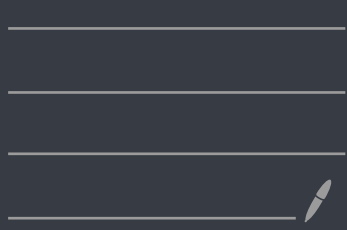
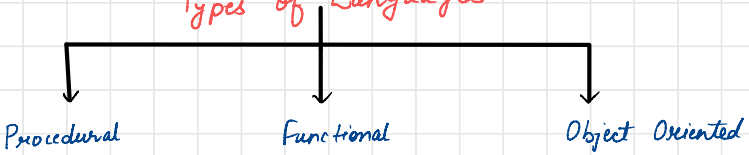
JAVA + DSA



Types of Languages



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.

Functional:-

→ Writing a program only in pure functions i.e. never modify variables but only create new ones as an output.
→ Used in situations where we have to perform lots of different operations on the same set of data, like ML.
→ First class Functions?

Object Oriented:-

→ Revolves around objects.
→ Code + Data = Object
→ Developed to make it easier to develop, debug, reuse & maintain software.

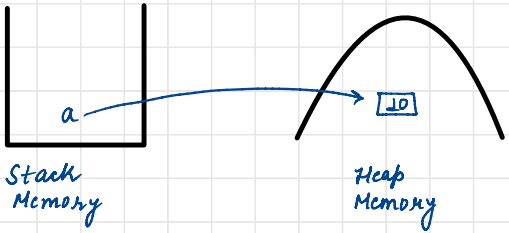
Static Vs Dynamic Languages

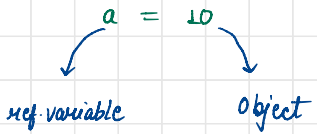
Static

→ perform type checking at compile time
→ Errors will show at compile time
→ Declare datatype before you use it
→ More Control

Dynamic

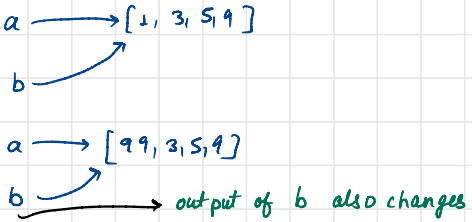
→ Perform type checking at runtime. runtime.
→ Error might not show till program is run
→ No need to declare datatype of variables
→ Saves time in writing code but might give error at runtime



