TOPIC 3: INTRODUCTION TO COMPUTER PROGRAMMING

BY

SIR. SAGENGE SAMIKE

MEANING OF COMPUTER PROGRAMMING

- **Programming:** Is the process of creating a set of instructions that a computer can follow to perform specific tasks or solve problems.
- It involves writing code in a language that a computer can understand and execute.

- **Computer Programming**: Is the process of writing instructions that a computer can understand and execute. These instructions are written in a programming language and tell the computer what to do step by step to complete a task or solve a problem.

REASONS FOR LEARNING PROGRAMMING

i. It improves problem solving skills
ii. It enhances creativity
iii. It is a foundation of most modern technologies
iv. It helps in automating repetitive tasks
v. It boosts career prospects
vi. Its skills can be applied in various fields
vii. It develops critical thinking

IMPORTANCE OF PROGRAMMING

i. It helps in developing technologies used in daily life
ii. It helps in solving real life problems
iii. It helps in generating new ideas
iv. Its skills can lead to good career opportunities
v. It help to understand how computers and technology work
vi. It is used in industries to make work faster and easier
vii. It helps to reduce mistakes
viii. It makes global communication possible through the use of Internet and social media
ix. It allows people to create and adjust software to meet specific needs
x. It is helpful in science and education

ROLE OF PROGRAMMING LANGUAGES

- **A programming language:** Is a set of rules and instructions that a computer can understand and execute.
- It is like a language humans use to communicate with computers, telling them what to do, how to do it, and when to do it. Programming languages are essential tools in software development enabling developers to create, modify and maintain software applications.
- Examples of programming languages include Python, Java, JavaScript, C,C#,C++, Swift and PHP

- The roles of programming languages in software development includes the following:-
- They allow developers to write step by step commands that instruct computers to perform specific task efficiently
- They enable creation of various types of software
- They increase productivity and efficiency by automating repetitive processes
- They aid in software security
- They enable developers to write higher performance code that optimizes software efficiency, reduce processing time and improves responsiveness

THINGS THAT MAKES A COMPUTER WORK

- A computer performs its function because of the following

i. Hardware
ii. Software

HARDWARE

- **Hardware** refers to the physical parts of a computer that can be seen and touched. In a computer, hardware carries out the tasks while software provides the instructions to follow.
- The following are the main hardware found in a computer, namely
- **CPU(Central Processing Unit):** This is a computer's brain that follows instructions in programs
- **RAM(Main Memory):** It temporarily stores data that are currently used by a computer
- **Storage(Hard Drive or SSD):** It saves data for future use
- **Input devices:** They helps a user to enter commands into a computer
- **Output devices:** They are used to show or display results

SOFTWARE

- **Software** is the collection of instructions or programs that direct a computer's hardware to perform specific tasks.
- Software includes all the apps and programs that run in a computer.

TYPES OF SOFTWARE

- There are two types of software, namely
- System Software
- Application Software

DATA STORAGE ON COMPUTERS

- Computers use binary (0s and 1s) to store data such as words, pictures, and music.
- Example:
  - Letter **"A"** is stored as **01000001**
  - Number **"157"** is stored as **10011101**

NOTE: Bigger numbers need more bytes, computers can combine bytes to store large values like 65,535 or even more.

STORING TEXT, NUMBERS AND MEDIA

- **Texts:** Computers use systems like ASCII and Unicode to convert letters into binary for storage
- **Images:** Computers breaks down an image into millions of tiny dots called **pixels,** each pixel's color information is translated into binary number.

- **Sound:** Sound is saved in small pieces(samples), it is also stored as binary numbers.

HOW A PROGRAM WORKS

- A program consists of many small instructions that the computer must follow. When a program run, these instructions are executed by the Central Processing Unit(CPU).
- The CPU follows the instructions by doing things like:-

i. Reading from memory
ii. Doing calculation
iii. Moving data
iv. Comparing number

- Programs are saved or stored on the hard drive. When they are opened, they are copied to the RAM where a CPU can fetch and execute their instructions during processing.
- Generally, programming is done by the following a few simple steps as follows

i. Writing a list of instructions(also called a Program)
ii. Saving the program on the computer
iii. Running the program which loads into the computer's main memory(RAM)
iv. The CPU reads and follows the instructions to perform the task

IMPORTANT TERM IN PROGRAMMING

- The following are the basic terms used in programming:-

i. Compiler
ii. Interpreter
iii. Source code
iv. Object code
v. Translator

A: COMPILER

- A compiler: Is a special software program that translate code written in a high-level programming language into machine language(binary code) that a computer's CPU can understand and execute.
- It processes the entire source code at once converting it into object code or an executable file.
- Compared to interpreters, compilers require less time during execution as the code is already full compiled.
- Examples of compilers include Dev-C++, Borland++ and Visual C++.

IMPORTANCE OF USING A COMPILER

- It translate source code into machine language
- It detect and report errors in the code
- It is used for generating executable file

EXAMPLES OF COMPILERS FOR DIFFERENT PROGRAMMING LANGUAGES

PROGRAMMING LANGUAGE	COMPILER
C	GCC(GNU Compiler Collection), Clang, Microsoft Visual C++, Turbo C
C++	G++(part of GCC), Clang++, Microsoft Visual C++ (MSVC), Borland C++
Python	Cython, PyInstaller, Nuitka
JavaScript	Babel, Google closure compiler, TypeScript compiler(tsc)
Java	Java compiler(javac),Eclipse compiler for Java(ECJ),GraalVM compiler
PHP	HipHop Virtual Machine(HHVM), PHP Zend engine, RoadRunner
Swift	Swift compiler(swiftc)
Kotlin	Kotlin compiler(kotlinc),IntelliJ IDEA Kotlin compiler
Go	Go compiler(gc), GCC Go
Paschal	Free Paschal Compiler(FPC), Delphi Compiler
Turbo Paschal	Turbo Paschal Compiler
Rust	Rust compiler(rustc)
Fortlan	GNU Fortlan(gfortlan), Intel fortlan compiler,IBM XL Fortlan
R	R Compiler(Bytecode Compiler), GCC for R

B: INTERPRETER

- An Interpreter: Is a type of translator that converts source code into machine code one statement at a time.
- Unlike a compiler, an interpreter does not translate the entire high-level program at once, instead it processes each line sequentially during execution that takes a bit longer.
- Programming languages that use interpreters include Perl, Rubby, Python and PHP.

