Bitwise Shift Left Operator
<<	Bitwise Shift Right Operator

Example

- **Write a Python program that reads two numbers from the user and then displays the following:**

1. The summation of the two numbers.
2. The difference between the first number and the second number.
3. The difference between the second number and the first number.
4. The multiplication between the two numbers.
5. The division between the first number and the second number (the first divided by the second).
6. The division between the second number and the first number (the second divided by the first).
7. The remainder of the first number is divided by the second number.
8. The remainder of the second number is divided by the first number.
9. The increment of the first number.
10. The decrement of the first number.
11. The increment of the second number.
12. The decrement of the second number.

Answer:

```
# Input two numbers from the user
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Perform calculations
summation = num1 + num2
diff1 = num1 - num2
diff2 = num2 - num1
multiplication = num1 * num2

# Check if the second number is not zero before performing division
if num2 != 0:
    division1 = num1 / num2
else:
    division1 = "Cannot divide by zero"

# Check if the first number is not zero before performing division
if num1 != 0:
    division2 = num2 / num1
else:
    division2 = "Cannot divide by zero"

# Check if the second number is not zero before performing remainde operation
if num2 != 0:
    remainder1 = num1 % num2
else:
    remainder1 = "Cannot divide by zero"

# Check if the first number is not zero before performing remainder operation
if num1 != 0:
    remainder2 = num2 % num1
else:
    remainder2 = "Cannot divide by zero"

# Increment and decrement operations
increment_num1 = num1 + 1
decrement_num1 = num1 - 1
increment_num2 = num2 + 1
decrement_num2 = num2 - 1

# Display results
print(f"1. Summation: {summation}")
print(f"2. Difference (1st - 2nd): {diff1}")
print(f"3. Difference (2nd - 1st): {diff2}")
print(f"4. Multiplication: {multiplication}")
print(f"5. Division (1st / 2nd): {division1}")
print(f"6. Division (2nd / 1st): {division2}")
print(f"7. Remainder (1st % 2nd): {remainder1}")
print(f"8. Remainder (2nd % 1st): {remainder2}")
print(f"9. Increment of 1st number: {increment_num1}")
print(f"10. Decrement of 1st number: {decrement_num1}")
print(f"11. Increment of 2nd number: {increment_num2}")
print(f"12. Decrement of 2nd number: {decrement_num2}")
```

Example

Write a Python program that reads two numbers from a user and then displays the following:

1. Add 9 to the number.
2. Subtract 9 from the number.
3. Multiply the number by 2
4. Divide the number over 3
5. Find the remainder of dividing the number over 5
6. Apply and perform an operation between the number and the number 3
7. Apply or perform an operation between the number and the number 3
8. Shift the input value by 2 to the right.
9. Shift the input value by 2 to the left.

Answer:

```
# Input a number from the user
num = float(input("Enter a number: "))

# Perform operations
addition_result = num + 9
subtraction_result = num - 9
multiplication_result = num * 2

# Check if the input number is not zero before performing division
if num != 0:
    division_result = num / 3
else:
    division_result = "Cannot divide by zero"
```

```
# Check if the input number is an integer before performing modulus operation
if num.is_integer():
    remainder_result = int(num) % 5
else:
    remainder_result = "Input is not an integer"

# Bitwise AND and OR operations
and_result = int(num) & 3
or_result = int(num) | 3

# Bitwise right and left shift operations
shift_right_result = int(num) >> 2
shift_left_result = int(num) << 2

# Display results
print(f"1. Add 9: {addition_result}")
print(f"2. Subtract 9: {subtraction_result}")
print(f"3. Multiply by 2: {multiplication_result}")
print(f"4. Divide by 3: {division_result}")
print(f"5. Remainder of dividing by 5: {remainder_result}")
print(f"6. Bitwise AND with 3: {and_result}")
print(f"7. Bitwise OR with 3: {or_result}")
print(f"8. Bitwise Right Shift by 2: {shift_right_result}")
print(f"9. Bitwise Left Shift by 2: {shift_left_result}")
```

This program takes a number as input from the user and then performs the specified operations, including addition, subtraction, multiplication, division, remainder, bitwise AND, bitwise OR, bitwise right shift, and bitwise left shift, and displays the results. Note that for some operations, we have added checks to handle cases where the input may not be suitable for the operation (e.g., division by zero or non-integer input for modulus).

Example

Write a Python program that reads two numbers from a user and then displays the following:

1. If the first number is equal to the second number.
2. If the first number is not equal to the second number, multiply the number by 2
3. If the first number is greater than the second number.
4. If the first number is less than the second number.
5. If the first number is greater than or equal to the second number.

Answer:

```
# Read two numbers from the user
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Perform comparisons
if num1 == num2:
    print("1. The first number is equal to the second number.")
else:
    print("1. The first number is not equal to the second number.")

    # Multiply the number by 2 if they are not equal
    num1 *= 2
```

```
if num1 > num2:
    print("3. The first number is greater than the second number.")

if num1 < num2:
    print("4. The first number is less than the second number.")

if num1 >= num2:
    print("5. The first number is greater than or equal to the second number

if num1 <= num2:
    print("6. The first number is less than or equal to the second number.")
```

This program takes two numbers as input and performs the specified comparisons, displaying the corresponding messages based on the conditions. If the first number is not equal to the second number, it multiplies the first number by 2, as indicated in point (2).

Example

Write a Python program that reads a number from a user and then displays the following:

1. If the number is greater than 10 and less than 20
2. If the number is greater than 10 or less than 20
3. If the number is not greater than 10 and less than 20
4. If the number is not greater than 10 or less than 20

Answer:

```
# Input a number from the user
num = float(input("Enter a number: "))

# Check the conditions
condition_1 = num > 10 and num < 20
condition_2 = num > 10 or num < 20
condition_3 = not (num > 10 and num < 20)

# Display results
print(f"1. Is the number greater than 10 and less than 20? {condition_1}")
print(f"2. Is the number greater than 10 or less than 20? {condition_2}")
print(f"3. Is the number not greater than 10 and less than 20? {condition_3}")
```

This program takes a number as input from the user and then evaluates the specified conditions using logical operators (`and`, `or`, `not`) and displays the results accordingly.

13 Conditional Statements

Conditional statements allow you to make decisions in your Python programs. In this chapter, we'll explore how to use `if`, `elif`, and `else` statements for decision-making.

Using `if` Statements for Decision-Making:

The `if` statement is used to execute a block of code if a condition is true.

Here's the basic structure:

```
if condition:
    # Code to execute if the condition is true
```

Figure (72): `if` statement

Example

```
age = 18

if age >= 18:
    print("You are an adult.")
```

Figure (73): Example of `if` statement

In this example, the code inside the `if` block is executed because the condition "age >= 18" is true.

Adding Multiple Conditions with `elif`:

You can use the `elif` (short for "else if") statement to specify additional conditions to check `if` the initial if condition is false. Here's how it works:

```
if condition1:
    # Code to execute if condition1 is true
elif condition2:
    # Code to execute if condition2 is true
```

Figure (74): `elif` statement

Example

```
grade = 85

if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
else:
    print("D")
```

Figure (75): Example of `elif` statement

In this example, the program checks multiple conditions and prints the corresponding grade based on the grade variable.

Handling Cases with `else`:

The `else` statement is used to execute a block of code when none of the preceding conditions are true. It provides a fallback option. Here's how it's used:

```
if condition:
    # Code to execute if the condition is true
else:
    # Code to execute if the condition is false
```

Figure (76): Handling cases with `else`

Example

```
age = 15

if age >= 18:
    print("You are an adult.")
else:
    print("You are not yet an adult.")
```

Figure (77): Example of handling cases with `else`

In this example, if the `age` is less than 18, the code inside the `else` block is executed.

Example

Write a program to show a message depending on the temperature degree, given that the temperature is 19°C. The output message should be "It's a cold weather". When the degree is less than 20°C, otherwise, it should be "It's a nice weather".

```
# Input the temperature in Celsius
temperature = float(input("Enter the temperature in Celsius: "))

# Check the temperature and display a message
if temperature < 20:
    print("It's a cold weather")
else:
    print("It's a nice weather")
```

The console output:

```
It's a nice weather
```

Exercise (35)

Apply the same previous example, but with the temperature as an input, so the temperature can be any number.

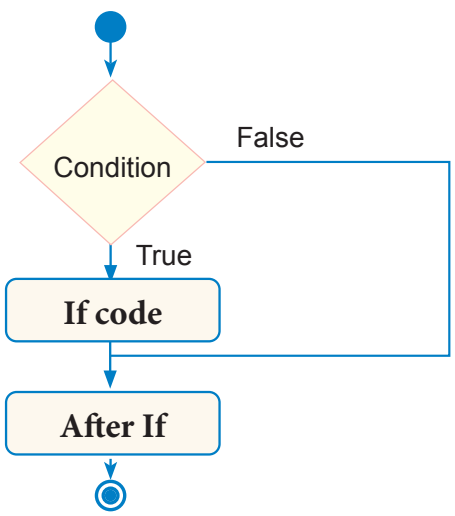


Figure (78): If-else statements flowchart

Answer:

```
# Define a condition (you can replace this with your specific condition)
condition = True

# Check the condition
if condition:
    print("Condition is true: Inside the 'if' block")
else:
    print("Condition is false: Inside the 'else' block")

print("Code execution continues after the 'if' statement")
```

Note (18)

1. In Python, an `if` statement can have zero or one `else` clause, and it must come after any `elif` (short for “else if”) clauses if present.
2. An `if` statement in Python can have zero to many `elif` clauses (short for “else if”), and they must come before the `else` clause if there is one.
3. In Python, once an `elif` statement succeeds (its condition evaluates to `True`), none of the remaining `elif` clauses or the `else` clause, if present, will be tested. This means that only the first `elif` (if any) that evaluates to `True` will be executed.

Example

Write a Python program code to get the greatest number among three numbers: number 1, number 2, and number 3 with values of 10, 50, and 5.5, respectively.

```
# Define the three numbers
number1 = 10
number2 = 50
number3 = 5.5

# Find the greatest number using the max() function
greatest_number = max(number1, number2, number3)

# Display the result
print(f"The greatest number among {number1}, {number2}, and {number3} is:
{greatest_number}")
```

The console output:

```
The greatest number among 10, 50, and 5.5 is: 50
```

Exercise (36)

Apply the same previous example, but with reading the three numbers (number 1, number 2, and number 3) as an input.

Answer:

```
# Input three numbers from the user
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
number3 = float(input("Enter the third number: "))

# Find the greatest number using the max() function
greatest_number = max(number1, number2, number3)

# Display the result
print(f"The greatest number among {number1}, {number2}, and {number3} is:
{greatest_number}")
```

This program prompts the user to enter three numbers, converts them to float values (assuming decimal input), and then finds and displays the greatest number among the three using the `**max()**` function.

Exercise (37)

Define Python variables to store four grades for students and print the corresponding letter describing the grade. For example, A (90–100) and B (89–80).

Answer:

```
# Define variables to store student grades
grade1 = 95
grade2 = 85
grade3 = 75
grade4 = 60

# Determine the letter grades based on the grade ranges
def get_letter_grade(score):
    if 90 <= score <= 100:
        return 'A'
    elif 80 <= score < 90:
        return 'B'
    elif 70 <= score < 80:
        return 'C'
    elif 60 <= score < 70:
        return 'D'
    else:
        return 'F'

# Print the corresponding letter grades for each student
print(f"Student 1 got an {get_letter_grade(grade1)}")
print(f"Student 2 got a {get_letter_grade(grade2)}")
print(f"Student 3 got a {get_letter_grade(grade3)}")
print(f"Student 4 got a {get_letter_grade(grade4)}")
```

In this program, we define four variables to store student grades. We also define a `**get_letter_grade()**` function that takes a numeric score as input and returns the corresponding letter grade based on the provided grade ranges (A, B, C, D, or F). We then print the letter grades for each student using this function.

Example

Write a Python program using the switch statement to print the equivalent grade name to the given grade character. Assume that the variable is called grade and has the value 'E'. The table contains the different grades. Note: This is an assumed table.

The output is the equivalent name of this grade.

Grade Character	Grade Name
S	Satisfactory
E	Excellent
F	Fail
P	Pass

```
# Define a dictionary to map grade characters to their equivalent names
grade_names = {
    'S': 'Satisfactory',
    'E': 'Excellent',
    'F': 'Fail',
    'P': 'Pass'
}

# Prompt the user to input a grade character
grade = input("Enter a grade character (S, E, F, or P): ")

# Check if the grade character exists in the dictionary
if grade in grade_names:
    # Retrieve and print the equivalent name
    equivalent_name = grade_names[grade]
    print(f"The equivalent name of grade {grade} is: {equivalent_name}")
else:
    print("Grade not found in the table. Please enter a valid grade character.")
```

Output:

```
Enter a grade character (S, E, F, or P): S
The equivalent name of grade S is: Satisfactory
```

14 Loops

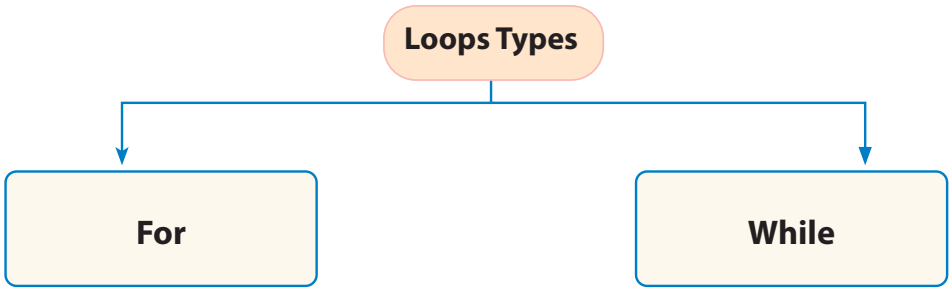


Figure (79): Loop types

Figure (79) shows the different types of loops in the Python programming language, which are for and while loops.

Using for Loops to Iterate over Sequences

A `for` loop is used to iterate over a sequence (e.g., a list, string, tuple, or range) and execute a block of code for each item in the sequence. Here's the basic structure:

```
for item in sequence:
    # Code to execute for each item
```

Figure (80): Using for loops to iterate over sequences

Example

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

Figure (81): Example of using for loops to iterate over sequences

In this example, the `for` loop iterates through the `fruits` list and prints each item.

Example

Write a Python program to print numbers from 1 to 100 using the `for` loop.

```
a for loop to print numbers from 1 to 100
er in range(1, 101):
t(number)
```

In this program, we use the range() function to generate a sequence of numbers from 1 to 100, and then we iterate through this sequence using a for loop. Inside the loop, we print each number. This will output numbers from 1 to 100, one per line.

Employing While Loops for Conditional Repetition

A `while` loop is used for conditional repetition. It continues executing a block of code as long as a specified condition is true. Here's the basic structure:

```
while condition:
    # Code to execute while the condition is true
```

Employing while loops for conditional repetition

Example

```
count = 0

while count < 5:
    print(count)
    count += 1
```

Figure (82): Example of employing while loops for conditional repetition

In this example, the `while` loop continues as long as the `count` is less than 5, printing the current value of `count` in each iteration.

Note (19)

If there is no termination condition, a while loop in Python will continue to execute indefinitely.

Example

Write a program to print numbers from 1 to 100 using the **`while`** loop.

```
# Initialize a variable to start at 1
number = 1

# Use a while loop to print numbers from 1 to 100
while number <= 100:
    print(number)
    number += 1
```

In this program:

1. We initialize a variable `number` to start at 1
2. We use a `while` loop to continue executing as long as `number` is less than or equal to 100
3. Inside the loop, we print the value of `number` and then increment it by 1 using `number += 1`.
4. The loop will continue until `number` reaches 101, at which point the condition `number <= 100` becomes `False`, and the loop terminates.

This program will print numbers from 1 to 100, one per line.

Exercise (38)

Write a program that reads a number from the user and then prints the numbers from 1 to the entered number using a while loop.

Answer:

```
# Read a number from the user
number = int(input("Enter a number: "))

# Initialize a variable to start at 1
count = 1

# Use a while loop to print numbers from 1 to the entered number
while count <= number:
    print(count)
    count += 1
```

In this program:

1. We first prompt the user to enter a number and use `**int(input(...))**` to read an integer value from the user.
2. We initialize a variable `**count**` to start at 1
3. We use a `**while**` loop to continue executing as long as `**count**` is less than or equal to the entered number.
4. Inside the loop, we print the value of `**count**` and then increment it by 1 using `**count += 1**`.
5. The loop will continue until `**count**` reaches the entered number, at which point the condition **count** `**<= number**` becomes `**False**`, and the loop terminates.
6. The program will print numbers from 1 to the entered number, one per line.

Loop Control Statements: "break"

The `**break**` statement is used to exit a loop prematurely, typically when a certain condition is met. It allows you to terminate the loop and continue with the rest of the program.

Example

```
numbers = [1, 2, 3, 4, 5]

for number in numbers:
    if number == 3:
        break
    print(number)
```

Figure (83): Using a for loop with a break statement

In this example, when the loop encounters the number `3`, the `break` statement is executed, and the loop terminates.

Loop Control Statements: "continue"

The `continue` statement is used to skip the current iteration of a loop and proceed to the next iteration. It allows you to skip specific items or conditions within the loop.

Example

```
numbers = [1, 2, 3, 4, 5]

for number in numbers:
    if number == 3:
        continue
    print(number)
```

Figure (84): Using a for loop with a continue statement

In this example, when the loop encounters the number `3`, the `continue` statement is executed, and the loop skips printing it.

Loops are powerful tools for automating repetitive tasks and iterating over data. The ability to control the flow of loops using `break` and `continue` statements provides fine-grained control over loop execution. In the following chapters, we'll delve into data structures like lists and dictionaries and explore more advanced topics in Python programming.

15 Object-Oriented Programming

The Object-Oriented Programming (OOP) is a programming paradigm that relies on object concepts. Each object comprises data and code. Object data is known as attributes, which define the properties of this object. While object code represents the functionality that this object can perform, object-oriented programming follows a set of concepts: inheritance, polymorphism, abstraction, and encapsulation.

a Object-Oriented Programming Concepts

As shown in figure (85) the hierarchy of object-oriented programming (OOP) concepts.

