INTRODUCTION TO PROGRAMMING

Types of languages

Procedural | Functional | Object Oriented

1] PROCEDURAL

- Specifies a series of well structured steps and procedures to compose a program.
- Contains a systematic order of statements, functions and commands to complete a task.
- C, C++, Python, Java.

2] FUNCTIONAL

- Writing a program only in pure functions i.e. never modify variables, but only create new ones as output.
- Used in situations where we have to perform lots of different operations on the same set of data, like ML.
- Follow first-class functions.
- C++, Python.

3] OBJECT ORIENTED

- Revolves around objects.
- Code + Data = Object
- Developed to make it easier to develop, debug, reuse and maintain software.
- C++, Python, Java.

Classes → named group of properties and functions.
[PROPERTY]

Object → An instance of the class.
[Thing actually in the memory]

Compilation → Source code → Machine code

Static vs Dynamic Language

STATIC	DYNAMIC
i) Perform type checking at compile time.	ii) Perform type checking runtime
ii) Errors will show at compile time	ii) Errors might not show till program is run.
iii) Declare datatype before you use it.	iii) No need to declare datatype of variables.
iv) More control	iv) Saves time in writing code but might give error at runtime.

MAJOR DIFFERENCE

`int a = 10`

Variable 'a' can store values of same datatype

//For errors: `int a = "Pri" //Error`

`a=10`

a — 2

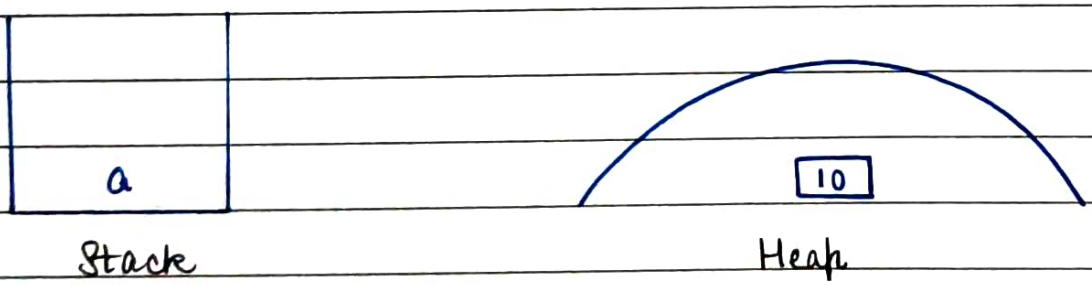
"Pri"

Variable 'a' can take both values i.e. both the datatypes.

`a = 10` } // No Error

`a = "Pri"` }

MEMORY



→ object → heap memory

a = 10

↓

reference variable → stack memory

Variables in stack memory point towards the object in heap memory.

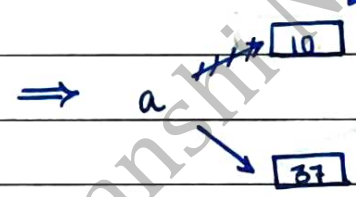
#POINTS TO REMEMBER :

- More than one reference variables can point towards the same object.
- If any one of these reference variables change the object, original object is going to be changed and it's going to be changed for all.

Object with no reference variable

- It will be removed from the memory when garbage collection hits. → It hits automatically.
- Garbage Collection means that objects which do not have a reference variable pointing towards them will be removed.