C: SOURCE CODE

- Source code: Is a collection of computer instructions written by a programmer using a high-level or intermediate programming language.
- Source code are also simply called **Code**
- Example of Source code written in C++:

```
void main()
  {
  cout<< "What is your name?";
  }
```

D: OBJECT CODE

- **Object code:** Is a machine-readable code in a binary format(0s and 1s) that is obtained after converting source code.

- The object code is also called machine code because it can be only read and understood by the processor.

```
Source Code
void main()
    {
    cout<< "The year 2022?";
    }

Object code
011101001010
101001010001
010111001110
110010010010
```

PROGRAMMING ENVIRONMENT

- **Programming environment:** Is the space where developers write, test, and run their code.
- It includes tools that make coding easier and more efficient.

TYPES OF PROGRAMMING ENVIRONENT

- There are different types of programming environments depending on how and where coding is done. The following are the two main types of programming environment:-

i. Integrated Development Environment(IDE)
ii. Online code editors
iii. Text editors

A: INTEGRATED DEVELOPMENT ENVIRONMENT(IDE)

- Is a software that helps programmers write, test, and fix their code more easily.
- It combines several tools into one platform to make coding more faster and more organized. Examples of IDE include PyCharm, Visual Studio and Eclipse.
- An IDE usually includes the following tools:-

i. **Code editor:** This is where a programmer write the code. It makes coding easier by highlighting important parts, suggesting possible completions and making the code look neat.
ii. **Compiler or Interpreter:** These are tools that turn the code into instructions can understand and run.
iii. **Debugger:** These are tools that help a programmer to find and fix errors made in the code.
iv. **Build tools:** These are tools that help a programmer to automate common tasks such as checking for errors, testing code or running a program.

B: ONLINE CODE EDITORS

- **An online code editor:** Is a web-based tool that allows programmers to write and run code directly in a web browser without installing any software.
- Online code editors are great for beginners, students, and people who want to code on the go.
- Examples of Online code editors include:-

❖ **Replit:** Is a tool that supports multiple programming languages and allows users to save projects online. Replit also enables team collaboration.

❖ **Scratch:** Is a visual coding platform designed for beginners especially children to create animations and games using drag-and-drop blocks.

❖ **JSFiddle:** Is a web-based tool for experimenting with JavaScript, HTML and CSS.

FEATURES OF ONLINE CODE EDITORS

- The following are some features of online code editors:-

i. No installation is required
ii. They are accessible from any device with an Internet connection
iii. They provide built-in collaboration features for group coding projects.

C: TEXT EDITORS

- **A text editor:** Is a tool where a programmer type and modify programs.
- Modern text editors provide syntax highlighting, auto-completion, and error detection to improve code efficiency.

C: TEXT EDITORS

- Examples of commonly used text editors include:-

i. Basic editors: These include editors like Notepad(Windows), Nano(Linux), vi/vim(Linux/MacOS)
ii. Advanced code editors: These include Visual Studio Code(VS Code), Sublime Text, Atom
iii. Integrated editors in IDES: Code::Blocks, Dev-C++, Clion

LIBRARIES AND FRAMEWORKS

- **Libraries:** Are collection of pre-written code that programmers can use to perform common tasks without writing the code from scratch.
- Libraries can be used to perform tasks like database connection, creating a user interface, and handling files.

EXAMPLE OF LIBRARIES FOR DIFFERENT PROGRAMMING LANGUAGE

PROGRAMMING LANGUAGE	LIBRARIES
Python	NumPy, Pandas, Matplotlib, and Random
Java	java.util, java.io, and java.lang
C	stdio.h, math.h
JavaScript	JQuery, React, and Lodash
C++	iostream, fstream, and string

- **A framework:** Is a ready-made structure that provides tools and rules for building applications, making development faster and more organized.
- It is pre-built structure that provides a foundation for developing software applications.
- Example of frameworks: Django, and Flask(Python), Angular, and Vue.js(JavaScript), Spring(Java).

DEBUGGING TOOLS

- **Debugging tools:** Are tools used to find and fix errors or bugs in code.
- Debugging tools help programmers step through their code to identify the source of the problem.
- **NOTE:** Debugging tools are sometimes known as **Debuggers.**

INTRODUCTION TO PROGRAMMING LANGUAGE

- A programming language is a tool for communicating with a computer. Different programming languages are suited for different tasks.

OVERVIEW OF PROGRAMMING LANGUAGES

- A programming language is a formal language comprising a set of instructions that can be used to produce various kinds of outputs.
- It enable humans to communicate commands to a computer, allowing the creation of software programs, control of machine behavior and expression of algorithms.