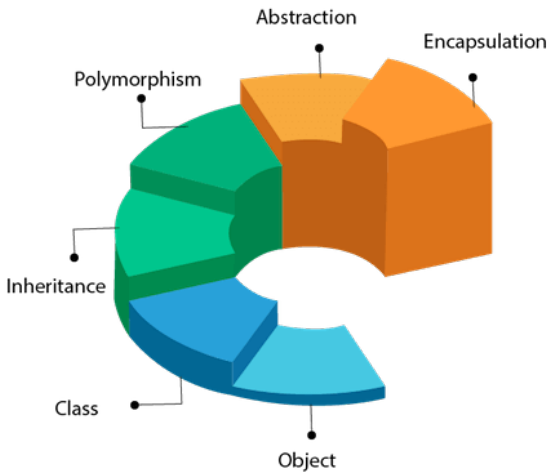


Figure (85): Object-oriented programming concepts

Object:

The object in the OOP is the thing that has features and functionality. In other words, the object is the stone that is used in building an application. The object is an instance of a class. Each object has three main characteristics, which are identity, state, and behavior. Object identity is the name of this object within the system. The object state refers to the value of the variables within this object. Object behaviors are the actions that this object can take.

Note (20)

The object, with its attributes and associated functions, is combined into something called a class.

Example

A car is an example of an object. The car has a set of variables like color, speed, acceleration, number of seats, etc. A car object can be instance of vehicle class.

Exercise (39)

What are the main characteristics of an object in the OOP?

Answer:

In Python, as in most object-oriented programming (OOP) languages, objects are the fundamental building blocks of the OOP. Objects have several main characteristics:

1. **State:** Objects store their data, often referred to as attributes or properties. These attributes represent the object's current state. In Python, attributes are typically defined as variables within the object's class.
2. **Behavior:** Objects can perform actions, and these actions are represented by methods. Methods are functions defined within the object's class that can operate on the object's attributes and modify its state. Methods define the object's behavior.
3. **Identity:** Each object has a unique identity, which is typically represented by its memory address or reference. Two objects may have the same state and behavior but are considered distinct if they have different identities.
4. **Encapsulation:** It is the concept of bundling an object's state (attributes) and behavior (methods) together into a single unit called a class. This promotes data hiding and allows for access control over an object's attributes.
5. **Abstraction:** It involves simplifying complex reality by modeling objects based on real-world entities. It focuses on essential properties and behaviors while hiding unnecessary details. Abstraction allows you to create classes that represent high-level concepts without exposing low-level implementation details.
6. **Inheritance:** It is a mechanism that allows one class (subclass or derived class) to inherit the attributes and methods of another class (superclass or base class). It promotes code reuse and the creation of hierarchies of related classes.
7. **Polymorphism:** It allows objects of different classes to be treated as objects of a common base class. It enables you to write more generic code that can work with

objects of various types. Polymorphism is often achieved through method overriding and interfaces.

8. **Dynamic Binding:** In Python, object references are dynamic, meaning that the type of an object can be determined at runtime. This allows for flexibility in programming, as objects can be used in a more dynamic and adaptable manner.

These characteristics collectively define the essence of objects in the OOP. Objects enable the modeling of real-world entities, encapsulate data and behavior, promote code organization and reusability, and contribute to the modular and maintainable design of software systems.

Class:

The class in the OOP is a user-defined type that represents an object. A class description consists of attributes and methods of objects. An object is an instance of a class. Several objects can be made of one class, in which all the instances will have the same attributes and functionalities with different identities.

Example

Assuming that you want to make an application that stores and tracks students' grades, a class named students, for example, needs to be created, and another class named students' details and associated functionalities also needs to be created.

Inheritance:

Inheritance features mean that a class can use code from another one. Inheritance in programming is very likely to result in inheritance in real life; for example, a child can inherit a set of personal traits from his parents. From a programming perspective, if a class has a set of features, then the children will inherit these features as well.

Example

If we have a class named shape and we have several classes that inherit from the shape class, each child class, instead of being close, will have a method to calculate the area of the shape from the other one.

Exercise (40)

What is meant by inheritance in the OOP?

Answer:

Inheritance is a fundamental concept that enables one class (subclass) to inherit attributes and methods from another class (superclass). The key points about inheritance are:

1. **Code Reusability:** Inheritance promotes code reuse by allowing subclasses to inherit attributes and methods from a superclass.
2. **Superclass-Subclass Relationship:** The superclass is the class being inherited from, and the subclass inherits its attributes and methods.
3. **Subclass Extension:** Subclasses can add new attributes and methods or override existing ones inherited from the superclass, enabling specialization and customization.
4. **Method Overriding:** Subclasses can override methods from the superclass to provide new implementations.
5. **Hierarchical Structure**: Inheritance creates class hierarchies, organizing classes in a tree-like structure with a common base class.
6. **Access to Superclass Members:** Subclasses can access superclass members using the `super()` keyword, allowing them to use or extend the superclass's behavior.

O Polymorphism:

Polymorphism is one of the fundamental concepts in the OOP. Polymorphism describes a situation in which different classes can perform the same function in different ways.

Example

If we have a class named shape and two objects, a circle and a triangle, the class shape has a function named calculate-area; this function should behave differently for each object based on its nature.

Abstraction:

Abstraction in the OOP means that objects share only necessary data and functionality with the rest of the code. All unnecessary implementations should be kept private for each object. This concept enables system expansion and changes very easily.

Exercise (41)

What is meant by abstraction in the OOP?

Answer:

In object-oriented programming (OOP), abstraction is a fundamental concept that involves simplifying complex systems or entities by modeling them based on their essential characteristics while hiding unnecessary details. Abstraction allows you to create a high-level representation of an object, concept, or system, focusing on what is relevant and important for a particular context.

Encapsulation:

Encapsulation in the OOP means that all the information is contained in an object, and only a set of information can be shared with the rest of the code. This concept ensures security and prevents data corruption.

Exercise (42)

What is meant by encapsulation in the OOP?

Answer:

In object-oriented programming (OOP), encapsulation is one of the fundamental principles that involves bundling data (attributes or properties) and the methods (functions or behaviors) that operate on that data into a single unit called a class. Encapsulation restricts direct access to some of an object's components, emphasizing the concept of data hiding. It allows you to control access to the internal state of an object, ensuring that the data remains consistent and valid.

b Advantages of Structured Programming

1. Code reuse is simplified by using objects.
2. The modularity of objects means that they can be easily developed, updated, and shared.
3. Debugging is easier when objects are in use and a coding issue arises.
4. Objects can be deployed without revealing the details of their implementations.

Exercise (43)

1. Based on your knowledge, define object-oriented programming.
2. List and define the main concepts in object-oriented programming.
3. List some of the advantages of object-oriented programming.

Answer:

1. Object-Oriented Programming (OOP):

- The OOP is a programming paradigm that organizes code into objects, which are instances of classes.
- Objects encapsulate data (attributes) and behaviors (methods) related to a specific entity or concept.
- The OOP promotes modularity, reusability, and abstraction, allowing for more structured and maintainable code.

2. Main Concepts in the OOP:

- **Class:** A blueprint or template for creating objects. It defines the attributes and methods that objects of the class will have.
- **Object:** An instance of a class, representing a real-world entity or concept. Objects have states (attributes) and behaviors (methods).
- **Inheritance:** The mechanism that allows a class (subclass) to inherit attributes and methods from another class (superclass), promoting code reuse and hierarchy.
- **Polymorphism:** The ability of objects of different classes to respond to the same method or function name in a way appropriate to their types.
- **Encapsulation:** The concept of bundling data (attributes) and methods (behaviors) into a single unit (class) and controlling access to them.

3. Advantages of the OOP:

- **Modularity:** Code is organized into self-contained objects and classes, making it easier to manage and understand.
- **Reusability:** Objects and classes can be reused in different parts of the program, reducing redundant code.
- **Abstraction:** Complex systems can be represented at a high level, hiding implementation details.
- **Encapsulation:** Data and methods are encapsulated within objects, improving data integrity and access control.
- **Inheritance:** It promotes code hierarchy and reusability by allowing classes to inherit from others.
- **Polymorphism:** It enables more flexible and adaptable code, as objects can respond to common interfaces.

Object-oriented programming enhances code organization, reusability, and abstraction, making it a powerful paradigm for building scalable and maintainable software systems.

16 Functional Programming

Functional Programming (FP) is another programming paradigm that complements Object-Oriented Programming (OOP) and has gained popularity in recent years. It focuses on using functions as first-class citizens and emphasizes immutability, pure functions, and declarative programming. In this section, we'll explore the key concepts and advantages of Functional Programming in Python.

a Concepts of Functional Programming

1. First-Class Functions

In FP, functions are treated as first-class citizens, meaning they can be assigned to variables, passed as arguments to other functions, and returned from functions.

This enables higher-order functions, which are functions that take other functions as arguments or return functions as results.

2. Immutability

In FP, data is immutable, which means once it's created, it cannot be changed. Instead, new data is created with each operation.

Immutable data structures like tuples and frozensets are often used.

3. Pure Functions

Pure functions are functions that always produce the same output for the same input, without side effects.

They do not modify external states, making code more predictable and testable.

Example

The pure function:

```
def sum(x, y):      # sum is a function taking x and y as arguments
    return x + y    # sum is returning the sum of x and y without changing
```

4. Recursion

Recursion is a common technique in FP for solving problems by breaking them down into smaller, similar subproblems.

Python supports recursion, but it's important to handle base cases properly to avoid infinite recursion.

Example

The recursive function:

```
def fib(n):
    if n <= 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Figure (86): Example of a recursive function

5. Higher-Order Functions

Higher-order functions take one or more functions as arguments or return functions as results.

Examples include `map()`, `filter()`, and `reduce()`.

6. Declarative Programming

FP encourages declarative programming, where you describe what you want to achieve rather than specifying how to achieve it.

This often leads to more concise and readable code.

b Advantages of Functional Programming

1. Readability and Maintainability:

FP promotes clear, concise, and self-contained functions, making code easier to read and maintain.

2. Predictability:

With pure functions and immutability, code becomes more predictable, reducing bugs and making debugging easier.

3. Concurrent and Parallel Programming:

FP is well-suited for concurrent and parallel programming, as immutable data and pure functions can be safely shared across threads or processes.

4. Testability:

Pure functions are easy to test since they only depend on their input, producing consistent results.

5. Reusable Code:

Functional programming encourages the creation of reusable functions, leading to more modular and reusable code.

6. Functional Libraries:

Python has libraries like `functools` and `itertools` that provide functional programming tools, enhancing Python's functional capabilities.

Incorporating functional programming concepts into your Python code can lead to more maintainable, reliable, and scalable software. It complements the object-oriented approach, allowing you to choose the best paradigm for each part of your application.

Exercise (44)

1. Based on your knowledge, define functional programming.
2. List and define the main concepts in functional programming.
3. List some of the advantages of functional programming.

1. Functional Programming (FP):

- Functional Programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing states and mutable data.
- In FP, functions are first-class citizens, meaning they can be assigned to variables, passed as arguments, and returned as results.
- FP promotes immutability, pure functions, and declarative programming.

2. Main Concepts in Functional Programming:

- **Pure Functions:** They are functions that always produce the same output for the same input and do not have side effects. They rely only on their input parameters and do not modify external states.
- **Immutability:** In FP, data is immutable, meaning once created, it cannot be changed. Instead, new data is created with each operation. This leads to more predictable code.
- **Higher-Order Functions:** They are functions that can take other functions as arguments or return functions as results. They enable powerful abstractions and transformations of data.
- **Function Composition:** It is the act of combining multiple functions to create a new function. It allows you to build complex behavior by composing simpler functions.
- **Recursion:** It is a technique in FP where a function calls itself to solve problems by breaking them down into smaller and similar subproblems.
- **Declarative Programming:** FP encourages declarative programming, where you describe what you want to achieve rather than specifying how to achieve it. This leads to more concise and readable code.
- **Lazy Evaluation:** In some FP languages, lazy evaluation is used, where expressions are not evaluated until their results are actually needed. This can improve efficiency in certain situations.

3. Advantages of Functional Programming:

- **Readability and Maintainability:** FP promotes clear, concise, and self-contained functions, making code easier to read and maintain.

- **Predictability:** Pure functions and immutability make code more predictable, reducing bugs and making debugging easier.
- **Concurrent and Parallel Programming:** FP is well-suited for concurrent and parallel programming, as immutable data and pure functions can be safely shared across threads or processes.
- **Testability:** Pure functions are easy to test since they only depend on their input, producing consistent results.
- **Reusable Code:** Functional programming encourages the creation of reusable functions, leading to more modular and reusable code.
- **Functional Libraries:** Python has libraries like `functools` and `itertools` that provide functional programming tools, enhancing Python's functional capabilities.

Functional programming is a powerful paradigm for building robust, maintainable, and scalable software by focusing on clear, composable, and predictable functions. It complements other programming paradigms, like object-oriented programming, allowing you to choose the best approach for different parts of your application.

17 Lists

Lists are one of the fundamental data structures in Python, allowing you to store and manipulate collections of data. In this chapter, we will explore how to create, access, modify, and iterate through lists.

Creating Lists:

You can create lists by enclosing a sequence of values in square brackets `[]`, separated by commas. Lists can contain elements of different data types.

Example

```
fruits = ["apple", "banana", "cherry"]
numbers = [1, 2, 3, 4, 5]
mixed = ["apple", 42, True]
```

Figure (87): Creating lists

Accessing List Elements:

List elements can be accessed using their index. Python uses zero-based indexing, meaning the first element has an index of 0, the second has an index of 1, and so on.

Example

```
fruits = ["apple", "banana", "cherry"]

first_fruit = fruits[0]  # "apple"
second_fruit = fruits[1]  # "banana"
```

Figure (88): Accessing list elements

You can also use negative indexing to access elements from the end of the list:

```
last_fruit = fruits[-1]  # "cherry"
```

Figure (89): Using indexing to access elements

Adding Elements with "append":

The "append()" method is used to add an element to the end of a list.

Example

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
# fruits is now ["apple", "banana", "cherry", "orange"]
```

Figure (90): Adding elements with `append`

Inserting Elements with `insert`:

The `insert()` method allows you to add an element at a specific position in the list.

Example

```
fruits = ["apple", "banana", "cherry"]
fruits.insert(1, "orange")
# fruits is now ["apple", "orange", "banana", "cherry"]
```

Figure (91): Inserting elements with `insert`

Removing Elements with remove

The `remove()` method is used to remove the first occurrence of a specific value from the list.

Example

```
fruits = ["apple", "banana", "cherry"]
fruits.remove("banana")
# fruits is now ["apple", "cherry"]
```

Figure (92): Removing Elements with `remove`

Iterating Through Lists Using Loops:

You can iterate through the elements of a list using `for` loops. This allows you to perform operations on each element.

Example

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

Figure (93): Iterating through lists using loops

This code will print each fruit in the `fruits` list.

Exercise (45)

Create a Python list named `fruits` containing the names of three different fruits. Then, print the second fruit in the list.

Answer:

```
# Create the list of fruits
fruits = ["apple", "banana", "orange"]

# Print the second fruit in the list (index 1)
print(f"The second fruit is: {fruits[1]}")
```

Exercise (46)

You have a list called `colors` containing the names of colors. Add the color “green” to the end of the list, insert the color “red” at the beginning of the list, and remove the color “blue” if it exists.

Answer:

```
# Initial list of colors
colors = ["blue", "yellow", "orange"]

# Add "green" to the end of the list
colors.append("green")

# Insert "red" at the beginning of the list
colors.insert(0, "red")

# Remove "blue" if it exists
if "blue" in colors:
    colors.remove("blue")

# Print the modified list
print("Updated list of colors:", colors)
```

Exercise (47)

Given a list of `numbers` called numbers, use a `for` loop to print each number in the list on a separate line.

Answer:

```
# List of numbers
numbers = [10, 20, 30, 40, 50]

# Iterate through the list and print each number
for number in numbers:
    print(number)
```

These exercises cover creating and accessing lists, modifying lists using `**append**`, `**insert**`, and `**remove**`, and iterating through lists using `**for**` loops.

Lists are versatile data structures that are commonly used in Python for storing and manipulating collections of data. Understanding how to create, access, modify, and iterate through lists is essential for many programming tasks. In the following chapters, we'll explore more advanced data structures and concepts in Python.

18 Tuples and Dictionaries

Tuples and dictionaries are two essential data structures in Python that provide different ways to store and manage data. In this chapter, we will explore tuples, which are immutable, and dictionaries, which are mutable and used for key-value pair storage.

Creating Tuples:

Tuples are similar to lists but are immutable, meaning their elements cannot be changed once defined. You create tuples by enclosing a sequence of values in parentheses `()`, separated by commas.

Example

```
coordinates = (3, 4)
fruits = ("apple", "banana", "cherry")
```

Figure (94): Creating tuples

Accessing Tuple Elements:

You can access tuple elements using indexing, similar to lists.

Example

```
coordinates = (3, 4)
fruits = ("apple", "banana", "cherry")
```

Figure (95): Accessing tuple elements

Tuples are often used to represent collections of related, immutable data.

Creating Dictionaries:

Dictionaries are used to store data in key-value pairs. You create dictionaries using curly braces `{}` and specifying key-value pairs.

Example

```
person = {"name": "Alice", "age": 30, "city": "New York"}
```

Figure (96): Creating dictionaries

Accessing Dictionary Values:

You can access dictionary values using their keys.

Example

```
person = {"name": "Alice", "age": 30, "city": "New York"}
name = person["name"]  # "Alice"
age = person["age"]  # 30
```

Figure (97): Accessing dictionary values

Adding Items:

You can add new key-value pairs to a dictionary by assigning a value to a new key.

Example

```
person = {"name": "Alice", "age": 30}
person["city"] = "New York"
# person is now {"name": "Alice", "age": 30, "city": "New York"}
```

Figure (98): Adding items to dictionaries

Updating Items:

To update the value of an existing key, simply reassign it.

Example

```
person = {"name": "Alice", "age": 30}
person["age"] = 31
# person is now {"name": "Alice", "age": 31}
```

Figure (99): Updating items of dictionaries

Deleting Items:

You can remove items from a dictionary using the `del` keyword.

Example

```
person = {"name": "Alice", "age": 30}
del person["age"]
# person is now {"name": "Alice"}
```

Figure (100): Deleting items from dictionaries

Dictionaries are highly versatile and useful for organizing and accessing data using meaningful keys.

Exercise (48)

Create a tuple named `days_of_week` containing the names of the days of the week. Attempt to change the name of the second day in the tuple to "Funday." Explain what happens and why.