For eg: a = 10



a = 37

⍟ FLOW OF PROGRAM

#FLOWCHARTS

Flow direction of program →

Start / Stop →

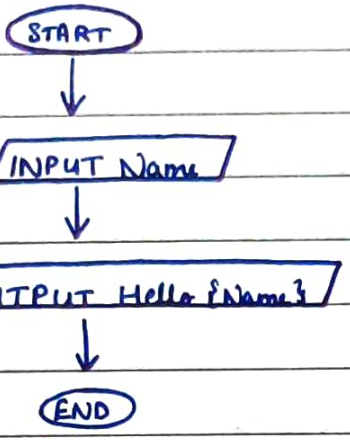
Input / Output →

Processing →

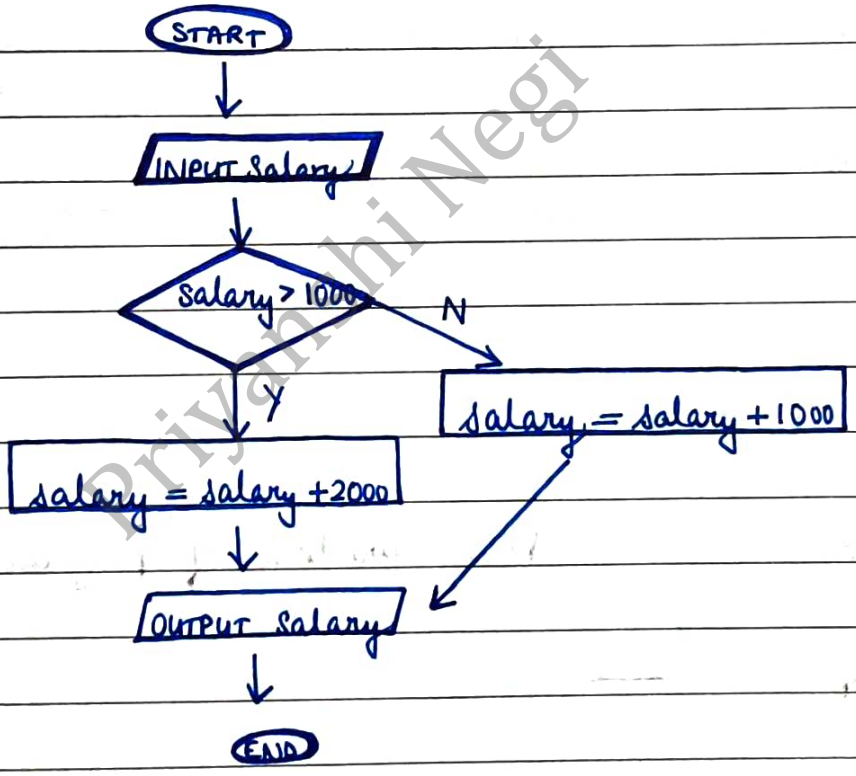
Condition →

Used to visualize our particular thought process or algorithm technique.

Q1. Take a name & output Hello Name



Q2. Take input of a salary. If the salary is greater than 10,000 add bonus as 2000, otherwise add bonus as 1000



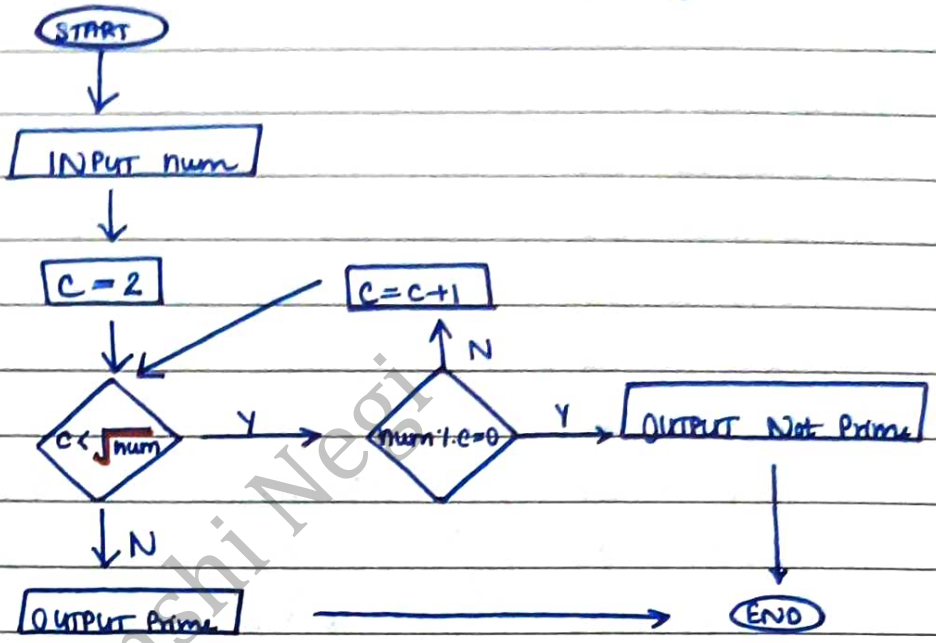
PSEUDOCODE

```
Start
input salary
if salary > 10000:
    salary = salary + 2000
else:
    salary = salary + 1000
output salary
exit
```

%. Stands for REMAINDER

Q3. Input a number and print whether it is prime or not.

if the no. is divisible by 1 & itself only.



PSEUDOCODE

```
Start
input num
c = 2
while   c < num :
      if   num %. c = 0 :
                  output "not prime"
                  exit

      c = c+1
end while
output "prime"
exit
```

PSEUDOCODE

- Rough code.
- Not on any programming language syntax.

num = 36

1 x 36 = 36
2 x 18 = 36
3 x 12 = 36
4 x 9 = 36
$\sqrt{num}$ ← 6 x 6 = 36
9 x 4 = 36
12 x 3 = 36
18 x 2 = 36
36 x 1 = 36

// Repetition is happening
∴ You need not check again for it.
⇒ You can use $c < \sqrt{num}$ instead of $c < num$ as the condition to reduce the complexity of the code.