TYPES OF PROGRAMMING LANGUAGES

- There are two types of programming languages, namely

i. Low-level languages
ii. High-level languages

A: LOW LEVEL PROGRAMMING LANGUAGES

- **A low-level programming language:** Is a programming language that provides little or no abstraction from a computer's hardware, offering direct control over CPU, memory and registers.
- Low-level programming languages are characterized by little efforts to translate programs into machine readable form.
- They are not portable, since they cannot easily be transferred from one computer to another. Low-level languages are machine dependent and do not require any compiler to translate them into machine code.

TYPES OF LOW-LEVEL PROGRAMMING LANGUAGES

- There are two types of low-level programming languages, namely

i. Machine language
ii. Assembly language

MACHINE LANGUAGE

- **Machine language:** Is the low-level programming language which consists of binary digits(0s and 1s) that directly instruct a computer's CPU.
- It is the only language hardware understands, requiring no translation to execute tasks loading data or performing arithmetic.
- Machine language is complex for humans to understand and use as a primary programming language since every instruction is written in a series of 0s and 1s.

ASSEMBLY LANGUAGE

- **Assembly language:** Is the low-level programming language that uses human-readable mnemonics to represent machine code instructions.
- Machine language is a step above machine language and is closer to how computers operate.
- Since every computer has its own architecture, each one has its own set of instructions making assembly language unique to that specific machine.

DIFFERENCE BETWEEN MACHINE LANGUAGE AND ASSEMBLY LANGAUGE

MACHINE LANGUAGE	ASSEMBLY LANGUAGE
It is only understood by the computers	It is only understood by human beings
Data is represented in binary formats(0s and 1s)	Data is represented in binary or hexadecimal
It does not allow modifications and error fixing	It allow modifications and error fixing
It is very difficult to use	It is easy to use compared to machine language
Execution is faster since data are represented in binary format	Execution is slower compared to machine language

ADVANTAGES OF LOW-LEVEL PROGRAMMING LANGUAGES

- They are used for designing programs which require less memory to be executed
- They do not need compilers or interpreters to change into machine-readable form
- They provides direct interaction with the computer's internal memory

- Low-level languages provide conducive environment for utilizing the processor
- Low-level languages communicates directly with computer hardware

DISADVANTAGES OF LOW-LEVEL PROGRAMMING LANGUAGE

- Low-level languages require high expertise in programming to design a program
- Programs created by low-level languages are not portable
- Low-level languages have challenging environment for developing and executing programs
- It is difficult to debug errors in low-level languages

B: HIGH-LEVEL PROGRAMMING LANGUAGES

- **High-level programming languages:** Are programming language that are used for writing programs or software that can be understood by humans and computers.
- High-level languages are machine independent, meaning that programs written in these languages can run on different computer systems without modification.
- Unlike low-level languages, high-level languages are closer to human languages than machine language making them easier to read, write and understand.
- Examples of high-level languages: Paschal, Fortran, COBOL, C, C++, Java, JavaScript, Python and Visual Basic.

TYPES OF HIGH-LEVEL PROGRAMMING LANGUAGES

- High-level programming languages are categorized into the following categories based on their approach

i. Procedural languages ii. Object-oriented languages
iii. Functional languages iv. Scripting languages
v. Logic-based languages vi. Domain-Specific Languages(DSLs)

I: PROCEDURAL LANGUAGES

- **Procedural languages:** Are languages that executes instructions sequentially step by step.
- Procedural languages are ideal for structured programming, where programs are organized into procedures or functions to enhance readability and maintainability.
- Examples of procedural language include C, Paschal and Fortran.

II: OBJECT-ORIENTED LANGUAGES

- **Object-oriented languages:** Are programming languages which bases on the concept of objects which encapsulate both data and behaviour.
- They supports key principles such as encapsulation, inheritance and polymorphism. Examples of Object-Oriented Programming language(OOP) include Java, C++ and Python

III: FUNCTIONAL LANGUAGES

- **Functional languages:** Are programming languages that thinks of programs as math functions.
- They encourage writing pure functions enhancing program reliability and minimizing side effects.
- Examples of functional languages include: Haskell, Lisp and F#

IV: SCRIPTING LANGUAGES

- **Scripting languages:** Are programming languages designed to automate the execution of tasks.
- These languages are typically interpreted, meaning that they are executed line by line by an interpreter rather than being compiled into machine code.
- Examples of scripting languages include Python, JavaScript and Perl.

V: LOGIC-BASED LANGUAGES

- **Logic-based languages:** Are programming languages that are used to write programs in a set of logical statements that defines relationship and rules rather than sequential instructions.
- They are commonly used in artificial intelligence and expert systems, where reasoning and inference are essential.
- Examples of logic-based languages include Prologue, Datalog and Mercury.

VI: DOMAIN-SPECIFIC LANGUAGES

- **Domain-Specific Languages:** Are programming languages designed for specific tasks rather than general purpose programming.
- Examples of Domain-specific languages include HTML(Hypertext Markup Language) and SQL(Structured Query Language).

ADVANTAGES OF HIGH-LEVEL PROGRAMMING LANGUAGES

i. They provide an easy way to recognize programs prone to errors
ii. They are machine independent and portable
iii. They provide a conducive environment for developing, executing and maintaining a program
iv. They provide an easy way to learn and use them
v. They are widely used for programming since they can run on different platforms

DISADVANTAGES OF HIGH-LEVEL PROGRAMMING LANGUAGES

i. Programs developed using high-level languages need to be interpreted or compiled to change them into machine readable form

ii. They are not fast as changing a program from source code to object code requires time
iii. They overload the processor with several typed statements
iv. They need enough memory to be maintained and executed