Answer:

```
# Create a tuple of days of the week
days_of_week = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday",

# Try to change the name of the second day
# This will result in a TypeError because tuples are immutable
try:
    days_of_week[1] = "Funday"
except TypeError as e:
    print(f"Error: {e}")
```

Explanation:

Tuples are immutable in Python, which means their elements cannot be modified once they are defined. When you attempt to change the name of the second day using indexing, it raises a `TypeError` because you cannot assign a new value to an element in a tuple.

Exercise (49)

Create a dictionary named `student_info` containing the following information about a student: “name,” “age,” and “major.” Access and print the student’s name from the dictionary.

Answer:

```
# Create a dictionary containing student information
student_info = {
    "name": "Alice",
    "age": 20,
    "major": "Computer Science"
}

# Access and print the student's name
print(f"Student's name: {student_info['name']}")
```

Exercise (50)

You have a dictionary named `car_info` with information about a car. Add the following information to the dictionary:

- "model": "Toyota Camry"
- "year": 2022

Update the "year" to 2023, and then delete the "color" entry from the dictionary if it exists.

Answer:

```
# Initial dictionary containing car information
car_info = {
    "make": "Toyota",
    "color": "Blue"
}

# Add model and year to the dictionary
car_info["model"] = "Toyota Camry"
car_info["year"] = 2022

# Update the year to 2023
car_info["year"] = 2023
```

```
# Delete the "color" entry if it exists
if "color" in car_info:
    del car_info["color"]

# Print the modified dictionary
print("Updated car information:", car_info)
```

19 Case Study

A Temperature Conversion Project is a software system that is responsible for helping people know the air temperature. The air temperature is measured in two units, either Celsius (C) or Fahrenheit (F). Some countries depend on the F when measuring the air temperature, such as the United States, and other countries depend on the C, such as Egypt. Celsius (C) and Fahrenheit (F) are not equal in value. So, this software system will help you if you leave your country. You could know the air temperature regardless of the new location. This could occur by entering the temperature of the new country and asking for the temperature of your country.

Note (21)

Writing code is the final step.

To start developing any software system like this, you should follow some rules:

1. Reading the Case Study: Understanding the requirements of a project by reading the case study is essential in any programming language, including Python.

2. Listing Required Functionalities: Identifying and listing the required functionalities is a crucial step in the software development process, regardless of the programming language.

3. Defining Inputs and Outputs: Clearly defining inputs and outputs is fundamental for designing functions and methods, which you'll write in Python to process data.

4. Drawing Sequence Diagrams: Sequence diagrams can help visualize the flow of interactions in your system, but their use may vary depending on the project's complexity.

5. Dividing the Project into Milestones: Breaking the project into milestones is a good practice, especially for managing larger projects in Python or any other language.

6. Documentation: Documenting your project, including requirements, design decisions, and code, is crucial for maintaining and collaborating on software development, regardless of the programming language.

7. Structured Programming: While structured programming is a valid paradigm, Python is more known for supporting multiple paradigms, including procedural, object-oriented, and functional programming. You can choose the paradigm that best suits your project's requirements.

Celsius Formula

Celsius to Fahrenheit	$°F = \left(\frac{9}{5} \times °C\right) + 32$
Fahrenheit to Celsius	$°C = \frac{5}{9}(°F - 32)$

Exercise (51)

Write a program that reads a number from the user and then prints the numbers from 1 to the entered number using the do-while loop.

Answer:

```
# Initialize a counter
counter = 1

# Ask the user for input
try:
    user_input = int(input("Enter a number: "))
except ValueError:
    print("Invalid input. Please enter a valid number.")
    exit()

# Check if the input is non-negative
if user_input < 0:
    print("Please enter a non-negative number.")
else:
    # Use a while loop to print numbers from 1 to the entered number
    while counter <= user_input:
        print(counter)
```

In this program:

1. We initialize a counter variable to 1
2. We use a "try...except" block to handle potential non-integer inputs and ask the user to enter a number.
3. We check if the entered number is non-negative.
4. If it's non-negative, we use a "while" loop to print numbers from 1 to the entered number, incrementing the counter in each iteration.

This achieves the desired behavior of printing numbers in a "do-while" style loop in Python.

Exercise (52)

Write a program that allows the user to enter time in seconds and then outputs how far an object would drop if it were in freefall for that length of time. Assume that the object starts at rest, there is no friction or resistance from air, and there is a constant acceleration of 32 feet per second due to gravity. Use the equation:

$$\text{Distance} = \frac{\text{acceleration} \times \text{time}^2}{2}$$

You should first compute the product and then divide the result by 2

Answer:

```
# Constants
ACCELERATION = 32  # Acceleration due to gravity in feet per second^2

# Get user input for time in seconds
try:
    time_seconds = float(input("Enter the time in seconds: "))
except ValueError:
    print("Invalid input. Please enter a valid number for time.")
    exit()

# Calculate the distance using the formula
distance = (ACCELERATION * (time_seconds ** 2)) / 2

# Print the result
print(f"The object would drop approximately {distance:.2f} feet.")
```

In this program:

1. We define a constant `**ACCELERATION**` representing the acceleration due to gravity in feet per second squared.
2. We use a "try...except" block to handle potential non-numeric input for the time in seconds and ask the user to enter a valid number.
3. We calculate the distance using the provided formula: `distance = (1/2) * acceleration * time^2`.
4. We print the calculated distance with two decimal places.

The program takes the time in seconds as input and calculates the distance the object would drop in freefall under the specified conditions.

Exercise (53)

A store sells carpets for $2.75 per meter. If a customer buys more than 10 m of carpet, they get a discount of 15% on every additional meter of carpet they purchase. Write a program that inputs the carpet length that a user wishes to buy, stores the value in a double variable, and calculates and outputs the total cost of the carpet.

Answer:

```
# Define the price per meter of carpet
price_per_meter = 2.75

# Input: Get the carpet length in meters from the user
try:
    carpet_length = float(input("Enter the length of carpet you wish to buy (in meters): "))
except ValueError:
    print("Invalid input. Please enter a valid number for carpet length.")
    exit()

# Initialize the total cost
total_cost = 0

# Calculate the total cost, considering the discount for lengths over 10 meters
if carpet_length <= 10:
    total_cost = price_per_meter * carpet_length

else:
    # Calculate cost for the first 10 meters
    total_cost = price_per_meter * 10
    # Calculate cost for the additional meters with a 15% discount
    additional_length = carpet_length - 10
    discounted_cost = price_per_meter * additional_length * 0.85
    total_cost += discounted_cost

# Output the total cost of the carpet
print(f"The total cost of {carpet_length} meters of carpet is ${total_cost:.2f}")
```

Exercise (54)

Write a program that determines whether a meeting room violates fire law regulations regarding the maximum room capacity. The program will read the maximum room capacity and the number of people attending the meeting. If the number of people is less than or equal to the maximum room capacity, the program announces that it is legal to hold the meeting and tells how many additional people may legally attend. If the number of people exceeds the maximum room capacity, the program announces that the meeting cannot be held as planned due to fire regulations and tells how many people must be excluded to meet the fire regulations. For a harder version, write your program so that it allows the calculation to be repeated as often as the user wishes.

Answer:

```
while True:
    # Input: Get the maximum room capacity and number of people attending the meeting
    try:
        max_capacity = int(input("Enter the maximum room capacity: "))
        num_attendees = int(input("Enter the number of people attending the meeting: "))
    except ValueError:
        print("Invalid input. Please enter valid integers.")
        continue

    # Check if the meeting complies with fire regulations
    if num_attendees <= max_capacity:
        # Legal to hold the meeting
        additional_people = max_capacity - num_attendees
        print("It is legal to hold the meeting.")
        if additional_people > 0:
            print(f"You may legally allow {additional_people} additional people.")
        else:
            print("The room is at maximum capacity.")
    else:
        # Violates fire regulations
        people_excluded = num_attendees - max_capacity
        print("The meeting cannot be held as planned due to fire regulations.")
        print(f"You must exclude {people_excluded} people to meet the fire regulations.")

    # Ask if the user wants to repeat the calculation
    repeat = input("Do you want to calculate again? (yes/no): ").strip().lower()
    if repeat != "yes":
        break
```

In this program:

- We use a while loop to allow the user to repeat the calculation as many times as they want.
- We use a "try...except" block to handle potential non-integer inputs.
- We check if the number of people attending the meeting is within the maximum room capacity.
- Depending on the outcome, we provide the appropriate message and calculate the number of people to exclude.
- After each calculation, we ask the user if they want to repeat, and if they choose not to, the loop exits. Otherwise, the loop continues for another calculation.

Exercise (55)

Write a program that reads three integer values. The numbers should then be output in ascending order, from the smallest to the largest. Can you do this with only if statements and three integer variables (Hint: try nesting if statements or using the && operator)? What happens if you input three identical numbers?

Answer:

```
# Input: Get three integer values from the user
try:
    num1 = int(input("Enter the first integer: "))
    num2 = int(input("Enter the second integer: "))
    num3 = int(input("Enter the third integer: "))
except ValueError:
    print("Invalid input. Please enter valid integers.")
    exit()

# Initialize variables to hold the sorted values
min_num = num1
mid_num = num2
max_num = num3
```

```
# Sort the numbers in ascending order
if mid_num < min_num:
    min_num, mid_num = mid_num, min_num
if max_num < mid_num:
    mid_num, max_num = max_num, mid_num
if mid_num < min_num:
    min_num, mid_num = mid_num, min_num

# Output the sorted numbers
print("Numbers in ascending order:", min_num, mid_num, max_num)
```

In this program:

- We use a "try...except" block to handle potential non-integer inputs.
- We initialize three integer variables (`**num1**`, `**num2**`, and `**num3**`) to store the input values.
- We use conditional statements to compare and rearrange the values so that they are in ascending order. This is done using a series of "if" statements.
- The output displays the sorted numbers in ascending order.

If you input three identical numbers, the program will still work correctly because the "if" statements check for equality and will not change the order if the numbers are the same. In this case, the output will be three identical numbers in ascending order.

Exercise (56)

Write a program that reads int values from the user until they enter a negative number like 21. Once the user has finished entering numbers, print out the highest value they've entered, the lowest value they've entered, and the total number of numbers they've entered. The negative number they entered should not be taken as one of the values entered.

Answer:

```
# Initialize variables for highest, lowest, and total
highest = float('-inf')
lowest = float('inf')
total_numbers = 0

while True:
    try:
        # Input: Get an integer value from the user
        num = int(input("Enter an integer (negative to quit): "))
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
        continue

    # Check if the number is negative to exit the loop
    if num < 0:
        break

    # Update highest, lowest, and total
    if num > highest:
        highest = num
    if num < lowest:
        lowest = num
    total_numbers += 1

# Check if any numbers were entered
if total_numbers > 0:
    print(f"Highest value entered: {highest}")
    print(f"Lowest value entered: {lowest}")
    print(f"Total number of numbers entered (excluding negative): {total_numbers}")
else:
    print("No valid numbers were entered.")
```

In this program:

- We use a `**while**` loop to continuously read integer values from the user until they enter a negative number.
- Inside the loop, we check if the input is a valid integer using a "try...except" block.
- If the entered number is negative, we break out of the loop to stop input.
- We maintain variables (`**highest**`, `**lowest**`, and `**total_numbers**`) to keep track of the highest value, the lowest value, and the total number of valid numbers entered.
- After the loop, we check if any valid numbers were entered (excluding the negative number) and print the results.
- If no valid numbers were entered (i.e., the user only entered a negative number to quit), we inform the user accordingly.

Exercise (57)

Write a program to calculate the slope between two points x^1, y^1 and x^2, y^2. The points should be entered as four double values in the order x^1, y^1, x^2, and y^2. The formula to calculate the slope, m, between two points is:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Output the calculated slope value. Use this value to output the equation of the line in the form:

$$y = mx + c$$

You can calculate the value of c from one of the pairs of points entered as input.

Answer:

```
# Input: Get the four double values from the user
try:
    x1 = float(input("Enter x1: "))
    y1 = float(input("Enter y1: "))
    x2 = float(input("Enter x2: "))
    y2 = float(input("Enter y2: "))
except ValueError:
    print("Invalid input. Please enter valid double values.")
    exit()

# Calculate the slope (m)
if x2 - x1 != 0:
    m = (y2 - y1) / (x2 - x1)
else:
    print("The slope is undefined (division by zero).")
    exit()

# Calculate the value of c (y-intercept) using one of the points
c = y1 - m * x1

# Output the calculated slope and the equation of the line
print(f"Slope (m) = {m}")
print(f"Equation of the line: y = {m}x + {c}")
```

In this program:

- We use a "try...except" block to handle potential non-double (float) inputs.
- We calculate the slope (`**m**`) using the formula `**m = (y2 - y1) / (x2 - x1)**`. Note that we check if the denominator (`**x2 - x1**`) is not zero to avoid division by zero, which would result in an undefined slope.

- We calculate the value of the y-intercept (`**c**`) using one of the points (in this case, we use `**(x1, y1)**`).
- Finally, we output both the calculated slope and the equation of the line in the desired form.

Answer:

```
# Input: Get the four double values from the user
try:
    x1 = float(input("Enter x1: "))
    y1 = float(input("Enter y1: "))
    x2 = float(input("Enter x2: "))
    y2 = float(input("Enter y2: "))
except ValueError:
    print("Invalid input. Please enter valid double values.")
    exit()

# Calculate the slope (m)
if x2 - x1 != 0:
    m = (y2 - y1) / (x2 - x1)
else:
    print("The slope is undefined (division by zero).")
    exit()

# Calculate the value of c (y-intercept) using one of the points
c = y1 - m * x1

# Output the calculated slope and the equation of the line
print(f"Slope (m) = {m}")
print(f"Equation of the line: y = {m}x + {c}")
```

In this program:

- We use a “try...except” block to handle potential non-double (float) inputs.
- We calculate the slope (`**m**`) using the formula `**m = (y2 - y1) / (x2 - x1)**`. Note that we check if the denominator (`**x2 - x1**`) is not zero to avoid division by zero, which would result in an undefined slope.
- We calculate the value of the y-intercept (`**c**`) using one of the points (in this case, we use `**(x1, y1)**`).
- Finally, we output both the calculated slope and the equation of the line in the desired form.
- Finally, we output the result based on whether the number is prime or not.

Exercise (58)

Write a program in Python to find prime numbers within a range.

Answer:

```
import math

# Input: Get the range from the user
try:
    start = int(input("Enter the start of the range: "))
    end = int(input("Enter the end of the range: "))
except ValueError:
    print("Invalid input. Please enter valid integers for the range.")
    exit()

# Function to check if a number is prime
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False
    return True

# Find and print prime numbers within the specified range
print(f"Prime numbers between {start} and {end}:")
for num in range(start, end + 1):
    if is_prime(num):
        print(num, end=", ")

print()  # Print a newline for better formatting
```

In this program:

- We use a "try...except" block to handle potential non-integer inputs for the range.
- We define a function is_prime(num) that checks if a number is prime, similar to the previous example.
- We iterate through the numbers in the specified range using a for loop.
- For each number in the range, we call the is_prime function to check if it's prime.
- If a number is prime, we print it.

This program will find and print prime numbers within the specified range, including both the start and end values.

Exercise (59)

Write a program in Python to find the sum of digits of a given number.

Answer:

```python
# Input: Get an integer value from the user
try:
    num = int(input("Enter an integer: "))
except ValueError:
    print("Invalid input. Please enter a valid integer.")
    exit()

# Function to calculate the sum of digits
def sum_of_digits(number):
    # Initialize a variable to store the sum
    total = 0

    # Iterate through each digit in the number
    while number > 0:
        # Extract the last digit
        digit = number % 10

        # Add the digit to the total
        total += digit

        # Remove the last digit from the number
        number //= 10

    return total

# Calculate and output the sum of digits
result = sum_of_digits(num)
print(f"The sum of digits in {num} is {result}.")
```

In this program:

- We use a "try...except" block to handle potential non-integer inputs.
- We define a function `**sum_of_digits(number)**` to calculate the sum of digits of the given number.
- Inside the function, we initialize a variable `**total**` to store the sum of digits.
- We use a `**while**` loop to repeatedly extract the last digit from the number, add it to `**total**`, and remove it from the number until the number becomes zero.
- After the loop, we return the total sum.
- Finally, we calculate the sum of digits for the user-input number and print the result.

Assessment

Q 1. Define the different types of conditional statements.

Q 2. Choose the correct answer:

a) 2.50 belongs to the …….. data type.

1. int 2. float
3. char 4. None of the above

b) Which of the following methods can be used to determine the size of a variable in memory in terms of the number of bytes?

1. len() 2. sys.getsizeof()
3. size() 4. length()

Q 3. True or False:

a) It is possible to have multiple `if` statements within the same Python program. ( )

b) Is the following statement valid when defining a variable in Python:
`default=5;` ( )

c) Is the following statement valid when defining a character variable in Python:
`afloat = 's';` ( )

d) You can have multiple `elif` statements within an `if` condition in Python. ( )

Q 4. Mention two methods for writing a new line in Python language.

Q 5. Write the syntax of reading a user input and writing an output using Python language.

Q 6. Write a Python program that reads the student's first, and last name, then prints a message welcoming him/her as follows:

Input: Ahmed Ali.

Output: Welcome Ahmed Ali.

Q 7. Create a Python program that displays n natural number terms and their sum.

Sample Output:

Enter the number of terms: 7

The natural numbers are: up to the 7th term. 1 2 3 4 5 6 7

The total number of natural numbers is 28

Q 8. Factorial of any number n is represented by n! and is equal to 1*2*3*. *(n-1)*n.

For example:

4! = 1*2*3*4 = 24

3! = 3*2*1 = 6

2! = 2*1 = 2

Also, 1! = 1

0! = 0

Write a Python program to calculate the factorial of a number.

a) The input is a positive integer number.

b) The output is the factorial of this positive integer number.

Q 9. Read two non-negative numbers from the user and calculate the product of these two numbers. (Not allowed to use the * operator).