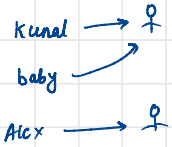


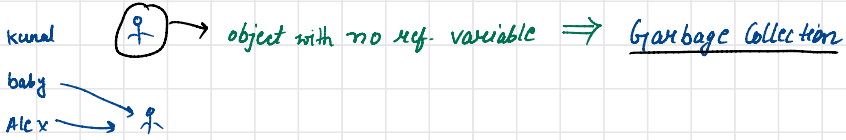
Important Example Memory

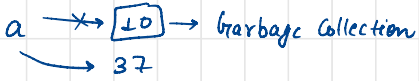
```
a = [1,3,5,9]
b = a
a[0] = 99
```



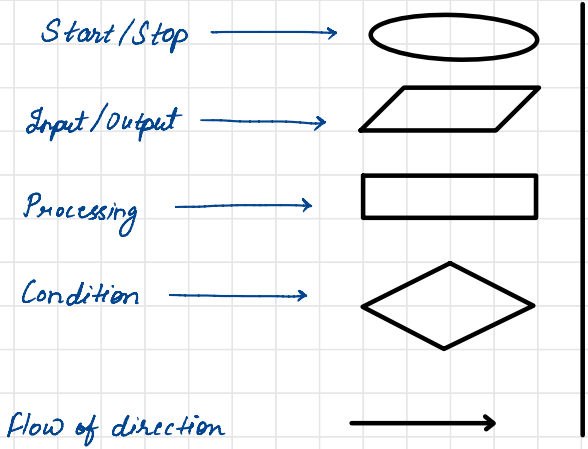
Now,



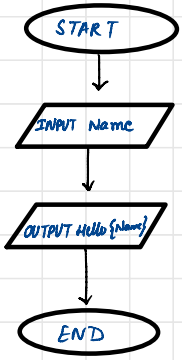




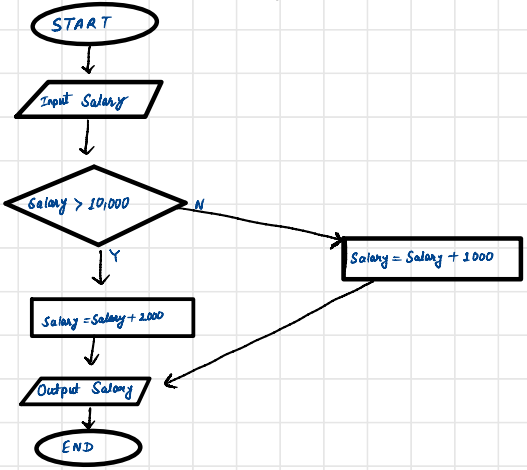
Flow of Program



eg:- take a name and output Hello Name



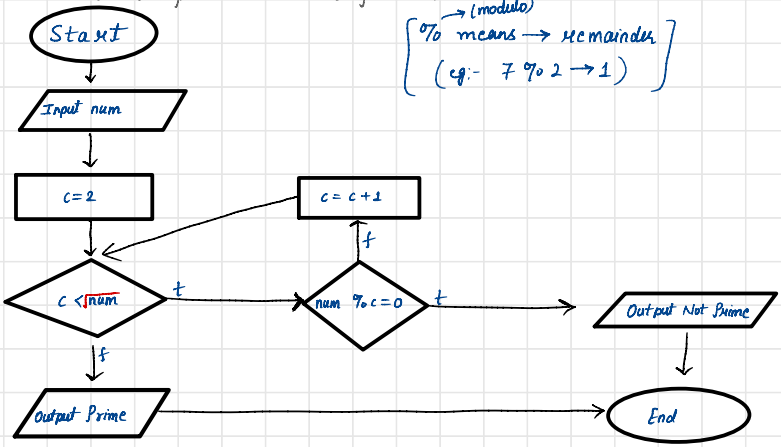
eg:- Take input of a salary. If the salary if greater than 10,000 add bonus as 2000, otherwise add bonus as 1000.



Psuedo Code

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

eg:- Input a number and print whether it is prime or not.



% → (modulo)
% means → remainder
(eg:- 7 % 2 → 1)

Pseudo Code

```
Start
input num
c = 2
while c < num:
    if num % c = 0
        output "not prime"
        exit
    c = c+1
end while
output "prime"
exit
```

1 x 36 = 36
2 x 18 = 36
3 x 12 = 36
4 x 9 = 36
6 x 6 = 36 } ⇒ repeated
~~9 x 4 = 36~~
~~12 x 3 = 36~~
~~18 x 2 = 36~~
~~36 x 1 = 36~~

(23456786543)^1/2
153156
possibilities reduced

Pseudo Code

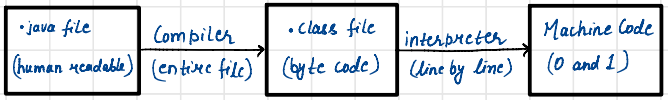
```
Start
input n
if n<=1:
    print("neither prime nor composite")
c = 2
while c*c <=n:
    if n % c ==0
        output "not prime"
        exit
    c += 1
end while
output "prime"
exit
```

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

17
int(sqrt(17)) = 4
2, 3, 4, 5, 6

Introduction to Java

* How Java code executes?



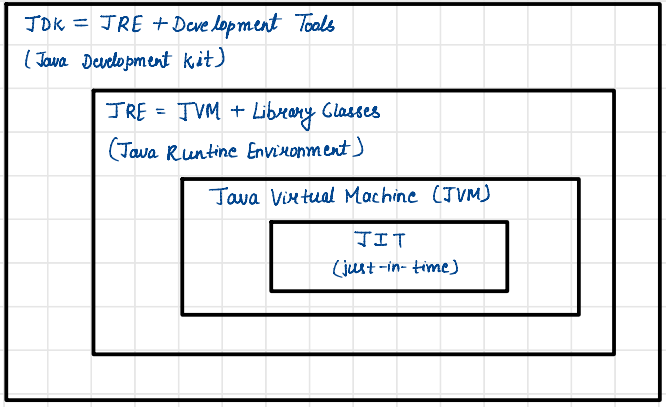
this is the source code

→ this code will not directly run on a system
→ 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 → (code that you can run that will give you output)
→ this executable code is a set of instructions for the computer
→ After compiling C/C++ code, we get .exe file which is a platform dependent
→ In Java we get bytecode, JVM converts this into machine code
→ Java is platform-independent but JVM is platform dependent.

* JAVA vs JRE vs JVM vs JIT



* JDK

→ Provides environment to develop and 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
4. archiver – jar
5. docs generator – javadoc
6. interpreter / loader

* JRE

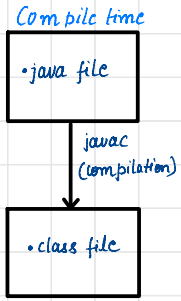
→ It is an installation package that provides environment to only run the program

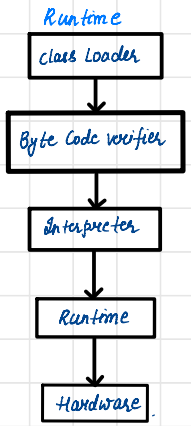
→ It consists of:-

1. Deployment technologies
2. User interface toolkits
3. Integration libraries
4. Base libraries
5. JVM

→ After we get the .class file, the next thing happen 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 code





* (How JVM works) Class Loader

→ Loading:

- reads .class file and generate binary data
- 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 and static block

- JVM contains the Stack and Heap memory allocations.

* JVM Execution

→ Interpreter:

- Line by line execution
- when one method is called many times, it will interpret again and again

* JIT:

- those methods that are repeated, JIT provides direct machine code so re-interpretation is not required.
- makes execution faster

Garbage Collector