DIFFERENCE BETWEEN HIGH-LEVEL LANGUAGE AND LOW-LEVEL LANGUAGE

FEATURE	HIGH-LEVEL LANGUAGE	LOW-LEVEL LANGUAGE
Level of abstraction	Provides high level of abstraction from the hardware	Provides less or no abstraction from the hardware
Execution speed	It is slower than low-level languages	It is faster than high-level languages
Memory efficiency	It consumes more memory	It consumes less memory
Translation	It requires a compiler or interpreter to convert the program into machine code	It requires an assembler to convert the program into machine code
Comprehensibility	It is easily understandable by the programmer	It is easily understandable by the computer
Machine dependency	It is machine-independent	It is machine-dependent
Portability	It is portable	It is not portable
Debugging and Maintenance	It is easy to debug and maintain	It is difficult to debug and maintain
Examples	Python, Java, C++, Rubby	Machine language, Assembly language

GENERATIONS OF PROGRAMMING LANGUAGES

- Programming languages are classified into five generations based on their level of abstraction and capabilities.
- The following are the five generations of programming languages:-

i. First generation
ii. Second generation
iii. Third generation
iv. Fourth generation
v. Fifth generation

A: FIRST GENERATION(1GL)

- First generation languages are also called **Machine Language** or **Machine Code.** They are called machine language or machine code because are the only language directly understood by the computer.
- First generation languages are the lowest level of programming language, their statements are written in binary code(0s and 1s) that the computer's CPU executes directly.
- Programs written in machine language(first generation language) do not require translation simply because the computer natively understand binary code.
- The table below shows an example of a program written in machine language that adds two numbers stored in memory locations **0** and **1**, with the result stored in location **2**.

Location of instruction	Machine instruction															
	Function code				Operand address											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
1	0	0	0		0	0	0	0	0	0	0	0	1	0	1	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

B: SECOND GENERATION(2GL)

- Second generation language is also called **Assembly Language**.
- **Assembly language:** Is a low-level programming language that uses human-readable mnemonics to represent the specific underlying machine code instruction.
- Unlike Machine Language which uses binary like 101010 to represent data, assembly language uses human-readable symbols(mnemonics) such as:
  - MOV: Move data
  - ADD: Addition
  - SUB: Subtraction
- Mnemonics code is made up of two or three letters such as LDN, STA, and JPU
- NOTE: Typical features of the assembly language differ from on computer to another

FEATURES FOR ASSEMBLY LANGUAGE(SECOND GENERATION)

i. It is machine-dependent ii. It uses mnemonics instead of binary code
iii. It require an assembler to translate it into machine language iv. It gives direct control over hardware

Example of a program written in assembly language

PROGRAM	MEANING OF CODE
1. MOVESTACK INDEX	1. Save the current base address of the stack in the index register
2. PUSH FIRST	2. Push the first number on the stack
3. PUSH SECON	3. Push the second number on the stack
4. INC STACK	4. Make space on the stack for the third number
5. JSR OAA	5. Jump to a subroutine which orders and adds
6. POP TOTAL	6. Remove the total from the stack
7. POP SECON	7. Remove the second number from the stack
8. POP FIRST	8. Remove the first number from the stack

C: THIRD GENERATION(3GL)

- Third generation languages are also called **High-level languages.**
- **Third generation languages:** Are programming languages that allow programs to be divided into subprograms(procedures). They are often called Structured or Procedural languages.
- Examples of Third generation languages include Paschal, Fortran, COBOL, BASIC, Ada and C.

D: FOURTH GENERATION(4GL)

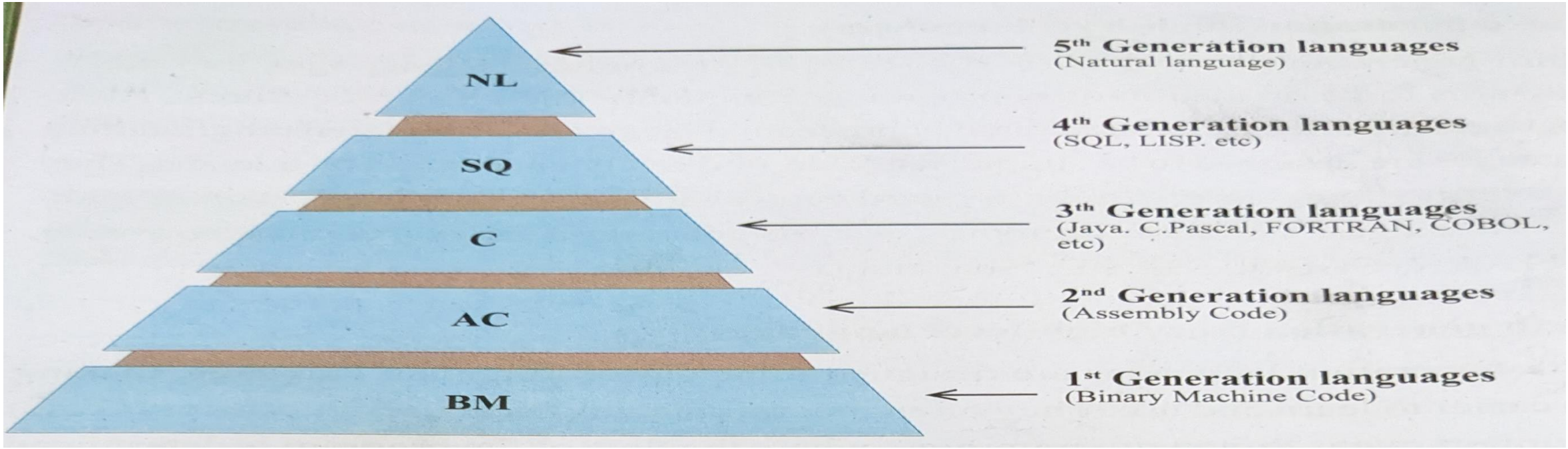
- Fourth generation languages are also called **Very high-level languages.**
- **Fourth generation languages:** Are programming languages designed that are designed to be closer to human language and easier to use than earlier generations.
- They often incorporate graphical user interfaces (GUIs) and support event-driven programming where the flow of the program is determined by user actions such as mouse clicks and keyboards input.