Q 10. What is the output of n1, n2, and i in the following block of code?

```
i = 0
n1, n2 = i, i + 1
i += 1
print(n1)
print(n2)
print(i)
```

Answer

Q 1. In Python, there are several types of conditional statements that allow you to control the flow of your program based on different conditions. The main types of conditional statements are:

1. **if Statement:** The basic conditional statement that allows you to execute a block of code if a condition is true.

```
if condition:
    # Code to be executed if the condition is true
```

2. **if-else Statement:** It extends the "if" statement by allowing you to specify an alternative block of code to execute if the condition is false.

```
if condition:
    # Code to be executed if the condition is true
else:
    # Code to be executed if the condition is false
```

3. **if-elif-else Statement:** It allows you to specify multiple conditions and their corresponding blocks of code. It executes the first block of code whose condition is true, or the "else" block if none of the conditions are true.

```
if condition1:
    # Code to be executed if condition1 is true
elif condition2:
    # Code to be executed if condition2 is true
else:
    # Code to be executed if no conditions are true
```

4. **Nested if Statements:** You can place one or more "if" statements inside another "if" statement to create complex conditional logic.

```
if condition1:
    if condition2:
        # Code to be executed if both condition1 and condition2 are true
```

5. **Ternary Conditional Operator:** Provides a concise way to write simple if-else statements in a single line.

```
result = value_if_true if condition else value_if_false
```

These conditional statements are fundamental for controlling the flow of your Python programs and making decisions based on different conditions. You can use them to create complex logic and make your programs more flexible and responsive to different situations.

Q 2.

a) 2.50 belongs to the **float** data type. So, the correct option is **2.float**.

b) To know the size of a variable in memory in terms of the number of bytes in Python, you can use the `**sys.getsizeof()**` method from the `**sys**` module.

Q 3.

a) True (It is possible to have multiple `**if**` statements within the same Python program.)

b) False (The statement `**default = 5;**` is not valid when defining a variable in Python. Variable names cannot start with a reserved word like `**default**`, and you don't need to specify the type like `**int**`.)

c) True (The statement `**afloat = 's';**` is valid when defining a character variable in Python.)

d) True (You can have multiple `**elif**` statements within an `**if**` condition in Python for handling multiple conditions sequentially.)

Q 4.

In Python, you can use the following two methods to write a new line:

1. **Using the `\n` Escape Sequence:** You can include the `\n` escape sequence within a string to create a new line. For example:

```
print("This is the first line.\nThis is the second line.")
```

Output:

```
This is the first line.
This is the second line.
```

2. **Using `print()` with No Arguments**: When you use the `print()` function with no arguments, it automatically prints a newline character, effectively moving to the next line. For example:

```
print("This is the first line.")
print("This is the second line.")
```

Output:

```
This is the first line.
This is the second line.
```

Both methods achieve the same result of writing a new line in Python.

Q 5.

In Python, you can read user input using the `input()` function and write output to the console using the `print()` function. Here's the syntax for reading user input and writing output:

```
# Read user input
user_input = input("Enter something: ")

# Process user input (if needed)
# ...

# Write output
print("You entered:", user_input)
```

In this syntax:

1. `input("Enter something: ")` prompts the user to enter something and waits for their input. The entered input is typically stored in a variable for further processing.
2. `print("You entered:", user_input)` is used to display the output to the console. You can replace `"You entered:"` with any message you want to display, and `user_input` contains the value entered by the user.

Q 6.

```
# Read the student's first and last name
full_name = input("Enter your first and last name: ")

# Split the full name into first name and last name
first_name, last_name = full_name.split()

# Print the welcome message
print(f"Welcome {first_name} {last_name}.")
```

In this program:

- We use the `**input()**` function to read the full name as input from the user.
- We then use the `**split()**` method to split the full name into two parts: the first name and the last name. By default, `**split()**` splits the string on spaces.
- Finally, we use an f-string to print the welcome message, combining the first name and last name entered by the user.

Q 7.

```
# Read the number of terms from the user
n = int(input("Enter the number of terms: "))

# Initialize variables
natural_numbers = []
total = 0

# Generate and display the natural numbers
for i in range(1, n + 1):
    natural_numbers.append(i)
    total += i

# Display the result
print(f"The natural numbers are: up to the {n}th term.", end=" ")
print(" ".join(map(str, natural_numbers)))
print(f"The total number of natural numbers is {total}.")
```

In this program:

- We read the number of terms, "n," from the user.
- We initialize two variables: `**natural_numbers**` as an empty list to store the natural numbers, and `**total**` to keep track of their sum.
- We use a`**for**` loop to generate the first "n" natural numbers, adding each number to the `**natural_numbers**` list and updating the `**total**` sum.
- Finally, we display the natural numbers and their sum using the `**print()**` function. The `**join()**` function is used to convert the list of natural numbers to a string for display.

Q 8.

```
# Recursive function to calculate factorial
def factorial(n):
    if n == 0:
        return 1  # 0! is defined as 1
    else:
        return n * factorial(n - 1)

# Input: Read a positive integer from the user
try:
    num = int(input("Enter a positive integer: "))
    if num < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        result = factorial(num)
        print(f"{num}! = {result}")
except ValueError:
    print("Invalid input. Please enter a positive integer.")
```

In this program:

- We define a recursive function `**factorial(n)**` that calculates the factorial of a positive integer `**n**`.
- If `**n**` is 0, we return 1 because, by convention, 0! is defined as 1.
- Otherwise, we recursively calculate `**n * factorial(n - 1)**`.
- We read a positive integer from the user using the `**input()**` function and handle cases where the input is not a positive integer.
- Finally, we calculate and print the factorial of the input number.

Q 9.

You can calculate the product of two non-negative numbers without using the `*` operator by repeatedly adding one of the numbers to itself for the other number of times.

Here's a Python program to do that:

```python
# Function to calculate the product of two numbers without *
def product_without_multiply(x, y):
    # Initialize the result
    result = 0

    # Iterate through the second number
    for _ in range(y):
        result += x

    return result

# Input: Read two non-negative numbers from the user
try:
    num1 = int(input("Enter the first non-negative number: "))
    num2 = int(input("Enter the second non-negative number: "))

    if num1 < 0 or num2 < 0:
        print("Please enter non-negative numbers.")
    else:
        # Calculate and display the product
        result = product_without_multiply(num1, num2)
        print(f"The product of {num1} and {num2} is {result}.")
except ValueError:
    print("Invalid input. Please enter non-negative numbers.")
```

In this program:

- We define a function `**product_without_multiply(x, y)**` that calculates the product of two numbers `**x**` and `**y**` without using the `*` operator.
- We initialize a variable result to 0.
- We use a `**for**` loop to iterate through the second number (`**y**`) and add the first number (`**x**`) to the `**result**` for each iteration.
- We read two non-negative numbers from the user and check if they are non-negative.
- If both numbers are non-negative, we calculate and display the product using the `**product_without_multiply**` function.

Q 10.

Output:

```
0
1
1
```

Here's the breakdown of the output:

- `n1` is assigned the value of `i` before it is incremented, so it gets the initial value of `0`.
- `n2` is assigned the value of `i` after it is incremented using the `++i` operation, so it gets the incremented value of `1`.
- Finally, `i` is incremented to `1` after both assignments.

Therefore, the values of `n1`, `n2`, and `i` are `0`, `1`, and `1`, respectively, when printed.

3rd Learning Outcome

Ensuring the usage of source control, for example (GIT).

Evidence and Proof Requirements

After completing this section, students should be able to:

- Create storage space for the project (Source Code Repository).
- Copy the project from the cloud (Clone).
- Document each stage of the project (Commit).
- Pull other updates from the cloud (Pull).
- Upload the latest update on the cloud (Push).

Source Control Management System

1 The Source Control

There were traditional ways in the past to exchange the source code of programs among software developers. These traditional ways like flash memory USB, hard disk, CD, or sharing on the cloud. The drawbacks of such these ways were that any change in the source code doesn't reflect in the other versions. So, any change needs to other exchange for the source code. However, during the great advances in technology, a new concept appeared called a source control tool.

The Source Control Tools

The target is to make a master point called a (Repository) for the source code and this repository is shared between the members of the developers. So, if any developer changes the source code either through addition, modification, or deletion for some blocks of code, he/she can merge these changes easily in the master repository. Therefore, the other members can pull these changes from the master point of code. Thus, each member is updated with any changes in the code.

After we learned what is the meaning of source control tool. In the following, we will mention some examples of the most commonly used open-source version control tools among developers. The following are different examples of these version control tools:

1. **Git** is the most common tool used for source code management.
2. **TFS** is a Team Foundation System developed by Microsoft.
3. **Mercurial** is a distributed source code management tool.
4. **CVS** is a Concurrent Versions System.
5. **Subversion.**
6. **Bazaar**.

The following figure includes the icon logo of each source code management tool:

Figure (101): Top version control systems

2 Source Control File States

Any exchanging files operation on the source control like Git has three states of these files before moving these changes into the remote repository.

1. Working directory.
2. Staging area.
3. Repository.

Working Directory: It is the place of the local version of the source code which may be our personal computer (PC) or laptop. Any changes in this state don't reflect on the main repository source code.

Staging Area: It is an intermediate state between the working directory (the local copy) and the repository (the main remote shared copy). It's an important stage because you could preview any changes before confirming these changes for the remote repository. Each group of changes could be added as a commit operation. Don't worry, we will discuss the commit operation in more detail next. But for now, when we make any changes in the source code, we could move these changes into the next step (staging area) and if the changes are correct and there are no conflicts. Then we could move these changes into the remote repository.

Repository: It is the local place of the source code that has the committed changes.

Note (22)

The changes in this stage still don't affect the remote repository until using the push command. The push command will be discussed later.

These stages are shown in figure (102):

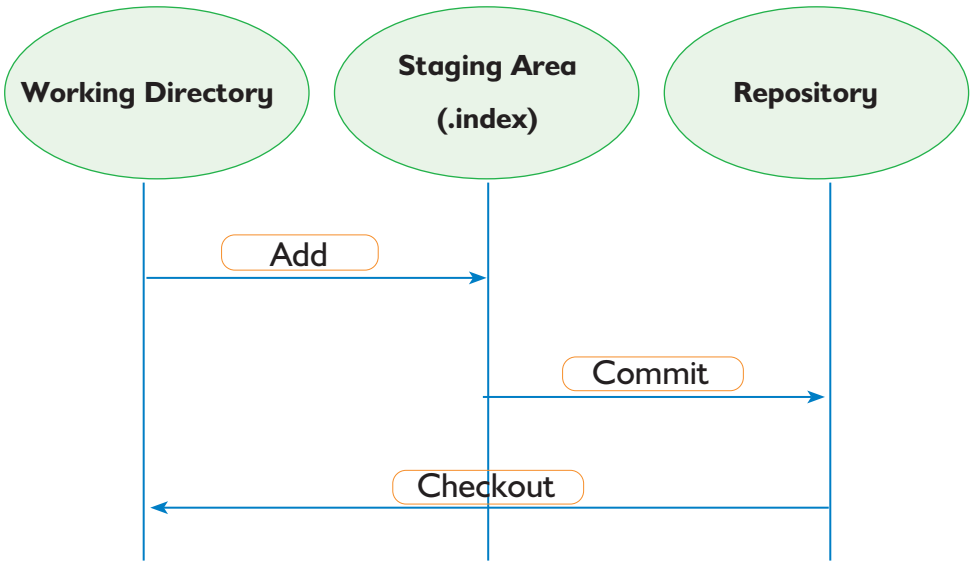


Figure (102): Sourcecontrolstates

Because the Git tool is the most commonly used source code management tool, we will discuss it in detail in the following from this lesson.

What is the Git software tool?

- GIT is a free and open-source distributed version control system (DVCS). Version control is a way to save changes over time without overwriting previous versions.
- Being distributed means that every developer working with a Git repository has a copy of that entire repository: every commit, every branch, and every file.
- DVCS uses a central server to store all files and enables team collaboration.
- GIT repositories can be connected, so you can work on one locally on your own machine, and connect it to a shared repository. This way, you can push and pull changes to a repository and easily collaborate with others.
- However, Git does not rely on the central server and that is why you can perform many operations when you are offline. You require a network connection only to publish your changes and take the latest changes.

Note (23)

There is a cloud-based service called GitHub. So, you should recognize the difference between Git and GitHub.

Git vs. GitHub

- Git is the software tool that controls and tracks your changes and facilitates the way of code exchanging.

But

- GitHub is just an online cloud-based hosting service for repositories (source codes).

So, if you have your storage host, you could use it instead of GitHub and use Git to track the exchanging process with developer members.

The following figure contains the benefits of using the Git tool:

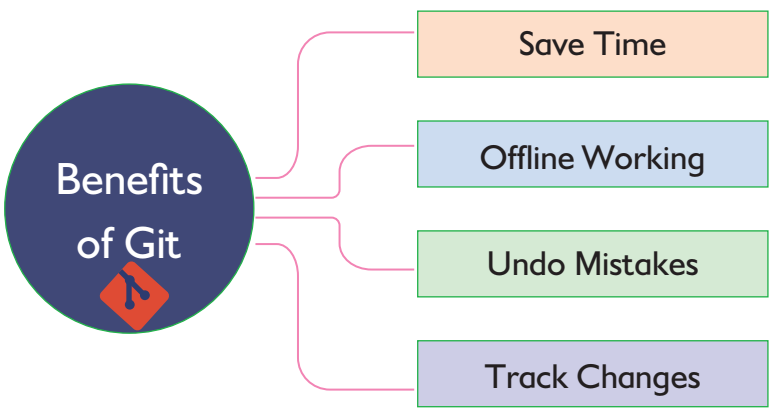


Figure (103): BenefitsofGit

Before creating the first repository:

After this delicious introduction to the source control and the Git software tool.

Besides, in Competence 1 we learned how to install GitHub which installed Git by default.

We have two important steps:

Firstly, make sure that Git is installed successfully about:

From the Windows icon search for either the command prompt Command Prompt Git (CMD) or Git Bash tool. Here we use Git Bash as shown in figure (104).

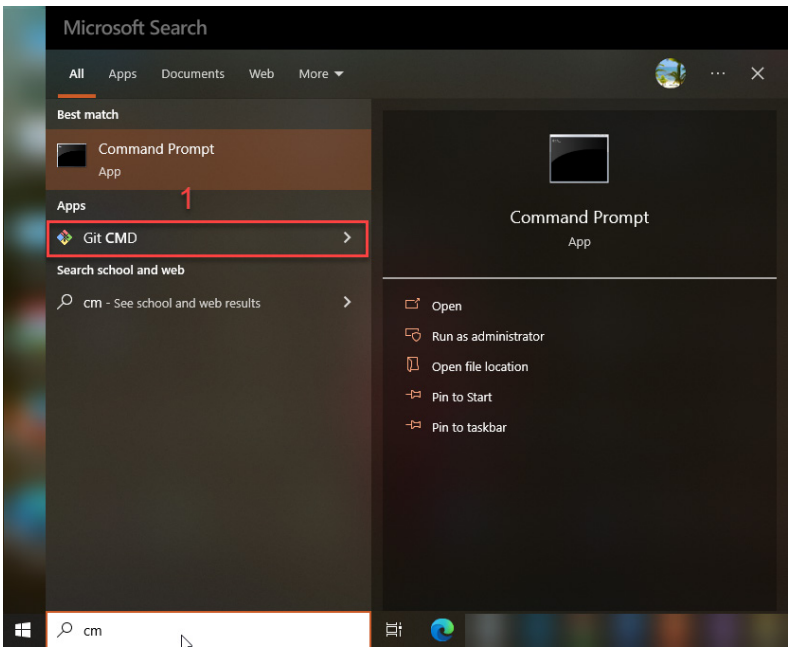


Figure (104): Verify Git installation

Open the Git CMD as shown in figure (105).

Figure (105): Open Git CMD

- Type the following command:

  As shown in figure (106): git - version

Figure (106): Get the Git version

- If Git is already installed, it should show the Git version something like in figure (106).
- Else review the installation steps in Competence 1

Secondly, every developer should have an account on any remote hosting source control website like GitHub to could be known to the other developers. So in the next steps, we will learn how to create an account on GitHub:

Open the GitHub website through https://github.com and click on Sign up as shown in figure (107).

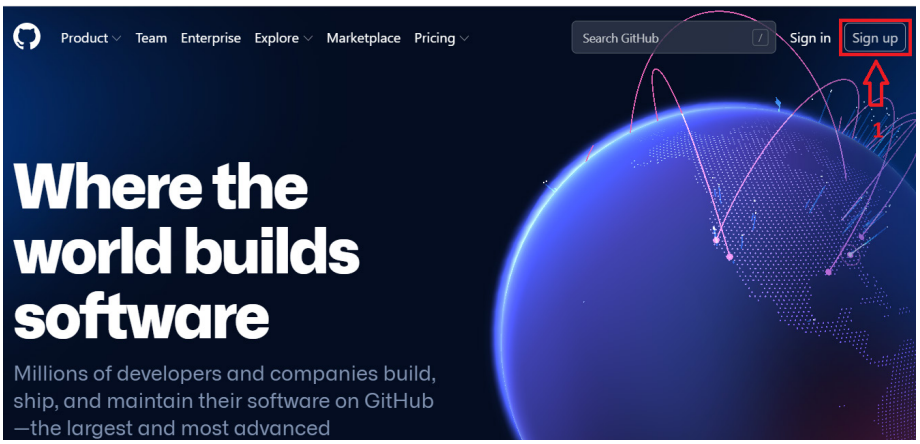


Figure (107): Create GitHub account step 1

- Enter a valid email as shown in figure (108).

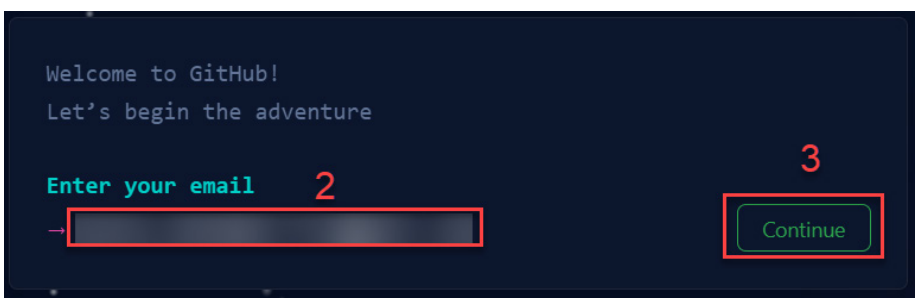


Figure (108): Create GitHub account step 2

- Enter your password as shown in figure (109).

