UNIT-1 : INTRODUCTION TO PROGRAMMING

1. What is programming?

Programming is a process of giving step by step instructions to a computer so it can perform a specific task.

- **Analogy :** Imagine you have a robot that can make coffee, but knows nothing unless you tell it exactly what to do. Your instructions might be:

  - Fill kettle with 200ml water.
  - Switch on kettle and boil water.
  - Place coffee powder in filter.
  - Pour boiling water slowly through the filter.
  - Pour it into cup.
  - Serve hot.

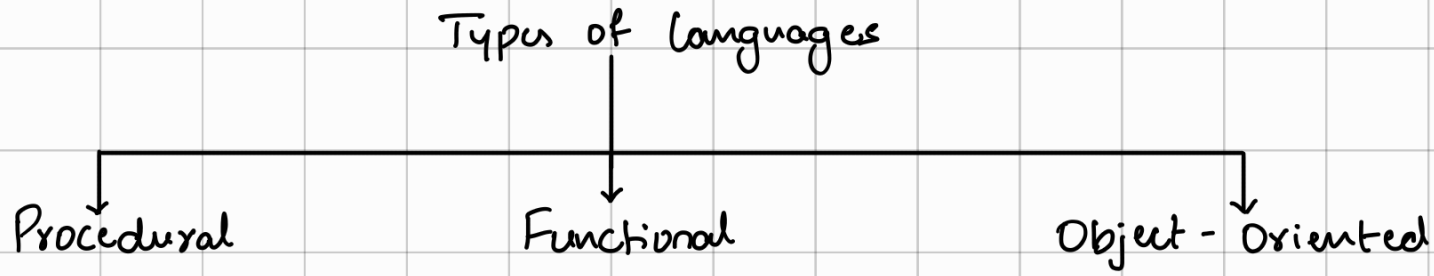
  If you skip any step or mix up the order the coffee won't turn right.

Programming works the same way, you give computer clear ordered steps of instructions in a programming language. If the instructions are wrong, incomplete or out of order then the computer will give wrong results.

2. How computers understand instructions?

Computers don't understand human languages like English, Hindi Kannada etc. Computers only understand binary language (0 & 1). Programming languages help us write instructions in human-readable format, then a compiler or interpreter translates into binary so computers can execute them.

3. Types of Languages



3.1 Procedural

- Follow a step-by-step sequence of instructions to solve problems.
- Example: C, Pascal, Java etc

3.2 Functional

- Writing program only in pure functions
- Used where same operation needs to be performed multiple times.
- Example: Haskell, Java & Python (both support functional style).

3.3 Object Oriented

- Organizes code into objects that bundles data (fields) & behaviour (methods) together
- Core principles (Also known as 4 pillars)
  - Encapsulation (hide details only show essential ones).
  - Inheritance (Inheriting data & behaviour of parent class).
  - Polymorphism (Different behaviour for same action).
  - Abstraction (Hide complexity).
- Example: Java, Python, C++, C#, Ruby

4. Static v/s Dynamic Language

Static Languages	Dynamic Languages
- Must declare type of variable before using it. - Type errors are caught at compile time - Generally faster in execution - Eg: Java, C, C++, Go, Rust	- No need to declare data type explicitly - Type checking happens while running - More flexibility - Eg: Python, JavaScript, Ruby

5. Stack v/s Heap Memory

5.1 Stack Memory

- Stack memory stores method calls, local variables & references
- Allocation happens when method is called.
- Allocated memory is freed when method ends.
- Very fast in terms of speed.
- All primitive data types & its value are directly stored in stack memory.