- Fourth generation languages frequently uses drop-down menus and forms to facilitate user interaction. Examples Python, Rubby, SQL, PHP, JavaScript, and Visual Basic.

E: FIFTH GENERATION(5GL)

- **Fifth generation languages:** Are programming language that are designed to enable computers to solve problems with minimal or no human intervention.
- Unlike traditional programming languages that require developers to write detailed step-by-step algorithms, fifth generation languages focus on defining problems using constraints and logical relationships.
- Fifth generation languages are used in fields like Artificial intelligence and Expert systems.

HIERARCHY OF GENERATION OF LANGUAGES



EXAMPLES OF POPULAR PROGRAMMING LANGUAGES

1. **Scratch:** Is a visual programming language that allows a user to create animations, games, and interactive stories using drag-and-drop blocks instead of writing code. It is a beginner-friendly visual programming language specially designed for children.

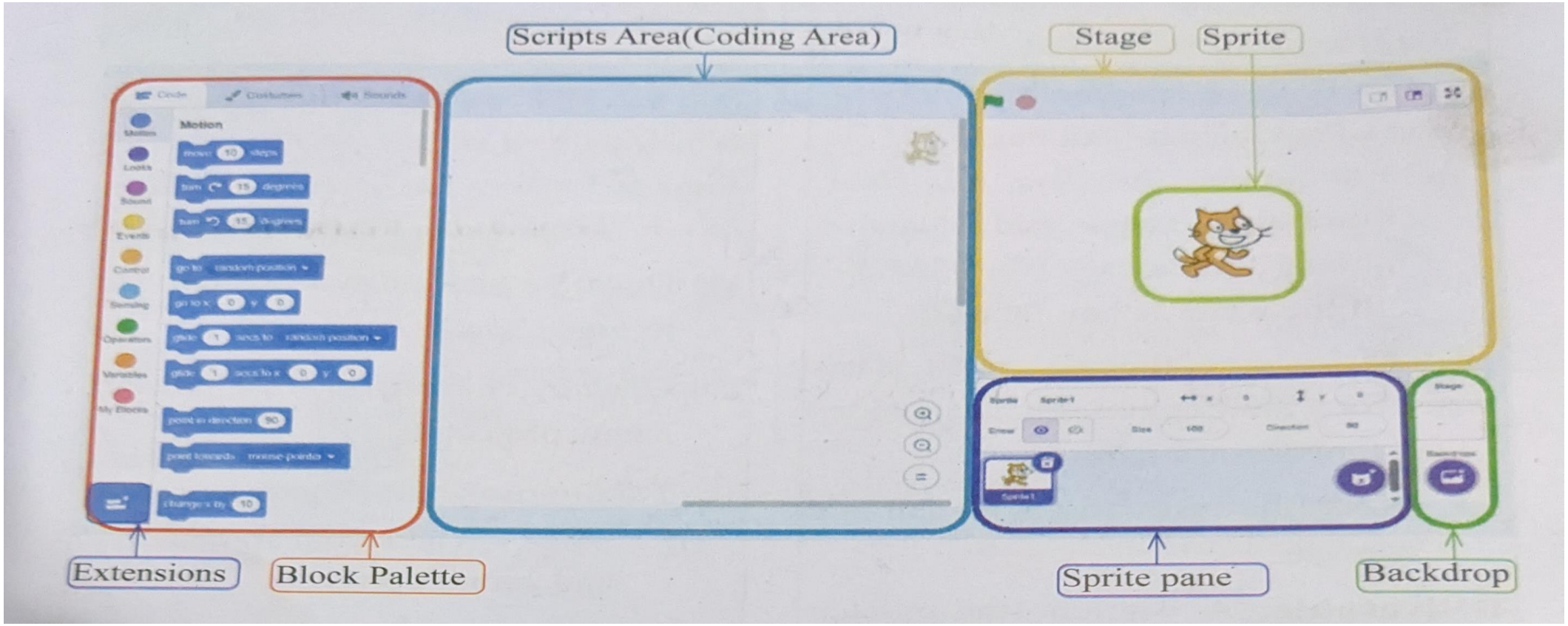
KEY AREAS OF THE SCRATCH INTERFACE

- **Stage:** Is the area where animations and games appear
- **Sprites:** Are characters or objects that perform actions
- **Blocks palette:** Is the area that contains different programming blocks
- **Script area:** Is the area where a user connect blocks to create actions. It is also called Coding Area

BASIC SCRATCH BLOCKS

- **Motion blocks:** Motion blocks are used to control movement of sprites. Example “move 10 steps”, “turn 15 degrees”.
- **Looks blocks:** Looks blocks are used to change the appearance of sprites. Example “say Hello”, “change Color”.
- **Sound blocks:** Sound blocks are used to add sounds and music
- **Events blocks:** Events blocks are used to start actions. Example “when green flag clicked”, “when key pressed”
- **Control blocks:** Control blocks are used to create loops and conditions. Example “repeat 10 times” “if-then”.