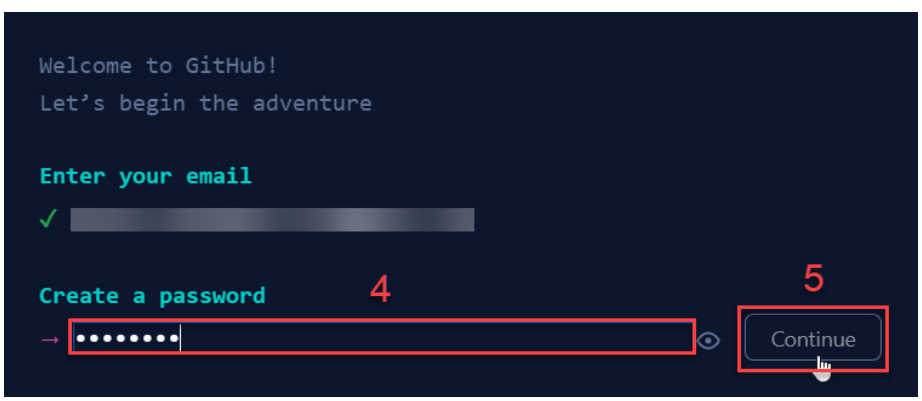


Figure (109): Create GitHub account step 3

- Enter your user name as shown in figure (110).

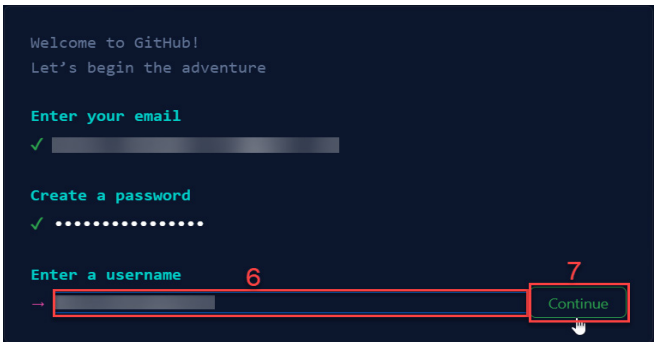


Figure (110): Create GitHub account step 4

- Continue with an option for receiving any updates about GitHub, followed by a simple puzzle for ensuring that you are a human.

- After that click on Create account as shown in figure (111).

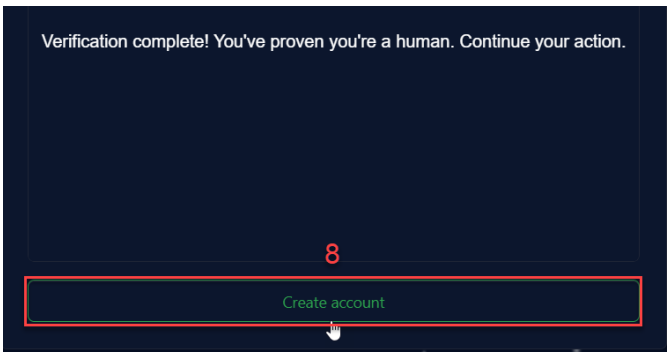


Figure (111): Create GitHub account step 5

- A verification code is sent for the email you signed up with it. Copy this verification code as shown in figure (112).

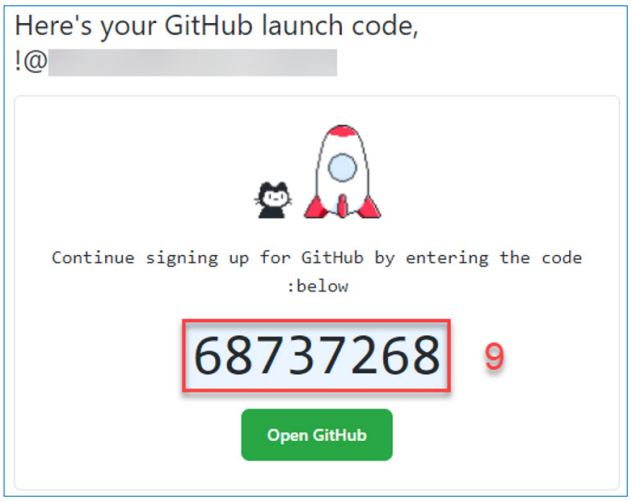


Figure (112): Create GitHub account step 6

- Paste this verification code on the following page as shown in figure (113).

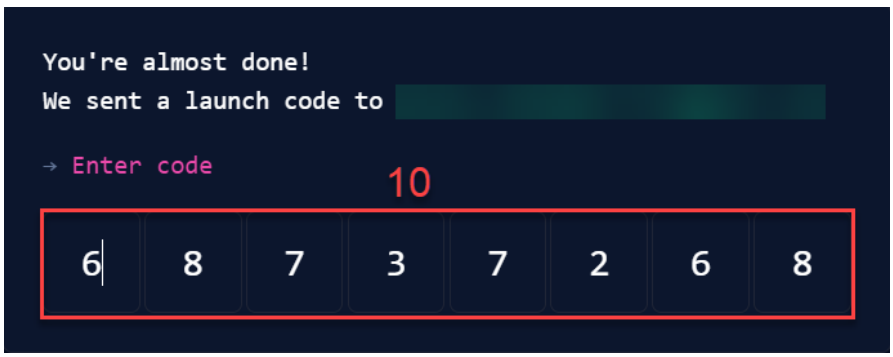


Figure (113): Create GitHub account step 7

- Finally, now you have created an account on GitHub successfully and the following image is the GitHub home page as shown in figure (114).

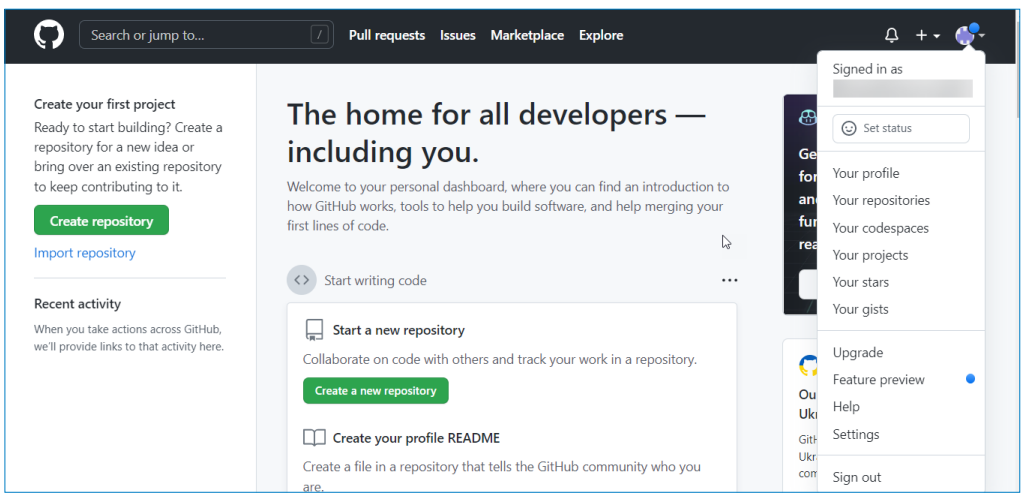


Figure (114): Create GitHub account step 8

The most commonly used Git commands:

Once Git is installed successfully, and you have created your account successfully. We could start by creating our first software project (repository). So, the next will be steps to create a new repository and apply the most commonly used commands like Clone, Add, Commit, Pull, and Push.

Now, we need to learn how to use Git with GitHub. There are two options to work on a software project:

1. Working on the existing repository of the project.
2. Create our repository.

Assume that we need to build a software project that targets to utilize every empty space in the parking of our company. So, we will set the environment of this software project to be shared among all developers who will contribute to this project. From this idea, we could follow the following steps to create our repository and each one could contribute his/her code.

Step 1: Create our repository on GitHub

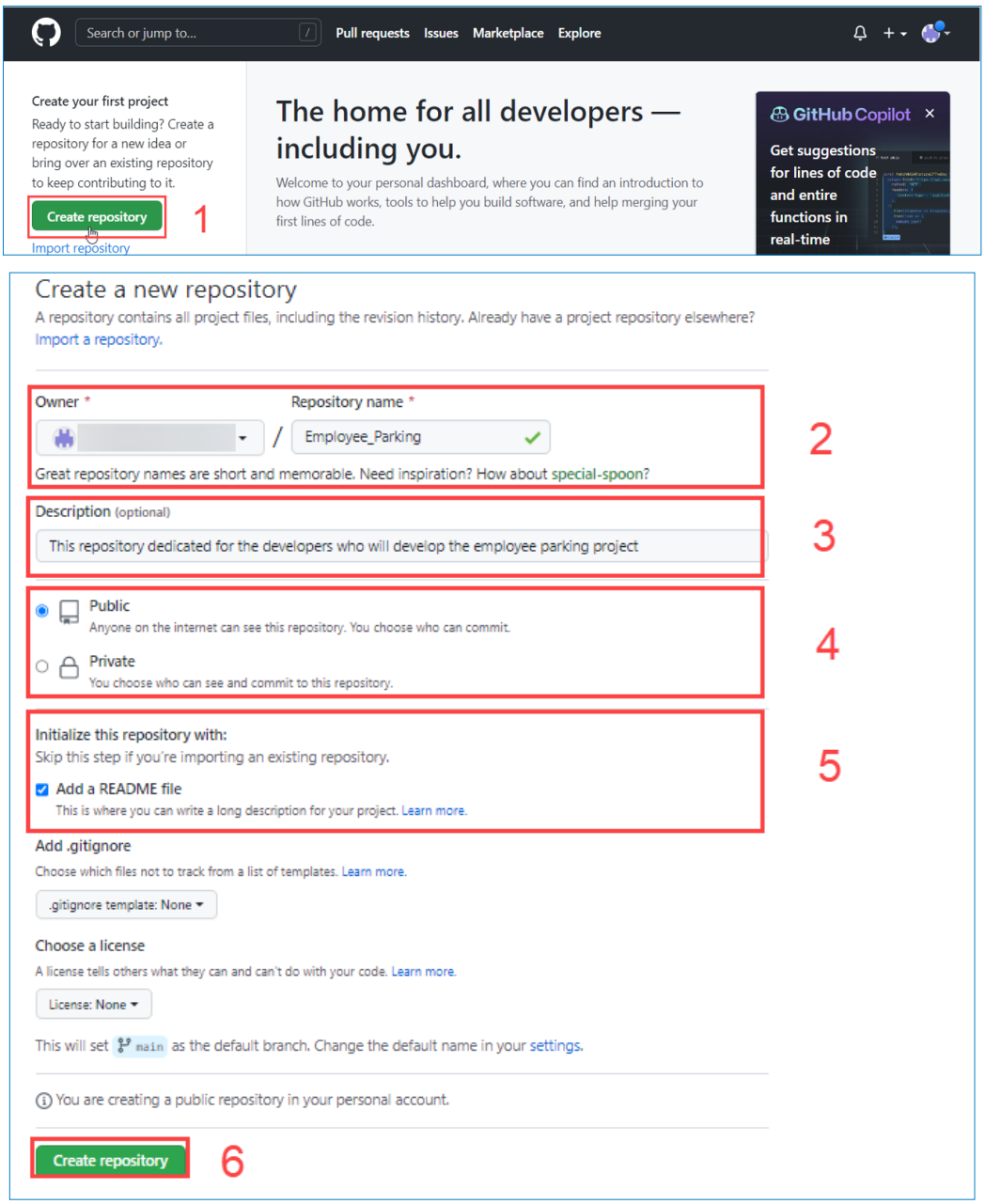


Figure (115): Create GitHub repository steps (1 - 6)

- In the image, number 2 contains the repository name called Employee_Parking.
- Number 3 contains the description of this repository. And this field is optional.
- Number 4 refers to the privacy of this repository either public or private. In our case, we would make it public for simplicity.
- Number 5 has the option of adding the readme file to the repository. In this file, you could write the instruction guide to use this software project.
- Finally, the repository of the Employee_Parking software project has been created successfully as in the following image as shown in figure (116).

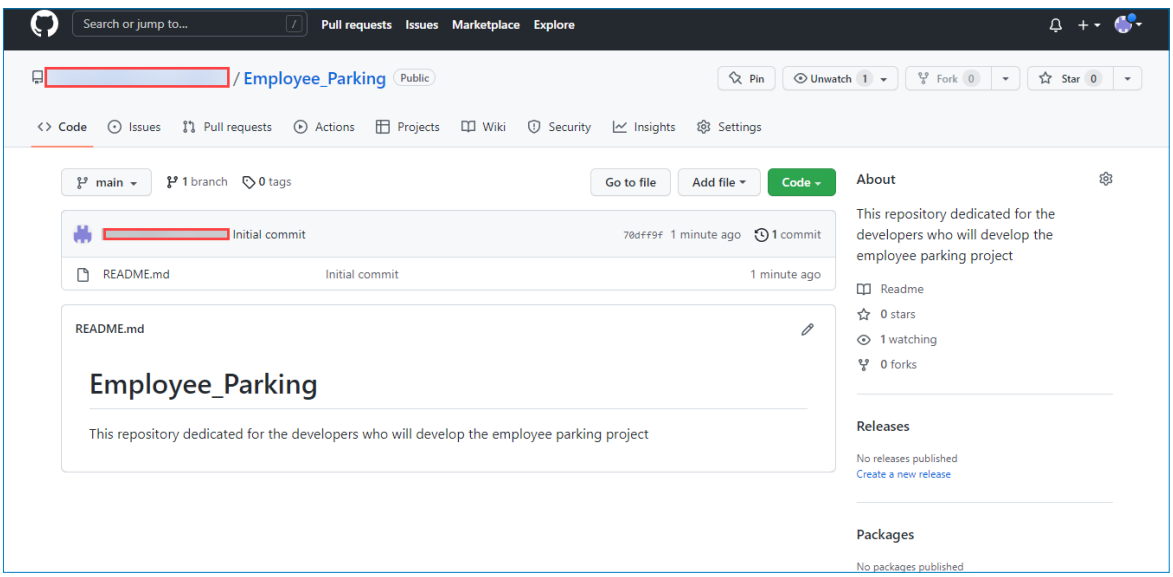


Figure (116): Github created a project repository

Note (24)

The red rectangle in the image has the username of the repository owner. But here is hidden for privacy.

Also, before moving into step 2, there is an important note:

If you make the repository public, then any developer could get a copy from this repo. If you make the repository private, then either get a token plus the repository link or the repository owner adds each team member as a collaborator.

Step 2: Each one of the team should have a copy from this repository

After one of us created the repository to be the main place for our parking system. Every other developer in the team should have a copy from this repository to be able to work on this repository locally.

This step could be performed easily in Git using the clone command.

Note (25)

The clone command: download an existing software source code on the remote hosting website like GitHub into the local personal computer.

The clone syntax:

Git clone remote-repository-link

The Pull Command:

Firstly, we could get the link to the remote repository according to the steps in the following image as shown in figure (117).

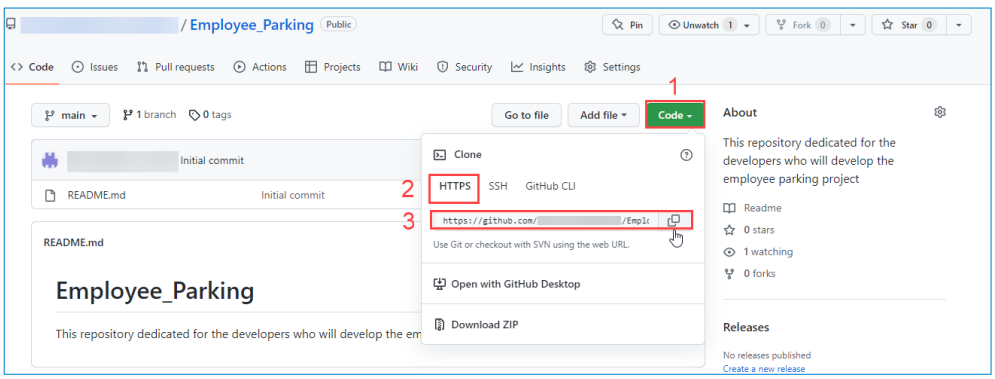


Figure (117): Get the remote repository link

- Then, open the Git Bash command line and write the clone command as shown in figure (118).

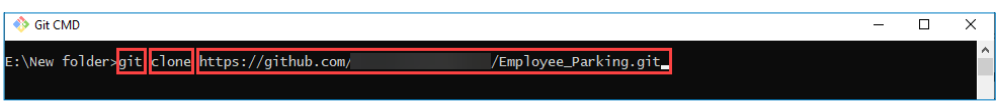


Figure (118): Clone remote repository command (clone)

In the following image, you notice that the Employee_Parking project is downloaded locally in partition (E) inside a folder named (New Folder) as shown in figure (119).

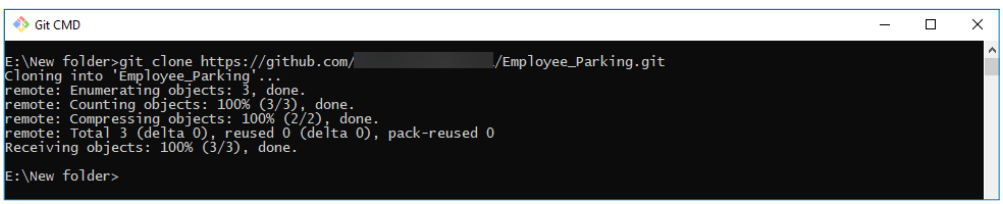


Figure (119): Cloned remote repository

- The following image contains all files in this project after loading locally as shown in figure (120).

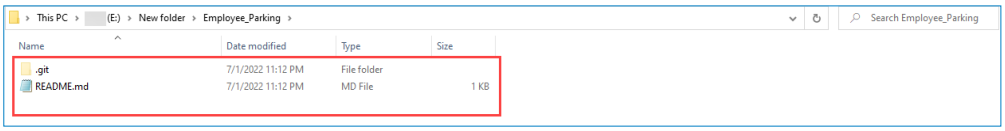


Figure (120): Cloned remote repository in file explorer

Step 3: Add files to the project locally

We will create a description document file in the same location as the project and name it project_requirements.docx as shown in figure (121).

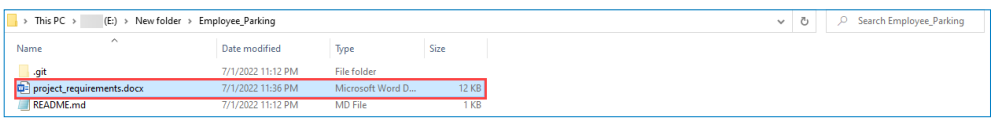


Figure (121): Add description file to project

Firstly, we should move the cursor of Git Bash into the project folder by using cd /d "local_project_path" as shown in figure (122).