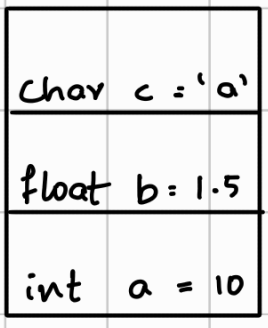
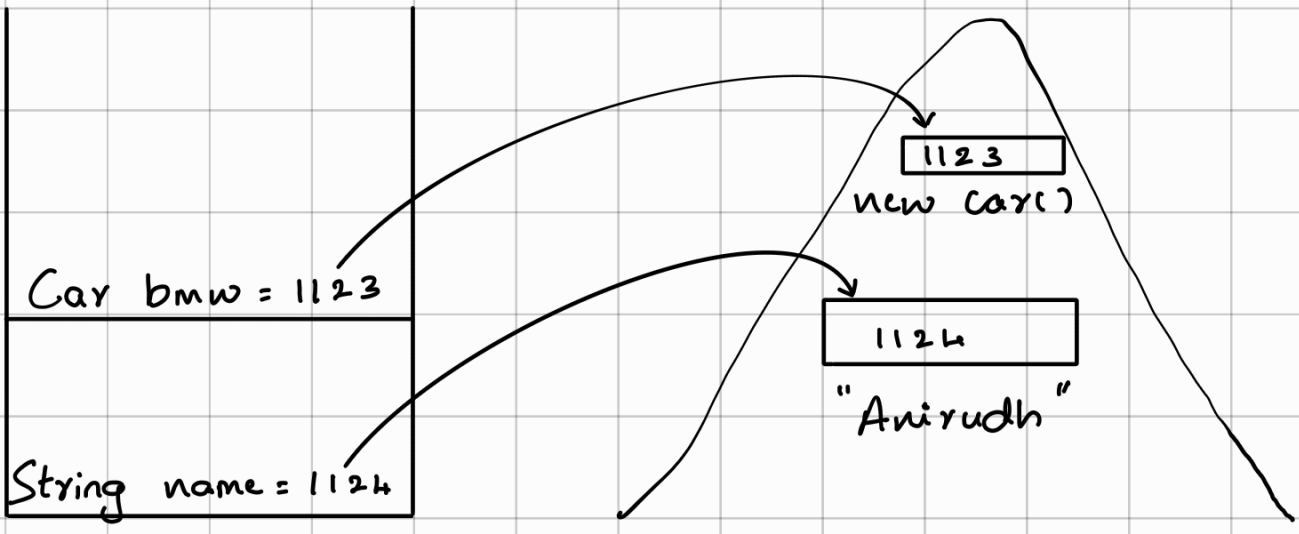


fig: Stack memory representation

5.2 Heap Memory

- Stores objects and instance variables.
- Memory allocation happens dynamically during program execution
- Memory allocation will exist untill garbage collector frees it.
- All Non-primitive data types & Wrapper classes are stored in heap memory.
- Slower in terms of speed when compared with stock memory.



6. Why start with java?

- Java is beginner friendly yet powerful for real-world applications.
- Java is platform independent.
- Java is used to develop Android apps, Web apps, finance systems etc
- Huge community & job demand

7. Key Programming Concepts

A. Variables - Store data

Variables are containers that are used to store & access data.

- Think variables as a labelled jar in your kitchen where you can put something in (storing) and take it out and use it (accessing).

Eg:

```
int age = 25;   // 'age' jar has the number 25
```

B. Data types - What can be stored inside the jar?

Data types define the type of value a variable can hold and operations that can be performed on those values.

- Different jars hold different things.

Eg:
- int → Whole numbers (-99, -50, 0, 1, 7)
- float & double → Decimal numbers (3.14, 2.9)
- Char → A single character or numbers ('A', 'b', '9', '$')
- String → Text ("Hello", "How are you?")
- boolean → true or false

C. Operators - Perform operations on data

Operators are special symbols that perform operations on variables and values. Operators are categorized into

- Arithmetic operators: Used to perform basic mathematical operations like addition (+), subtraction (-), multiplication (*) division (/), modulus (%.)

- Relational (or comparision) operators: Used to compare two values and return a boolean result.
  equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=).

- Logical Operators: Used to combine or modify boolean expressions. These include Logical AND (&&), Logical OR (||), Logical NOT (!).

X	Y	X and Y	X or Y	not(X)	not(Y)
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

- Assignment operators: Used to assign values to variables. Simple assignment operator (=), Compound assignment operators are +=, -=, *=, /=, %=.

- Unary operators: Operate on single operand. They are increment (++), decrement (--), unary plus (+), unary minus (-) & logical Not (!).

- Ternary operator: This is the only operator that takes 3 operands. It is shorthand for "if-else statement".

  Syntax: condition ? expression if True : expression if false;

- Bitwise operators: Used to perform operations on individual bits of integer value. These include bitwise AND (&), bitwise OR (|), bitwise XOR (^), bitwise NOT (~), left shift (<<), Right shift (>>) and Unsigned right shift (>>>).

D. Control Flow - Make decisions & Repeat Actions.

- Conditional statements: Decide what to do based on condition

Eg:

```
if (age >= 18) {
    System.out.println("You can vote");
} else {
    System.out.println("Too young to vote");
}
```

- Loops: Repeat something multiple times.

Eg:

```
for (int i = 0; i < 5; i++) {
    System.out.println("Hello");
}
```

E. Functions - Reusable block of code

Functions can be used again and again without rewriting same block of code.

Eg:

```
int addNumbers(int x, int y) {
    return x + y;
}
```