SCRATCH INTERFACE



GOOD CODING HABITS IN SCRATCH

- The following are simple habits every programmer should follow when coding using scratch

i. Keep the code neat and organized

- This involves arranging blocks in a logical order, grouping related blocks together to make the program easier to understand, and using meaningful names for sprites, variables, and broadcasts

ii. Reuse blocks and avoid repetition

- This involves creation of custom blocks(functions) for repeated actions instead of copying the same set of blocks many times, and storing repeated values in variables to make future changes easier.

iii. Test the program regularly

- This involves running the project often to spot errors or incorrect behavior early, trying different inputs to make sure the program works as expected, and using simple test cases to verify if custom blocks perform correctly.

iv. Handle errors and an expected behavior

- This involves using conditional blocks, adding clear messages or visual signals to help users know what's happening, and preventing unintended behavior like sprite going off-screen.

v. Break large tasks into smaller steps

- This involves decomposing complex behaviors into smaller custom blocks, and tackling each part separately.

vi. Save and version the work

vii. Keep learning and improving.

2. **Python:** Is a flexible high-level programming language known for being easy to read and write. It is used. It is used across many fields such as Artificial Intelligence(AI), data science, web development and automation.
3. **JavaScript:** Is a dynamic programming language mainly used to create interactive websites. It lets developers add features like animations, forms, and live updates.

- JavaScript is used for both front end(client-side) and back end(server-side) especially with frameworks like Node.js

4. **Java:** Is a robust object-oriented programming language used in developing many applications particularly mobile applications(Android apps) and large-scale enterprise systems. Java programs can run on any system with a Java Virtual Machine (JVM).

4. **C++:** Is a high-performance language based on C that includes object-oriented features. It is commonly used in industries requiring fast and efficient software such as video games, system software, embedded systems and in fields like robotics and high-performance computing because it provides fine control over system resources, making it perfect for tasks where speed and memory management are critical.

ALGORITHMS IN PROGRAMMING

- **An algorithm:** Is a step-by-step procedure or formula used to solve a specific problem. It acts as a set of instructions that guide how to process input and generate output.
- Algorithms are central to all computer programs and applications helping to solve problems in a structured and efficient way.
- There are various types of algorithms designed for different tasks, example
  - Sorting algorithms: Arrange data in a specific order
  - Searching algorithms: Finds an element within a dataset

Examples of algorithms in real-life

1. Making a sandwich
   Step 1: Gete bread, Step 2: Add filling, Step 3: Close the sandwich
2. Finding the largest number in a list
3. Giving directions

DATA TYPES AND DATA STRUCTURES

- **Data type:** Is a classification of data that specifies the kind of data a variable can store and the operations that can be performed on it.

COMMON DATA TYPE USED IN PROGRAMMING

- The common data type used in programming include:
  - **String**: Represents a sequence of characters such as words, phrases, or sentences. Example "Hello World!"
  - **Boolean:** Represents a logical value with only two possible outcomes like true/false, yes/no or on/off.
  - **Numeric types:** Represents data in the form of numbers such as **integer**(whole numbers without decimal points) and **float(**numbers with decimal points)
  - **Characters:** Represents a single character such as a letter, digit or symbol. Example "a", "7", "$"
  - **Array:** Is a collection of multiple values of the same data type stored in an indexed sequence. Example: [1,2,3,4], and ["apple", "banana", "cherry"].

- **Data structures:** refers to the methods of organizing and storing data in a way that allows efficient access and modification.
- The choice for data structure depends on the specific needs of a program, such as how fast it needs to search for data, add new data, or remove data.
- Examples of common data structures include: Stacks, heaps, trees, linked lists, queues, tables, and graphs.

CONTROL FLOW STRUCTURES

- **Control flow structures:** Are statements or constructs that determine the order in which instructions are executed in program.
- They help to manage how the program accesses, processes, and modifies the data within a given data structure.

TYPES OF CONTROL FLOW STRUCTURES

- There are three types of control flow structures, namely

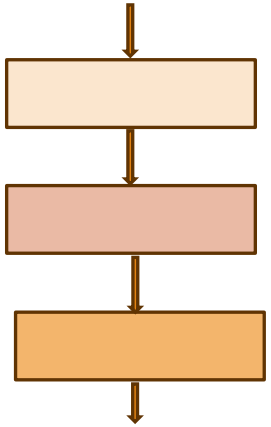
i. Sequence control structure
ii. Selection control structures
iii. Loops

A: SEQUENCE CONTROL STRUCTURE

- **Sequence control structure**: Is the type of control structure in which program instructions are executed one after another from top to bottom in the exact order they appear.
- In sequential control structure, each step runs exactly once.

FLOWCHART FOR SEQUENCE CONTROL STRUCTURE

Sequence

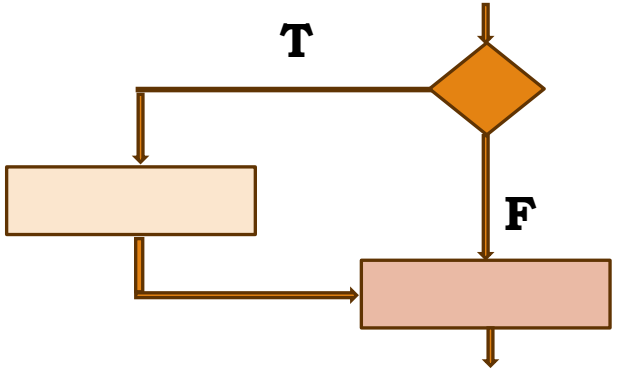


B: SELECTION CONTROL STRUCTURE

- **Selection control structure:** Is the type of control structure which allows a program to make decisions based on certain conditions.
- It directs the program to follow different paths depending on whether the condition is true or false