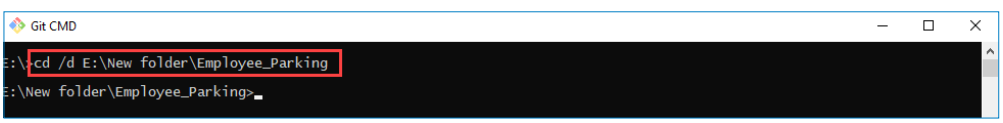


Figure (122): Move Git Bash to the project path

- Now, we could know any changes in the project folder locally by using the Git status command as shown in figure (123).

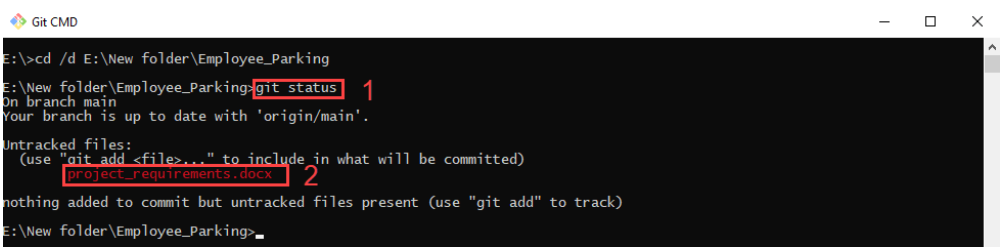


Figure (123): Get project status by "Git status" command

Step 4: Adding these changes to the staging area

- Use the following commands:

  - Git add <file name> or Git add <directory name> or Git add.

  The add command moves the changes from the working directory into the staging area.

Note (26)

Use the command Git then space then add the dot to apply all changes in all files in the project.

- Git commit -m “readable message” as shown in figure (124).

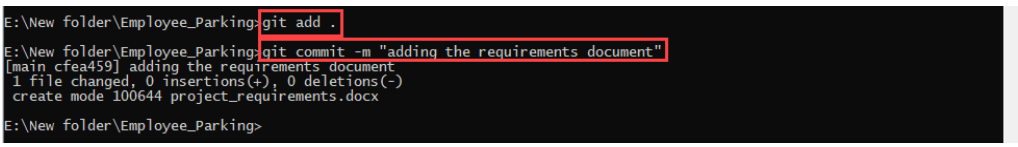


Figure (124): Confirm changes by "Git commit" command

Note (27)

After finishing step 4, the changes still don’t affect the remote repository. Figure (124) is after step 4.

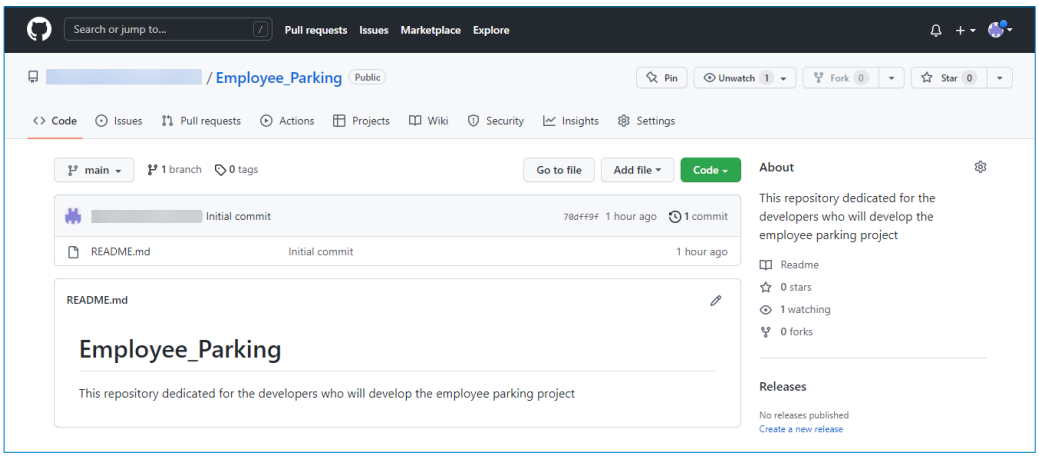


Figure (125): Remote Git repository after confirming changes

Step 5: Push these changes to the remote repository

The push command is used to apply these changes remotely by using:

Git push origin as in the following image as shown in figure (126).

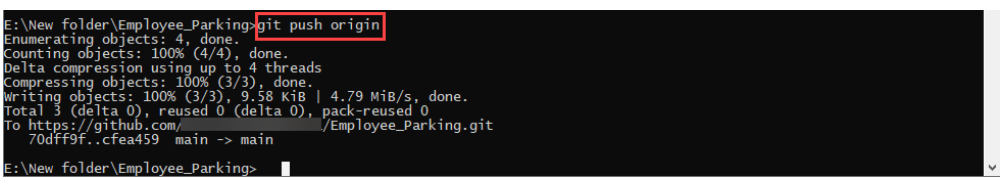


Figure (126): Push command

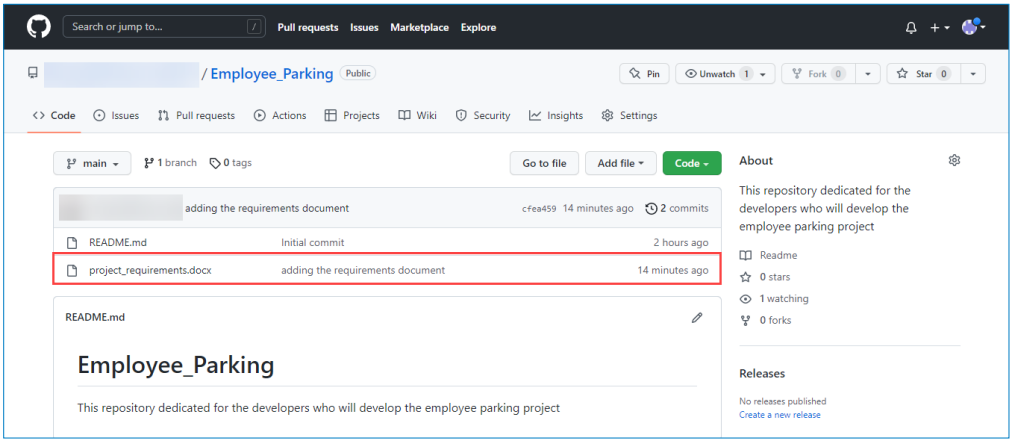


Figure (127): Remote repository after push command

In figure (127), we could see that the requirements file has been added successfully to the remote repository. So, any team member could see this change and get the latest from this project.

Note (28)

If any changes happen on the remote repository, how can I get these updates?

- The pull command is used to get any updates.
- The pull command syntax: Git pull

The Push Command:

Assuming that the repository owner makes changes like uploading an image file to the repository as shown in figures (128-131).

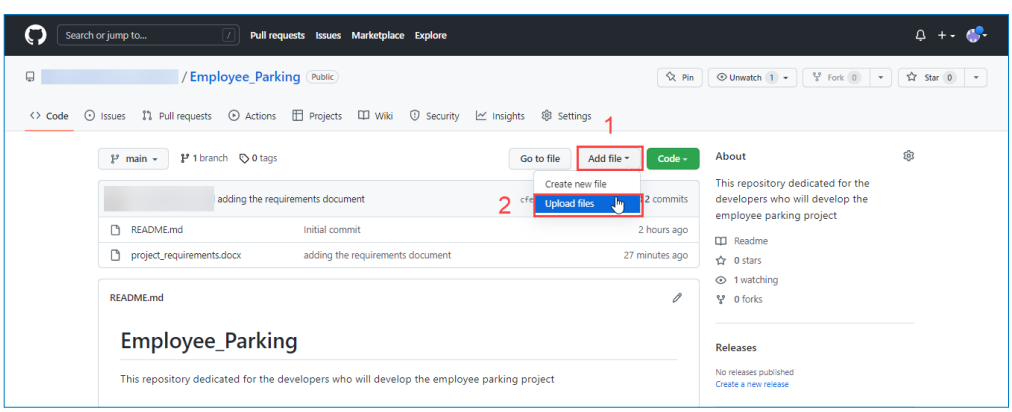


Figure (128): Web-based repository upload step 1

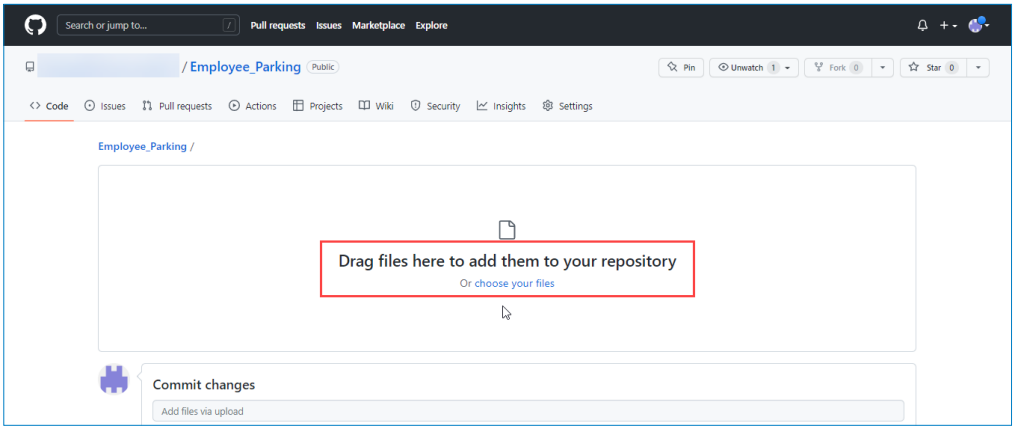


Figure (129): Web-based repository upload step 2

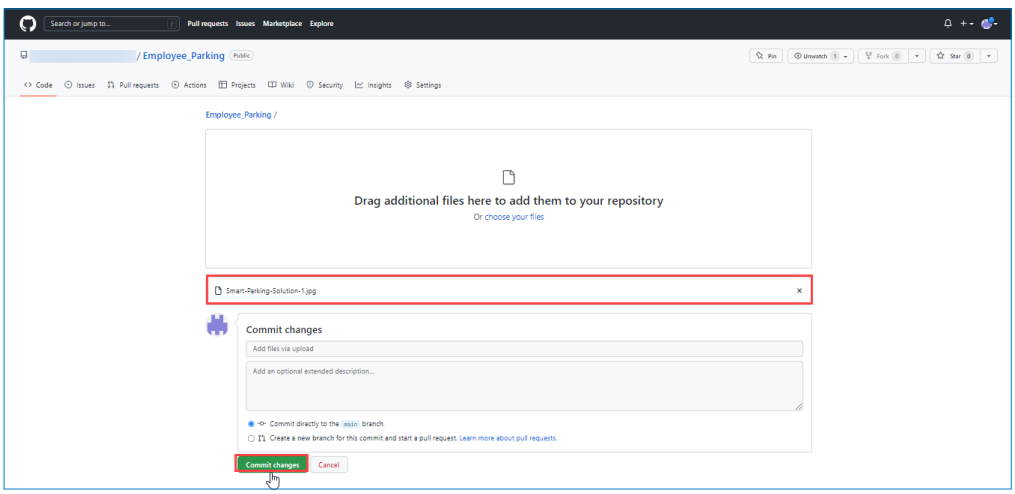


Figure (130): Web-based repository upload step 3

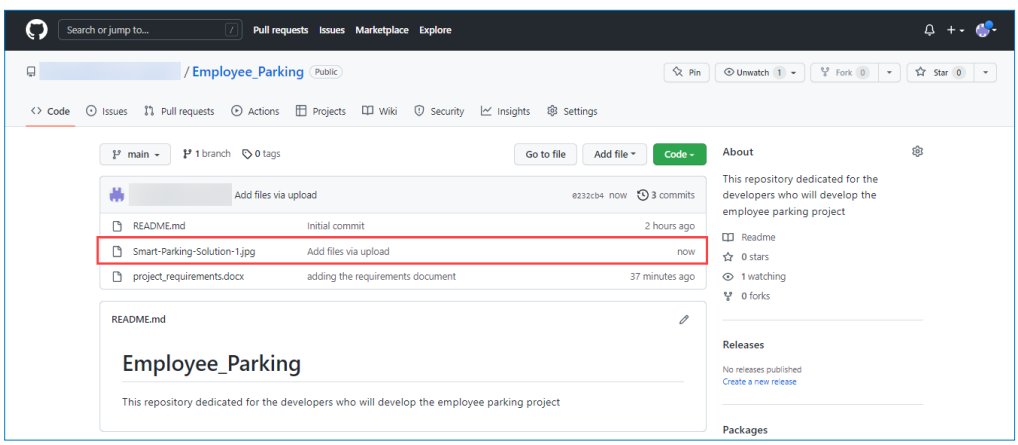


Figure (131): Web-based repository upload step 4

So, you like to get this change on our PC (locally). Simply, use the pull command. In the following image, there is one file changed in insertion type as shown in figure (132).

```
E:\New folder\Employee_Parking>git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 104.01 KiB | 608.00 KiB/s, done.
From https://github.com/                /Employee_Parking
   cfea459..0232cb4  main       -> origin/main
Updating cfea459..0232cb4
Fast-forward
 Smart-Parking-Solution-1.jpg | Bin 0 -> 106047 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Smart-Parking-Solution-1.jpg

E:\New folder\Employee_Parking>
```

Figure (132): Pull command

Now, when going into the working directory, we could see this image file downloaded locally as shown in figure (133).

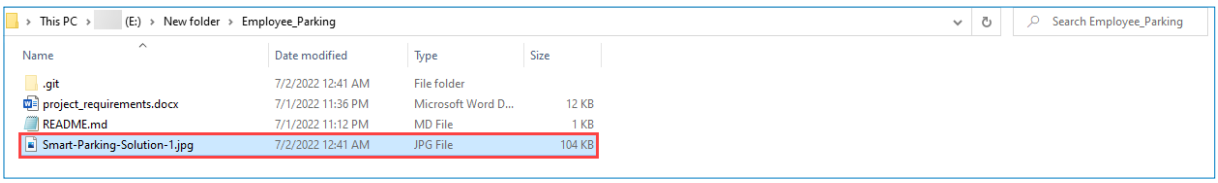


Figure (133): Pull command effect in file explorer

At the end of this lesson, we are happy to tell you that you have the most beneficial basics to start by using the source control tools in your work. Next, there is an assessment to test your understanding.

ASSESSMENT

Q 1. What is the source control?

Q 2. What is the importance of source control tools?

Q 3. Compare the following Git commands in terms of usage and syntax:

- Clone
- Add
- Commit
- Push
- Pull
- Status

Q 4. Mention the usage of the readme file.

Q 5. Thinking about any project idea, apply the following to it:

a) Create your account on GitHub.

b) Create a repository of the project idea.

c) Clone this project locally.

d) Make changes to the local version.

e) Move these changes into the staging area.

f) Apply these changes remotely.

Q 6. Do you need to use the pull command in Question 5? Why?

4th Learning Outcome

Verifying that the developed software meets its intended purpose.

Evidence and Proof Requirements

After completing this section, students should be able to:

- Test the program to make sure it is free of software errors.
- Test the program to ensure that its purpose is achieved.
- Adjust the program as needed.

Verify That the Developed Software Meets Its Intended Purpose

Introduction

In software development, it's crucial to ensure that your code functions correctly and meets its intended purpose. This section covers various techniques and tools to help you verify the quality and correctness of your Python programs.

1 Verifying That the Program Is Error-Free

To develop professional software, the programmer should make sure that the program is error-free. The error affects the program's productivity and may make the program not function correctly in some cases.

To make sure that the program is error-free, the developer should know the types of errors. There are three main types of programming errors, as shown in figure (134).

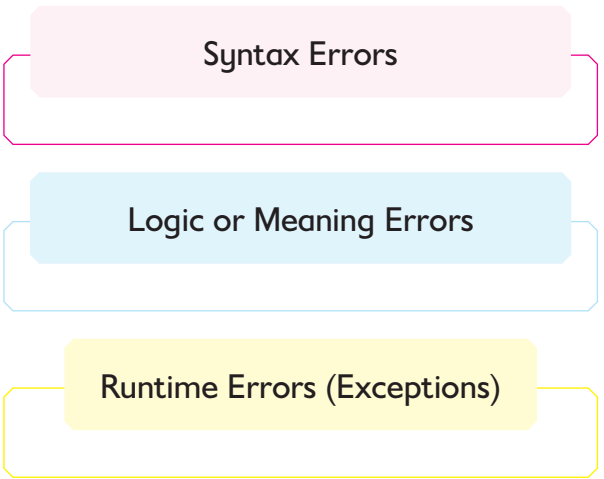


Figure (134): Types of errors in programming

Syntax Errors:

These errors occur when your code violates the rules of the Python language. They are typically caught by the Python interpreter during the compilation phase. Examples include misspelled variable names, missing colons at the end of statements, and incorrect indentation.

Example

Find the syntax error in the following code:

```
print("Hello, world!"
```

Figure (56x): Example of a syntax error

In this example, the syntax error occurs because the closing parenthesis `)` is missing. Python expects a closing parenthesis to match the opening parenthesis in the `print` function, and without it, the code is incorrect.

Answer:

```
print("Hello, world!")
```

To fix this syntax error, you should add the missing closing parenthesis: , Now, the code is correct, and it will print "Hello, world!" as expected.

Logic or Meaning Errors:

Logic errors occur when your code runs without producing any error messages but doesn't produce the desired or expected output. These errors are often the result of flawed algorithms or incorrect use of programming constructs. Identifying and debugging logic errors may require careful code inspection and testing.

Example

```
# Calculate the average of three numbers
num1 = 10
num2 = 20
num3 = 30

average = (num1 + num2 + num3) / 2  # Logic error: Divide by 2 instead of 3

print("The average is:", average)
```

Answer:

The output of the code will be: The average is 30.0

This is because the code calculates the average of num1, num2, and num3 by summing them and dividing by 2. However, there's a logic error in the code because it should be dividing by 3 to correctly calculate the average of three numbers.

Runtime Errors (Exceptions):

Runtime errors, also known as exceptions, occur during the execution of your program. They can occur for various reasons, such as dividing by zero, attempting to access an index that is out of bounds, or trying to use a variable that has not been defined. Proper error handling techniques, such as try-except blocks, can help mitigate the impact of runtime errors.