PSEUDOCODE

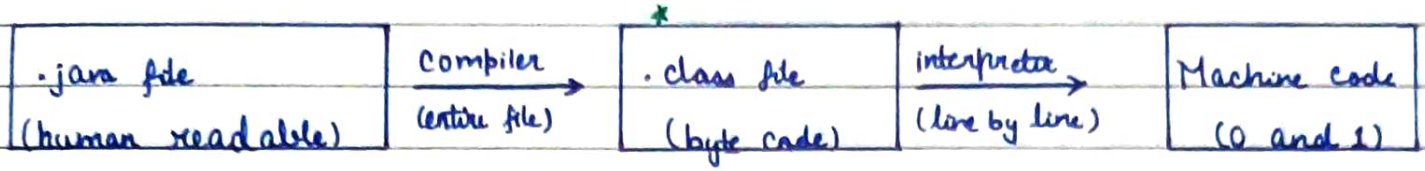
```
Start
input n
if n <= 1:
    print("neither prime nor composite")
    c = 2
    while c * c <= n:
        if n % c == 0
            output ("not prime")
            exit
        c += 1
    end while
    output ("prime")
exit
```

17
int(sqrt(17)) = 4
2, 3, 4 checked

36
sqrt(36) = 6
2, 3, 4, 5, 6 checked

INTRODUCTION

How Java code executes



- this is the source code

- this code will not run directly on a system. → Java Virtual Machine
- we need JVM to run this
- Reason why Java is platform independent

* More about platform independence

- It means that byte code can run on all operating systems.
- We need to convert source code to machine code so computer can understand.
- Compiler helps in doing this by turning it into executable code.
- this executable code is a set of instructions for the compiler.
- After compiling C/C++ code we get .exe file which is platform dependent.
- In Java, we get byte, JVM converts this to machine code.
- Java is platform independent but JVM is platform dependent.
  i.e. Byte code can run any OS.

Architecture of Java

→ Java Development Kit → Java Virtual Machine

JDK vs JRE vs JVM vs JIT → Just-In-Time compiler

↳ Java Runtime Environment

JDK = JRE + Development Tools
(Java Development Kit)

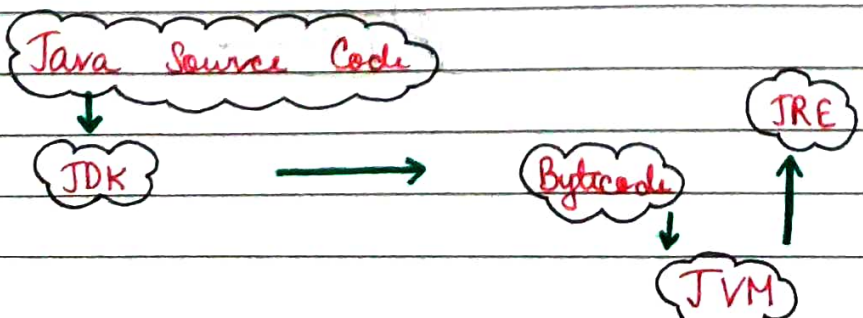
JRE = JVM + Library Classes
(Java Runtime Environment)

Java Virtual Machine (JVM)

JIT
(just-in-time)

JDK

- Provides an environment to develop & run the Java program.
- It is a package that includes:
  1) development tools - to provide an environment to develop your program.
  2) JRE - To execute your program.
  3) A compiler - javac [.java → .class (bytecode)]
  4) Archiver - jar
  5) docs generator - javadoc
  6) interpreter / loader



- JRE → A box (JVM + Library Classes)
- JVM → Actual Content inside the box
  (Does all the work)

JRE

- It is an installation package that provides environment to only run the program.
- It consists of:
  1) Deployment technologies
  2) User interface toolkits
  3) Integration libraries
  4) Base libraries
  5) JVM
- After we get .class file, the next thing happens at runtime:
  1) Class loader loads all classes needed to execute the program.
  2) JVM sends code to Byte code verifier to check the format of cod

Compile time

.java file
↓ javac (compilation)
.class file

Runtime

Class loader
↓
Byte code verifier
↓
Interpreter
↓
Runtime
↓
Hardware

JVM Execution

Interpretor:
- Line by line execution
- When one method is called many times, it will interpret again & again

JIT
- Those methods that are repeated, JIT provides direct machine code so re-interpretation is not required.
- Makes execution faster
- Garbage collector

(How JVM works) Class Loader

- Loading
  - reads .class file & generate binary data (object of a class)
  - an object of this class is created in heap.
- Linking
  - JVM verifies the .class file
  - allocates memory for class variables & default values
  - replace symbolic references from the type with direct references
- Initialization
  all static variables are assigned with their values defined in the code & static block
  (do not depend on the object of the classes)
  eg: Human class
  Object → Population variable (Static variable)

* JVM contains the Stack & Heap memory allocations.

object independent