FLOWCHART FOR SELECTION CONTROL STRUCTURE

Selection

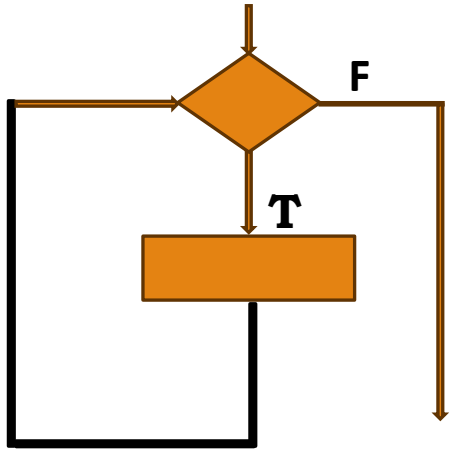


NOTE: Selection control structure sometimes are called **Conditional statements**

C: LOOPS

- **A loop:** Is a control structure that allow a statement or block of code to execute repeatedly as long as a specified condition is true
- Loops are fundamental and powerful because they help to automate repetitive tasks efficiently. Loops sometimes are known as **Iteration** or **Repetition**.

FLOWCHART FOR LOOPS(ITERATION CONTROL STRUCTURE)



TWO COMMON TYPES OF LOOPS

- **For loop:** Is the loop used when the number of iterations is known. In other terms, For loop repeats a block of code for a specific number of times.
- **While loop:** Is the loop used when the number of iterations is unknown. It continues to execute a block of code as long as a specified condition is true.

PROBLEM SOLVING SKILLS(THINKING LIKE A PROGRAMMER)

Breaking down complex problems

- Breaking down a complex problem into smaller manageable pieces, allows programmers to focus on solving one small part of the problem at a time. This leads to more efficient problem-solving and easier debugging.
- Below are the steps that shows how to break a problem when creating a simple game.

a) **Plan the rules and objectives:** At this point, a programmer must ask himself/herself the following questions to figure out what the game will be about
   - What is the objective of the game?
   - How does a player win or lose?
   - What challenges or obstacles will players face?
   - How long will the game last?
   - Will there be levels or stages?
   - What tools or mechanism does the game need?

b) **Define player actions(Move, Jump):** In this step, a programmer must outline the core actions the player can perform. Also, a programmer must specify the exact controls(keyboard, mouse or touch) required for each action

c) **Write code to display graphics and handle player actions:** This is done once the game's rules and actions are defined. This involves rendering the game world, creating characters and objects, and ensuring the game responds to player inputs.

PROGRAM DEVELOPMENT LIFE CYCLE(PDLC)

- **Program Development Life Cycle:** Is a structured process that outlines the key phases involved in developing a program.
- Program Development Life Cycle(PDLC) allows programmers improve efficiency and quality by analyzing and optimizing each stage of development.
- Program Development Life Cycle is also known as **Software Development Life Cycle(SDLC).**

STEPS OF PROGRAM DEVELOPMENT LIFE CYCLE

- Program Development Life Cycle includes the following steps

i. Problem definition
ii. Program designing(Designing the program)
iii. Coding the program
iv. Debugging the program
v. Testing the program
vi. Documenting the program
vii. Program maintenance

I: PROBLEM DEFINITION

- **Problem definition** involves clearly understanding and defining the problem to be solved. At this stage, the software developer identifies the issues, gathers user requirements, and determines what the program is expected to achieve.
- Problem definition is also known as **Problem recognition**

- Example of problem definition
- **Problem:** To calculate average of students
- **Input:** Student score and total number of students
- **Output:** Average score

II: PROGRAM DESIGNING

- **Problem design** is the second stage of the Program Development Life Cycle focusing on creating a system that meets the requirements defined during problem definition.
- The design tools that can be used in this stage include flowcharts, hierarchy diagrams, pseudocode, or entity-relationship diagrams(ERD).

III: CODING THE PROGRAM

- **Coding** is the third stage of the Program Development Life Cycle. At this stage the programmer begins writing the actual program using a chosen programming language.

IV: DEBUGGING THE PROGRAM

- **Debugging** is the process of identifying and fixing errors(bugs) in a program to ensure it runs correctly.
- The errors can be compile-time(like syntax mistakes) or run-time such as division by zero.

V: TESTING THE PROGRAM

- **Testing** is the fifth stage of the Program Development Life Cycle. The types of testing that can be done include the following
  - **Functionality testing:** It covers user interface, security and database
  - **Usability testing:** It involve target users to assess easy of use
  - **Compatibility testing:** It involves checking performance across devices