Example

Find the error in the following code:

```
x = 9
y = 0
result = x / y
print("result:", result)
```

Answer:

The code you've provided attempts to divide `x` by `y`, which will result in a `ZeroDivisionError` because you cannot divide by zero in Python or most programming languages. When you run this code, it will raise an error like this:

Debugger Software Tool:

A debugger software tool is a built-in tool inside the most integrated development editor toolkits, like Visual Studio. The debugging feature facilitates the tracing of the code about:

Verifying that the logic of the code is correct.

Watching the values of variables during the runtime.

The debugging is performed by putting a breakpoint on the line of code to debug. To perform debugging on a piece of code, you need to perform the following steps:

a Create a Project

Creating a Python project in Visual Studio Code (VS Code) involves several steps. Here's a general outline of the process:

Step 1: Create a new Python project

A project is how Visual Studio manages all the files that come together to produce a single application. Application files include source code, resources, and configurations.

A project formalizes and maintains the relationships among all the project's files. The project also manages external resources that are shared between multiple projects. A project allows your application to effortlessly expand and grow. Using projects is much easier than manually managing relationships in unplanned folders, scripts, text files, and your memory.

We will start with a simple project containing a single, empty code file.

In Visual Studio, select File > New > Project or press Ctrl+Shift+N. The Create a new project screen displays where you can search for and browse templates across different languages.

To view Python templates, search for Python. Search is a great way to find a template when you can't remember its location in the language tree.

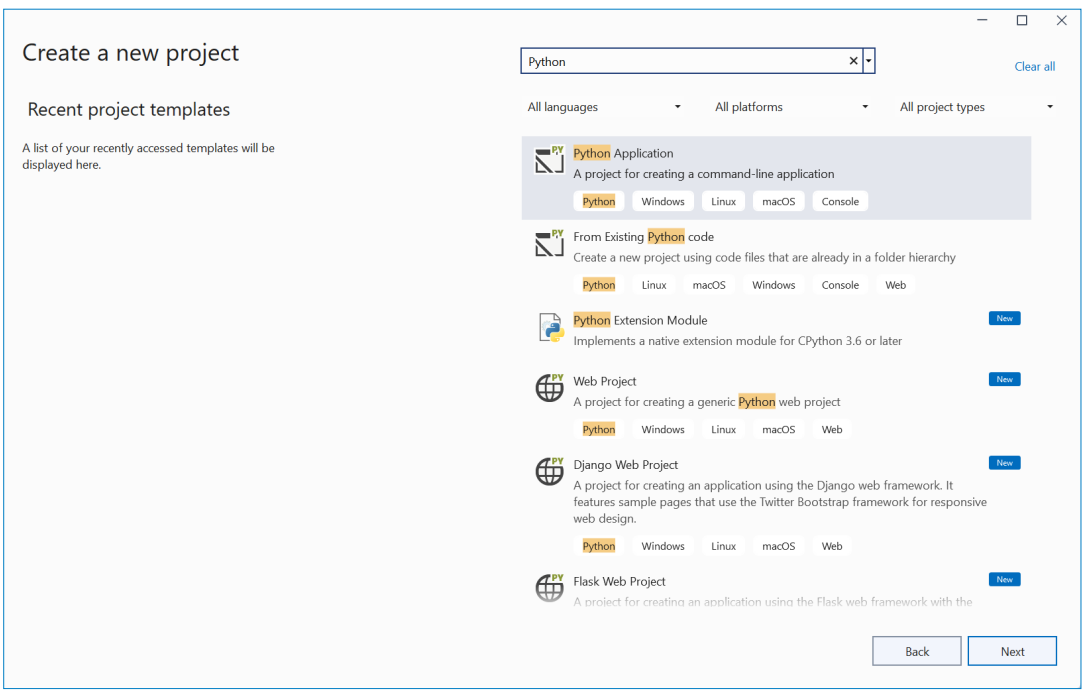


Figure (135): Create a new project step 1

Python web support in Visual Studio includes several project templates, such as web applications in the Bottle, Flask, and Django frameworks. When installing Python with the Visual Studio Installer, check "Python Web Support" under optional to install these templates. For this tutorial, start with an empty project.

Select the Python application template, and select Next.

On the Configure your new project screen, specify a name and file location for the project, and then select Create.

The new project opens in Visual Studio.

- The Visual Studio Solution Explorer window shows the project structure (1).
- The default code file opens in the editor (2).
- The Properties window shows more information for the item selected in Solution Explorer, including its exact location on disk (3).

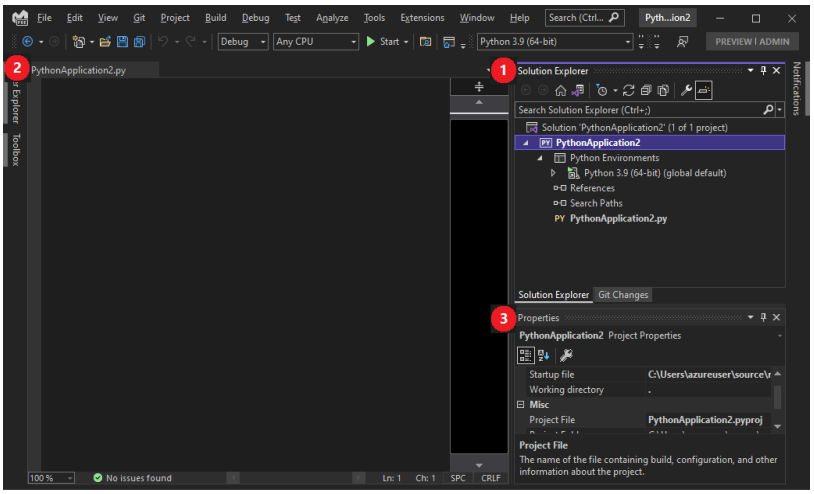
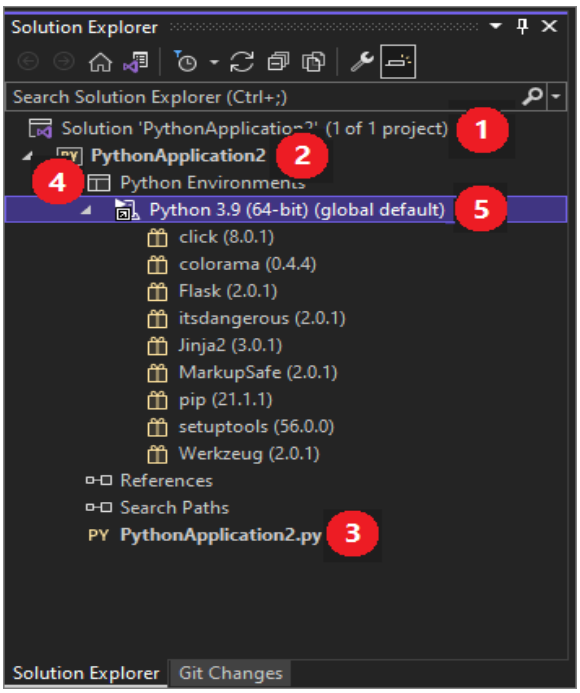


Figure (136): Create a new project step 2

Familiarize yourself with Solution Explorer, where you can browse files and folders in your project.



- At the top level is the solution, which by default has the same name as your project (1).
- A solution, which is shown as a .sln file on disk, is a container for one or more related projects. For example, if you write a C++ extension for your Python application, that C++ project can be in the same solution. The solution could also contain a project for a web service, and projects for dedicated test programs.
- Your project, with the name you gave in the Create a new project dialog box, displays in bold (2). On disk, the project is a .pyproj file in your project folder.
- Under your project are source files, in this case only a single .py file (3). Selecting a file displays its properties in the Properties window (if you do not see the Properties window, click the wrench in the Solution Explorer banner). Double-clicking a file opens it in whatever way is appropriate for that file.
- Also under the project is the Python Environments node (4). Expand the node to show the available Python interpreters.
- Expand an interpreter node to see the libraries installed in that environment (5).

Right-click any node or item in Solution Explorer to show a context menu of applicable commands. For example, Rename lets you change the name of a node or item, including the project and the solution.

b Write Code

To experience Python, create a file (using the File Explorer) named hello.py and paste in the following code, this will print " Hello World":

Figure (137): Sample code

c Start the Debugger

To start the debugger in Visual Studio Code (VS Code) for a Python project, you'll need to follow these steps:

1. **Open Your Python File:** Make sure you have a Python file (e.g., `main.py`) open in the VS Code editor. This should be the file you want to debug.
2. **Set Breakpoints (Optional):** If you want to pause the execution of your code at specific points for inspection, set breakpoints by clicking in the left margin of the code editor next to the line numbers where you want to pause.
3. **Launch the Debugger:**

• Using the Run and Debug Activity Bar:

- Click on the Run and Debug activity bar on the side panel (it looks like a "Play" button).

- You'll see a green triangle button labeled "Run and Debug" at the top. Click it to launch the debugger.

• Using Keyboard Shortcut:

- The default keyboard shortcut to start debugging is `F5`. Press `F5` while your Python file is open.

4. **Select a Debugger Configuration**: VS Code will prompt you to select a debugging configuration. Choose "Python" if it's available. If you don't have a `launch.json` configuration file, VS Code may offer to create one for you. Confirm your selection.
5. **Debugging Actions:**

Your code will start running in debug mode.

If you set breakpoints, the execution will pause when it reaches those breakpoints, and you can inspect variables, evaluate expressions, and control the execution flow.

Use the debugging toolbar at the top of the window to control the debugger (e.g., Step Over, Step Into, Continue, Stop).

6. **Inspect Variables:** You can hover over variable names in your code to see their current values or add them to the watch panel for continuous monitoring.

7. **Debug Console:** The Debug Console at the bottom of the VS Code window allows you to execute Python expressions and see their results while debugging.
8. **Debugging Toolbar:** TIt provides buttons for common debugging actions, such as stepping through code, continuing execution, and stopping debugging.
9. **Exit Debug Mode**: To exit debug mode, you can click the red square button in the top toolbar, which will terminate the debugger and return you to normal editing mode.

These steps should help you start and use the debugger in VS Code for Python debugging. Remember to set breakpoints at specific points in your code where you want to inspect variables or pause execution.

d Set a Breakpoint and Start the Debugger

A breakpoint is an intended stopping or pausing point in software for debugging reasons. Breakpoints are one of the most basic and essential features of reliable debugging. A breakpoint indicates where visual studio should suspend your running code so you can take a look at the values of variables, the behavior of memory, or whether or not a branch of code is getting run.

1. To set a breakpoint, click the left margin of the line of code or click on F9. For example, in n the for statement, set a breakpoint:

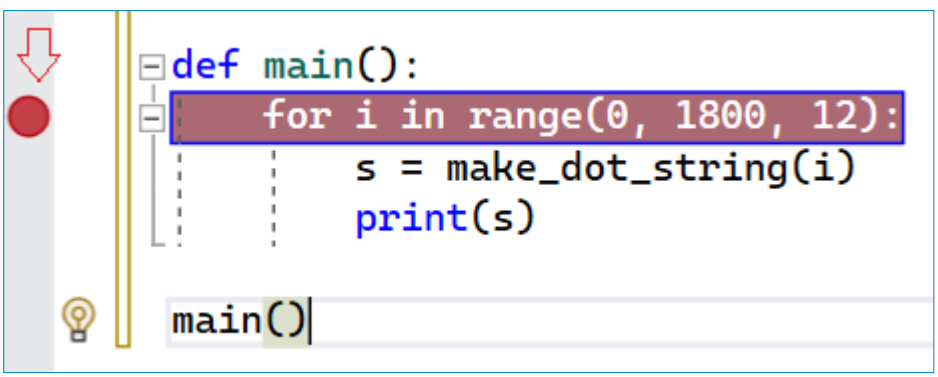


Figure (138): Set breakpoint

As shown in figure (138), a red circle appears where you set the breakpoint.

2. To start the debugger again (F5) and see that running the code stops on the line with that breakpoint. Here you can inspect the call stack and examine variables. Variables that are in-scope appear in the Autos window when they're defined; you

can also switch to the Locals view at the bottom of that window to show all variables that Visual Studio finds in the current scope (including functions), even before they're defined:

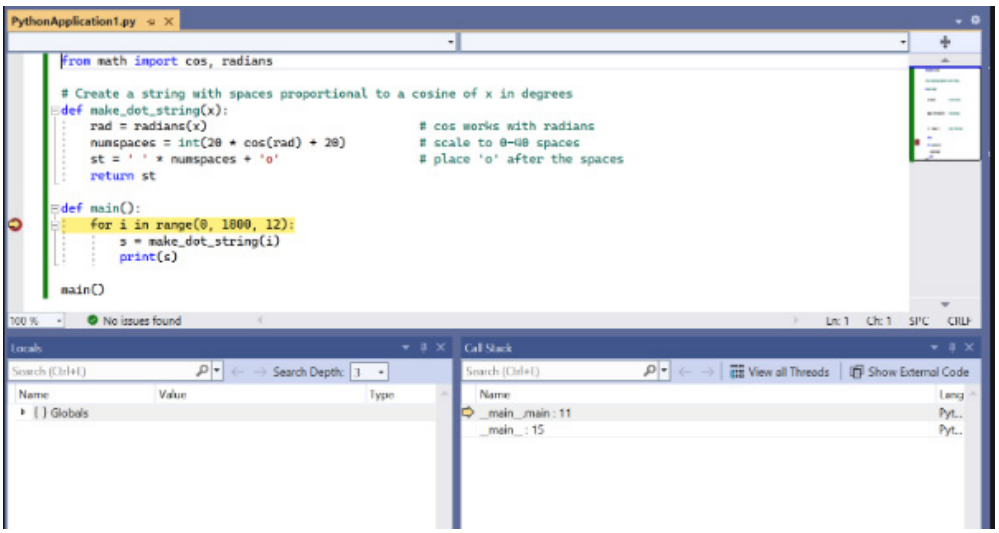


Figure (139): Start debugging with a breakpoint

e Navigate Code in the Debugger Using Step Commands

Sometimes you may need to set more than one breakpoint in your code. To navigate among the code that has several breakpoints, we usually use keyboard shortcuts.

1. By pressing F11 or choosing Debug > Step Into to get into the code of the current line.
2. By pressing F10 or choosing Debug > Step Over to jump over the current line of code. Step over aims to run the current line of code and pause on the successive line of code.

If you pressed (Shift+F11)) or Debug > Step Out inside a function, it would execute the rest of the current function until the current function returns.

3. By pressing F5 to jump to the next breakpoint.

f Inspect Variables with Data Tips

One of the main features of debugging is that it enables you to inspect. There are several ways to inspect variables during debugging, as follows:

1.

1. By hovering over the aimed variable, the current value appears, as shown in figure (140).

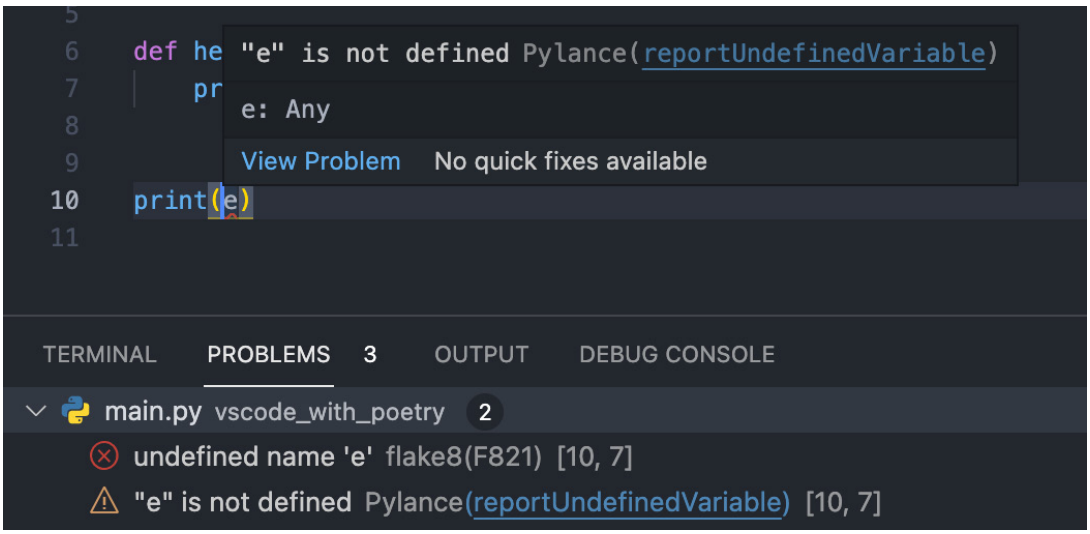


Figure (140): Inspect variables with a mouse hover

2. You can add a watch to monitor the value of a variable during debugging, the process generally involves:

- Right-clicking on the variable of interest while debugging.
- Selecting the "Add Watch" option from the context menu.
- This will create a watch expression, and you can see the current value of the variable and any changes it undergoes during debugging.

The ability to add watches is a powerful debugging feature that allows you to keep track of specific variables and their values as you step through your code. This helps you better understand how your code behaves and identify any issues or unexpected behavior.

2 Verifying That the Program Performs the Required Task

Once you have written and debugged your code, the next critical step is to ensure that the program performs the intended task correctly. This phase of software development involves running the program with various inputs and scenarios to validate its functionality.

Here are some essential steps and techniques for verifying that your program meets its intended purpose:

1. Test the Program with Different Inputs:

Testing your program with a variety of inputs is a fundamental step in verifying its functionality. Consider different use cases, edge cases, and scenarios that your program might encounter. Ensure that the program produces the correct output and handles various input conditions gracefully.

2. Test Boundary Cases:

Boundary testing involves evaluating the behavior of your program when inputs are at their minimum or maximum allowable values. This helps identify any issues related to input limits, such as buffer overflows or unexpected behavior when input values are at extremes.

```
def calculate_rectangle_area(width, height):
    if width <= 0 or height <= 0:
        return 0  # Invalid dimensions, return 0 area
    return width * height

# Test the program with boundary cases
width = 0  # Minimum allowable width
height = 0  # Minimum allowable height
area1 = calculate_rectangle_area(width, height)  # Expecting 0

width = 1000  # Maximum allowable width
height = 1000  # Maximum allowable height
area2 = calculate_rectangle_area(width, height)  # Expecting 1,000,000

print("Area 1:", area1)
print("Area 2:", area2)
```

Figure (141): Example of testing boundary cases

In this example, we test boundary cases for the `calculate_rectangle_area` function. We check two scenarios:

1. Minimum Allowable Dimensions: We set both width and height to their minimum allowable values, which is 0. The expected result is 0, since a rectangle with zero width or height has no area.
2. Maximum Allowable Dimensions: We set both width and height to their maximum allowable values, which is 1000. The expected result is 1,000,000 since the area of a square with a side length of 1000 is 1,000,000 square units.