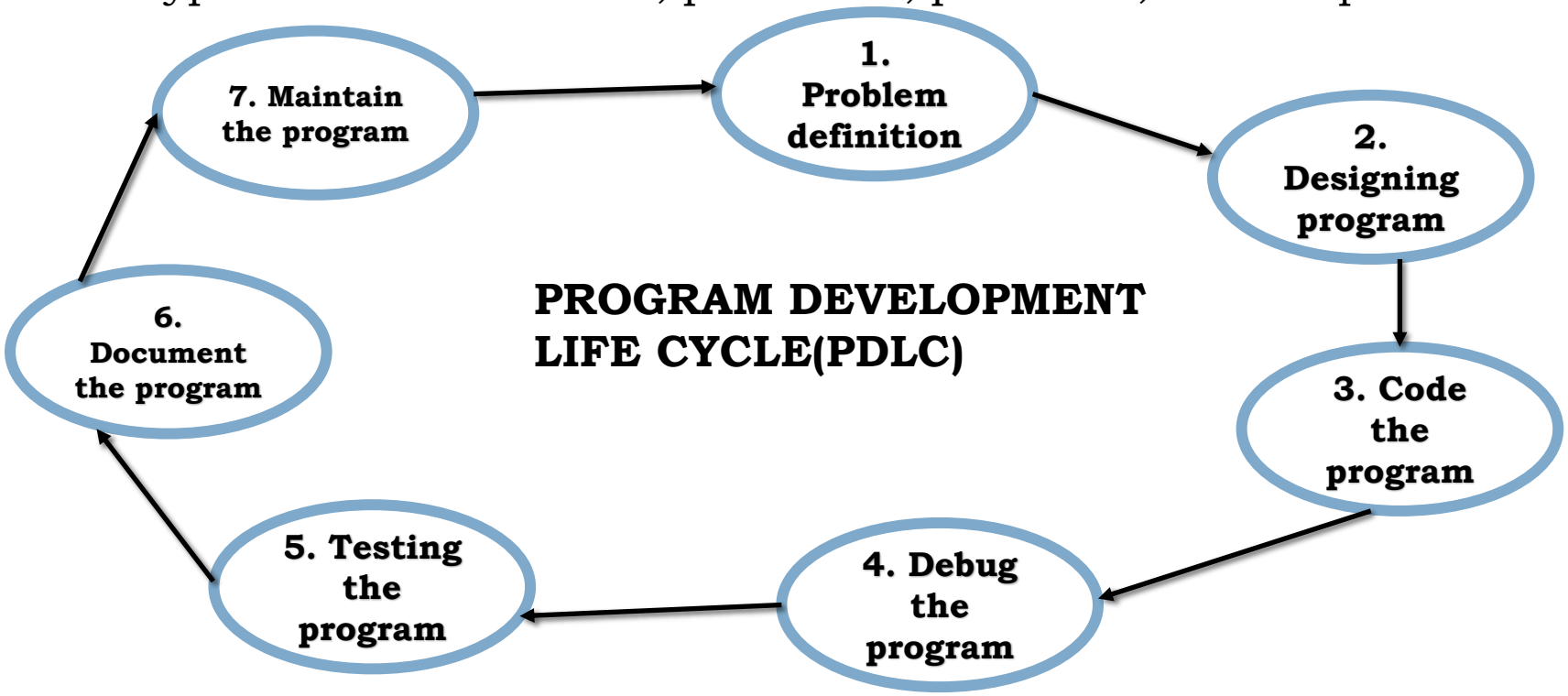
- **Compatibility testing:** It involves checking performance across devices
- **Performance testing:** It involves evaluating system response and behavior under heavy loads.

VI: DOCUMENTATION OF A PROGRAM

- Program documentation involves creating clear and simple written or visual materials to explain the software for end users who often lack programming knowledge.

VII: PROGRAM MAINTENANCE

- Program maintenance involves updating and modifying software after release to fix errors, enhance performance, remove outdated parts and add new features.
- Maintenance types include corrective, preventive, perfective, and adaptive.



CAREERS IN PROGRAMMING

- The following are the common and important programming related careers

i. Software engineer
ii. Software developer
iii. Web developer
iv. Game developer
v. Data Analyst(Data Scientist)
vi. Cybersecurity specialist
vii. System Analysts
viii. Database Administrator

THE FUTURE OF CODING

- As technology continues to evolve rapidly, coding skills are increasing in demand across different sectors.
- Beyond employment, learning to code enhances problem-solving abilities and critical thinking. It empowers individuals to create, automate, and innovate their chosen fields.
- Coding remains a valuable and future-proof skill that supports long-term career stability and success

DEBUGGING

- Bugs or programming errors are common and require debugging to resolve them. The debugging process involves the following:

i. Checking for syntax errors
ii. Running the program step-by-step to identify failures
iii. Using print statements to monitor variable values and execution flow.

DEBUGGING AND ERROR HANDLING

- **Error handling:** Is a technique used to prevent program failure by detecting and properly managing errors during execution.
- Error handling and debugging are essential in program development to ensure code reliability and smooth execution.

COMMON TYPES OF ERRORS

- The following are the common types of errors:

i. **Syntax errors:** Occurs when code violets language rules preventing compilation or execution. Example: Missing brackets, and misspelled keywords.
ii. **Runtime errors:** Occurs during program execution due to operations the system cannot handle. Example: Division by zero or accessing unavailable resources.
iii. **Logic errors:** Logic errors do not cause crashes but produce incorrect results due to flaws in logic or algorithm. Examples include Incorrect formulas or faulty conditional statements.

DEBUGGING TECHNIQUES

- The following are the common debugging techniques, namely

i. Code review
ii. Print and log statements
iii. Debugging tools
iv. Rubber duck debugging
v. Isolating code blocks
vi. Version control
vii. Unit testing

EXCEPTION HANDLING

- **An exception:** Is an unexpected event or error that occurs while a program is running. Examples include dividing by zero, or accessing a file that doesn't exist.
- **Exception handling:** Is a technique used to detect and handle runtime errors (exceptions).
- It allows a program to respond to errors in a controlled way, preventing it from crashing and helping to maintain the normal flow of execution.
- In exception handling, programming languages include special tools like ***try***, ***catch***, **except**, and ***finally***.

DIFFERENCE BETWEEN EXCEPTION HANDLING AND ERROR HANDLING

FEATURE	ERROR HANDLING	EXCEPTION HANDLING
Meaning	General approach used to manage all types of errors	Specific method used to handle run time errors
Scope	Covers all types of errors	Focuses mainly on run time errors only
Technique used	It involves various techniques	It uses built-in language features
Program flow	May stop program execution if not properly managed	Allows the program to continue running smoothly even after an error occurs.

COMMON EXCEPTIONS

i. Division by zero ii. File not found iii. Index out of range
iv. Value error v. Type error vi. Key error
vii. Type conversion error viii. Memory error ix. Import error