Now, let's create a table to summarize the boundary tests and their expected outcomes:

Test Case	Width	Height	Expected Area
Minimum Allowable Values	0	0	0
Maximum Allowable Values	1000	1000	1,000,000

Figure (142): Summarizing table of boundary tests

By testing these boundary cases, you ensure that the program handles the extreme input values correctly and returns the expected results.

Exercise (60)

Write three Testing Boundary Cases of the function in the following code:

```
def add(x, y):
    return x + y

x = 9
y = 12
```

Figure (143): Testing boundary cases

Answer:

Boundary Test Case 1: Negative Numbers

```
x = -5
y = -3
print(add(x, y))  # Expecting -8
```

Boundary Test Case 2: Zero Values

```
x = 0
y = 0
print(add(x, y))  # Expecting 0
```

Here, we're testing how the function handles the addition of zero values for both `x` and `y`.

Boundary Test Case 3: Large Numbers

```
x = 999999999
y = 1000000000
print(add(x, y))  # Expecting 1999999999
```

This test case examines whether the `add` function correctly handles the addition of large numbers without overflowing or encountering issues.

These boundary test cases cover various scenarios, including negative numbers, zero values, and large numbers, to ensure the `add` function behaves as expected in different situations.

3 Altering the Program Based on Requirements

To alter a program, the modification should affect only the required parts. To do that, the targeted functions should be selected accurately. Afterward, each modified part should enter a new testing process.

Assessment

Q 1. List and differentiate between the different types of errors.

Q 2. True or False:

a) Syntax errors in Python occur when the code violates the language's syntax rules during execution. ( )

b) Logic or semantic errors in Python refer to mistakes in the code's logic that prevent it from correctly solving the intended problem. ( )

c) Runtime errors in Python occur when the code attempts an illegal operation during execution, such as dividing by zero. ( )

d) In Python, you can set a breakpoint by using debugging tools or IDE-specific commands. ( )

e) In Python, F11 is often used to step into the current line of code when using debugging tools or IDEs. ( )

f) In Python, F5 is commonly used to continue or run the program, not specifically to step over a line of code. ( )

g) In Python, Shift+F11 is used to step out of the current line of code when using debugging tools or IDEs. ( )

Q 3. In your way, what is the functionality of the debugger tool?

Q 4. Make a Python project on VS code:

Creating a Calculator:

Objective: In this exercise, you'll create a simple calculator program that can perform basic arithmetic operations such as addition, subtraction, multiplication, and division.

Instructions:

1. Open VS code and create a new Python project.
2. Create a function `add` that takes two numbers as input and returns their sum.
3. Create a function `subtract` that takes two numbers as input and returns their difference.

4. Create a function `multiply` that takes two numbers as input and returns their product.
5. Create a function `divide` that takes two numbers as input and returns their quotient.
6. In a new code cell, implement a simple user interface that allows the user to choose an operation (addition, subtraction, multiplication, or division) and input two numbers.
7. Use conditional statements (if-elif-else) to determine which operation to perform based on the user's choice.
8. Call the corresponding function (add, subtract, multiply, or divide) to perform the selected operation.
9. Display the result to the user.
10. Test your calculator by performing various calculations, and make sure it handles division by zero gracefully.
11. Add comments and documentation to explain how the code works.
12. Run your code cells to test the calculator.

Optional Challenges:

1. Implement error handling to handle invalid inputs gracefully (e.g., non-numeric inputs).
2. Add more advanced operations (e.g., square root, exponentiation) to your calculator.
3. Create a loop that allows the user to perform multiple calculations without restarting the program.

This exercise will help you practice Python programming in VS code and build a simple project that demonstrates the use of functions, user input, and conditional statements.

Q 5. Show two different ways to inspect variable values during debugging.

Answer:

Q 1.

In Python, errors can be categorized into three main types: syntax errors, runtime errors (exceptions), and logical errors.

Let's differentiate between these types:

Syntax Errors:

- **Description:** Syntax errors occur when there is a violation of the language's syntax rules. These are typically caused by incorrect use of Python's syntax elements, such as missing colons, unmatched parentheses, or incorrect indentation.
- **Example:**

```
if x > 5  # Missing colon at the end of the line
    print("x is greater than 5")
```

- **Resolution**: Syntax errors need to be fixed before the program can run. Review the code and correct any syntax violations.

Runtime Errors (Exceptions):

- **Description:** Runtime errors, also known as exceptions, occur during program execution when an unexpected situation or error condition arises. These errors can be caused by various factors such as invalid input, division by zero, or trying to access a non-existent file.
- **Example:**

```
num1 = 10
num2 = 0
result = num1 / num2  # Division by zero error (ZeroDivisionError)
```

- **Resolution:** To handle runtime errors, you can use try-except blocks to catch and handle exceptions gracefully. Proper exception handling helps prevent program crashes and allows for error recovery.

Logical Errors:

Description: Logical errors, also known as semantic errors, occur when the code does not behave as intended due to flawed logic or incorrect algorithmic decisions. These errors may not cause the program to crash but can produce incorrect results.

- **Example:**

```
def calculate_average(numbers):
    total = 0
    for num in numbers:
        total += num
    average = total / len(numbers)
    return average
```

In this example, the logical error is that the division should be performed by the total number of elements, not the length of the list.

- **Resolution:** Identifying and fixing logical errors requires careful debugging and testing. Review the code's logic and algorithm to ensure it aligns with the intended behavior.

It's essential to understand and differentiate these error types to effectively debug and maintain Python programs. Using debugging tools, writing proper exception handling, and thorough testing can help address and prevent these errors in your code.

Q 2.

a) Syntax errors in Python occur when the code violates the language's syntax rules during execution. (True)

b) Logic or semantic errors in Python refer to mistakes in the code's logic that prevent it from correctly solving the intended problem. (True)

c) Runtime errors in Python occur when the code attempts an illegal operation during execution, such as dividing by zero. (True)

d) In Python, you can set a breakpoint by using debugging tools or IDE-specific commands. (True)

e) In Python, F11 is often used to step into the current line of code when using debugging tools or IDEs. (True)

f) In Python, F5 is commonly used to continue or run the program, not specifically to step over a line of code. (True)

g) In Python, Shift+F11 is used to step out of the current line of code when using debugging tools or IDEs. (True)

Q 3.

1. Code Debugging: The debugger allows you to identify and resolve issues in your Python code by stopping the execution at specified breakpoints or exceptions. This helps you inspect variables, evaluate expressions, and understand how your code behaves during runtime.
2. Stepping Through Code: You can step through your code line by line using commands like Step Into (F11), Step Over (F10), and Step Out (Shift+F11). This helps you understand the flow of execution and pinpoint problem areas.
3. Variable Inspection: While debugging, you can inspect the current values of variables, data structures, and objects. This is crucial for identifying incorrect or unexpected values that may be causing issues.
4. Expression Evaluation: You can evaluate expressions and run code snippets during debugging. This is useful for checking intermediate results or experimenting with fixes.
5. Breakpoints: You can set breakpoints in your code at specific locations where you want to pause execution and inspect variables. Breakpoints help you focus on specific parts of your code and investigate them in detail.
6. Watch Window: You can add variables or expressions to the watch window to monitor their values as you step through the code. This allows you to keep an eye on critical variables without manually inspecting them.
7. Call Stack: The debugger provides a call stack view that shows the hierarchy of function calls. This is useful for tracking how your program arrived at its current state.
8. Exception Handling: If an exception is raised during execution, the debugger can catch and display it, allowing you to diagnose and handle exceptions gracefully.
9. Configuration Options: You can configure various debugging options, such as enabling/disabling specific exceptions, specifying launch configurations, or setting up remote debugging for applications running on remote servers or devices.

Overall, the debugger tool in Python VS Code empowers developers to effectively troubleshoot and debug their code, ensuring that it runs correctly and reliably.

Q 4.

```python
# Define the add function
def add(x, y):
    return x + y

# Define the subtract function
def subtract(x, y):
    return x - y

# Define the multiply function
def multiply(x, y):
    return x * y

# Define the divide function
def divide(x, y):
    if y == 0:
        return "Cannot divide by zero"
    return x / y
```

```python
# User interface
print("Simple Calculator")
print("Select operation:")
print("1. Addition")
print("2. Subtraction")
print("3. Multiplication")
print("4. Division")

# Input from the user
choice = input("Enter choice (1/2/3/4): ")

# Check if the choice is valid
if choice not in ['1', '2', '3', '4']:
    print("Invalid choice")
else:
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    if choice == '1':
        result = add(num1, num2)
        operation = "+"
    elif choice == '2':
        result = subtract(num1, num2)
        operation = "-"
    elif choice == '3':
        result = multiply(num1, num2)
        operation = "*"
    else:
        result = divide(num1, num2)
        operation = "/"

    print(f"{num1} {operation} {num2} = {result}")
```

This code defines four basic arithmetic functions (add, subtract, multiply, and divide) and allows the user to choose an operation and input two numbers. It performs the selected operation and displays the result. The code also handles division by zero by checking if the second number is zero.

Q 5.

When debugging a Python program in a development environment like Visual Studio Code, you can inspect variable values in different ways. Here are two common methods:

1. **Using the Debug Console:**
   - While debugging, you can open the Debug Console in Visual Studio Code. This console allows you to enter Python expressions and view variable values interactively.
   - To open the Debug Console, click the "Debug Console" tab at the bottom of the window or press `Ctrl+Shift+Y`.
   - Once the Debug Console is open, you can type variable names and expressions to see their values. For example:

```
x
y + z
```

Press `Enter` after typing each expression to evaluate it and display the result.

2. **Using Hover and Watch:**
   - Visual Studio Code provides a convenient way to inspect variables directly in your code while debugging.
   - You can hover your mouse pointer over a variable in your code, and a tooltip will appear, showing the current value of that variable.
   - Additionally, you can set up "watches" in the Watch panel. To do this:

- While debugging, open the "Watch" panel by clicking the "Watch" tab in the Debug Side Panel or using the Ctrl+Shift+V shortcut.
- Click the "+" button to add a new watch expression.
- Enter the variable name or expression you want to watch and press Enter.
- Visual Studio Code will continuously display the value of the watched variable as you step through your code.

These methods allow you to inspect variable values interactively and keep track of how they change as you debug your Python program. You can choose the one that best suits your debugging workflow and preferences.

Final Assessment

Q 1. What is meant by encapsulation in the OOP?

Q 2. What are the advantages of object-oriented programming?

Q 3. What are the main concepts of functional programming?

Q 4. Why do we use the class diagram?

Q 5. Why do we use the use case diagram?

Q 6. Why do we use the sequence diagram?

Q 7. What are the submodules of designing the architecture?

Q 8. Why is the testing phase very important in the SDLC?

Q 9. List three different SDLC models.

Q 10. Fill in the blanks:

1. Each class in the class diagram should contain ..... and ..... for this class.
2. Each user dealing with the system is considered an ..... in the use case diagram.
3. The sequence diagram is one of the behavioral UML diagrams that show the ..... between the ..... and the ..... or between the system items.
4. Structured programming enhances the code flow and quality by utilizing ....., ..., and ..... .
5. Subroutines in structured programming are also known as ....., ..., or ..... .
6. Object-oriented programming follows a set of concepts which are ....., ..., ..., and ..... .

Q 11. Write a Python program to check if the input number is prime or not.

Q 12. Write a Python program to print the count of prime numbers within a range.

Q 13. Write a Python program to calculate X raised to the power n(Xn).

Q 14. Write a Python program to print the following shapes where the

- .....

Answers

Q 1.

Encapsulation in the OOP means that all the information is contained in an object and only a set of information can be shared with the rest of the code. This concept ensures security and prevents data corruption.

Q 2.

1. Code reuse is simplified by using objects.
2. The modularity of objects means that they can be easily developed, updated, and shared.
3. Debugging is easier when objects are in use and a coding issue arises.
4. Objects can be deployed without revealing the details of their implementations.

Q 3.

1. Pure functions.
2. Recursion.
3. Referential transparency.

Q 4.

1. It is simple.
2. Static representation of the software features (functions).
3. It is a visualized diagram.
4. It represents the relationship between the different objects in the project.
5. It lets you write your own software code functions.
6. It lets us identify the functionality of the software project.

Q 5.

1. It is dynamic.
2. It identifies and describes each interaction (visually) in the project.
3. It enables the project owner to preview the roles of the project in the graphical representation besides the documentation.
4. You could see the different types of users of the project.
5. Because of the relationship between the different users.

Q 6.

1. Because of dynamicity and time.
2. It shows the event across its lifetime.
3. It displays the different messages during the event's lifetime.
4. It could be used by the software developers or the project stakeholders.

Q 7.

1. Architecture.
2. User interface.
3. Platforms.
4. Programming.
5. Communication.
6. Security.

Q 8.

The testing phase aims to minimize the number of failures in the system. This leads to customer satisfaction and higher development quality.

Q 9.

1. Iterative model.
2. Waterfall model.
3. Agile model.

Q 10.

1. attributes - methods
2. actor
3. communication - system - users
4. control flow - repetition - subroutine structures
5. procedures - methods - functions
6. inheritance - polymorphism - abstraction - encapsulation.

Q 11.

To check if a given number is prime or not, you can use the following Python program:

```python
# Function to check if a number is prime
def is_prime(num):
    # Check if the number is less than 2 (not a prime number)
    if num < 2:
        return False

    # Check for factors from 2 to the square root of the number
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False

    # If no factors were found, the number is prime
    return True

# Input from the user
try:
    num = int(input("Enter a number: "))
    if is_prime(num):
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is not a prime number.")
except ValueError:
    print("Invalid input. Please enter a valid number.")
```

This program defines a function `is_prime` that checks whether a number is prime or not. It iterates through numbers from 2 to the square root of the input number and checks for factors. If no factors are found, the number is considered prime.

Remember to handle invalid inputs (e.g., non-integer inputs) to ensure your program works correctly.

Q 12.

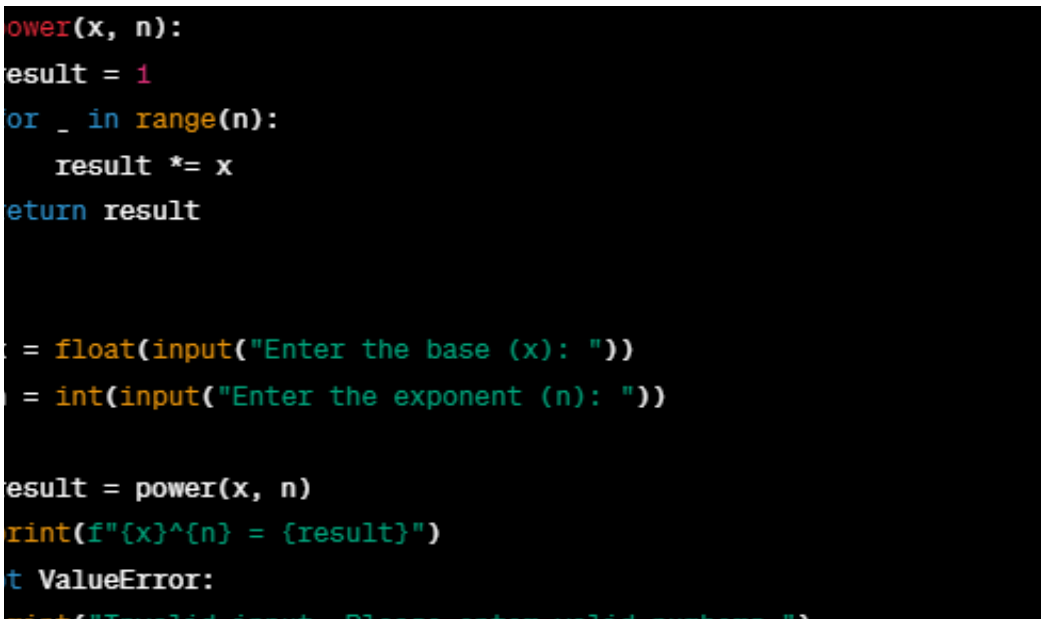
```
# Function to check if a number is prime
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

# Input range from the user
try:
    start = int(input("Enter the start of the range: "))
    end = int(input("Enter the end of the range: "))

    if start > end:
        print("Invalid range. Start should be less than or equal to end.")
    else:
        count = 0
        for num in range(start, end + 1):
            if is_prime(num):
                count += 1
        print(f"The count of prime numbers between {start} and {end} is {count}")
except ValueError:
    print("Invalid input. Please enter valid integers for the range.")
```

Q 13.

Here's a Python program that calculates x raised to the power of n (x ^ n):



```
ower(x, n):
esult = 1
or _ in range(n):
   result *= x
eturn result


 = float(input("Enter the base (x): "))
 = int(input("Enter the exponent (n): "))

esult = power(x, n)
rint(f"{x}^{n} = {result}")
t ValueError:
```

This program defines a `power` function that calculates the result of `x` raised to the power of `n`, and then it takes input from the user for `x` and `n`, calculates the result, and prints it.

Q 14.

To print shapes with a given height in Python, you can use loops. Here's a Python program that prints two different shapes: a right-angled triangle and an inverted right-angled triangle, both based on the height provided as input:

```
# Get the height of the shape from the user
height = int(input("Enter the height of the shape: "))

# Print a right-angled triangle
print("Right-Angled Triangle:")
for i in range(1, height + 1):
    print('*' * i)

# Print an inverted right-angled triangle
print("\nInverted Right-Angled Triangle:")
for i in range(height, 0, -1):
    print('*' * i)
```

This program first takes the height of the shape from the user as input. Then, it uses loops to print the right-angled triangle and inverted right-angled triangle patterns. The loops iterate from 1 to the height and from the height down to 1, respectively, printing asterisks (*) to form the shapes.

References

1.

1. Introduction to Computer Science: A Textbook for Beginners in Informatics by Gilbert Brands.
2. https://www.oreilly.com/library/view/learning-python/1565924649/