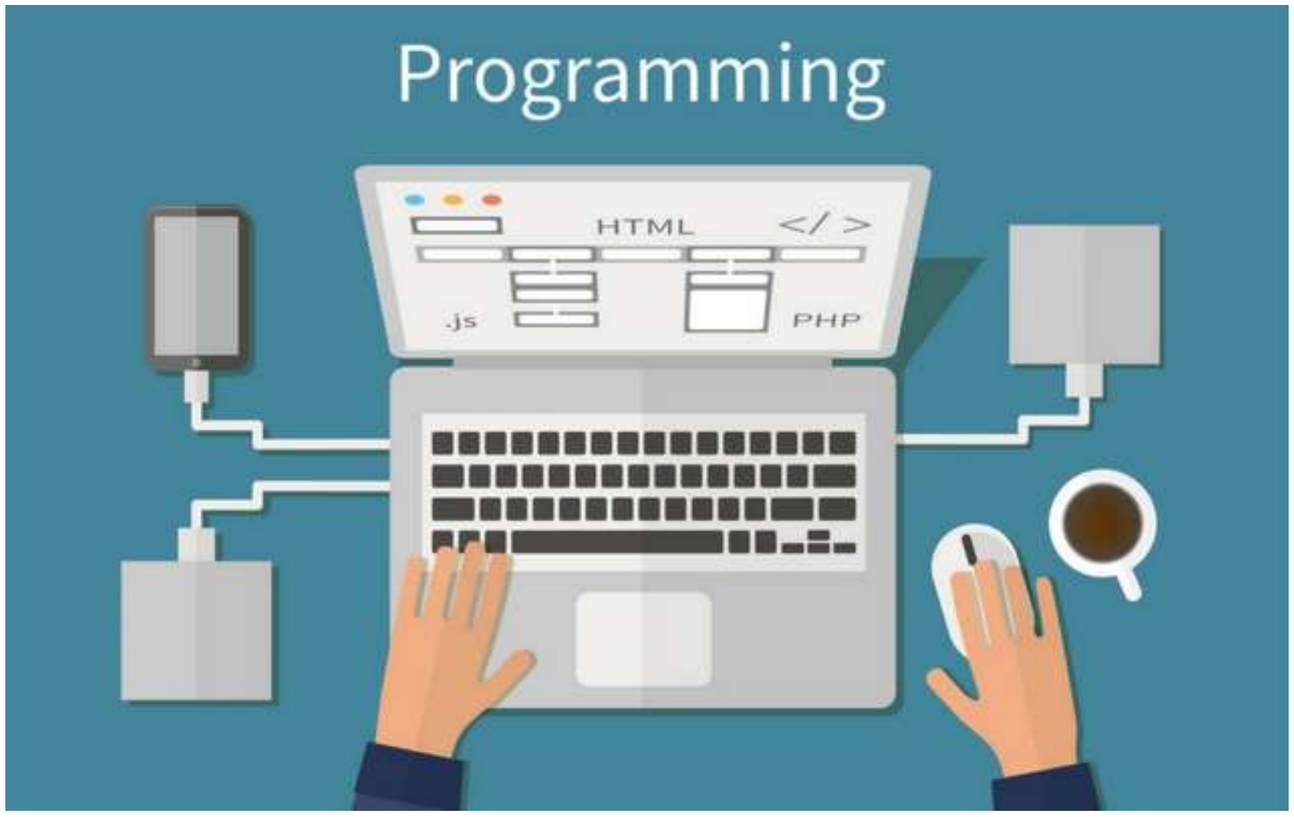
INTRODUCTION TO PROGRAMMING

Sourav Karmakar

souravkarmakar29@gmail.com

What is Programming?



Programming:

Attempting to get a computer to complete a *specific task* without making *mistakes*.

What is Programming?

Programming:

Attempting to get a computer to complete a ***specific task*** without making ***mistakes***.

- There has to be clear cut instructions to complete a specific task.
- That set of instructions are known as program (or code)
- Apart from the instructions, one also needs **input data** for a program to run correctly and to produce desired result.

Mistakes in a program can happen in various way:

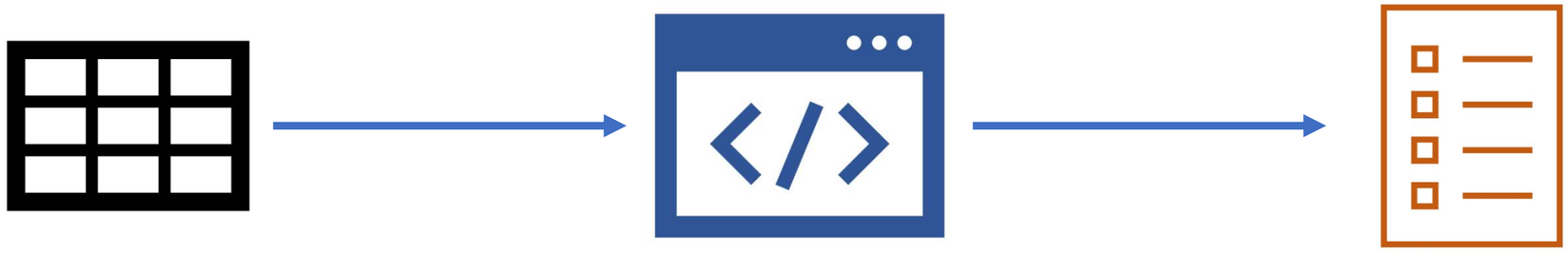
- Mistakes in instructions
- Mistakes in Input Data

Mostly all kind of mistakes / errors in a program are called **bugs** in programming world.

What is Programming?

An Example

A program to take input of the marks of all the students of a particular class of a school and give the name and roll number of rankers of different subjects.



Input Data

Marks of all the students of the class

Instructions

The code to process the data and do the desired task

Output

The list of the ranker students of different subject

What instructions do a machine understand?

- Machine (i.e. a Computer) can only understand a machine language.
- As the computers are electronic devices it only can understand 1 and 0 (binary numbers)
- Hence the machine language which the computer can understand comprises of only 1s and 0s

Now it is incredibly hard for a human being to write a code in machine language and to make it error-free.

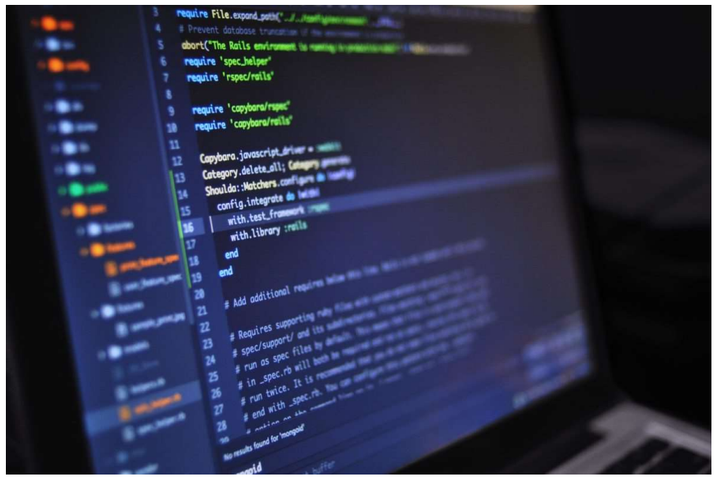
So what is the solution?

Answer : The Programming Languages

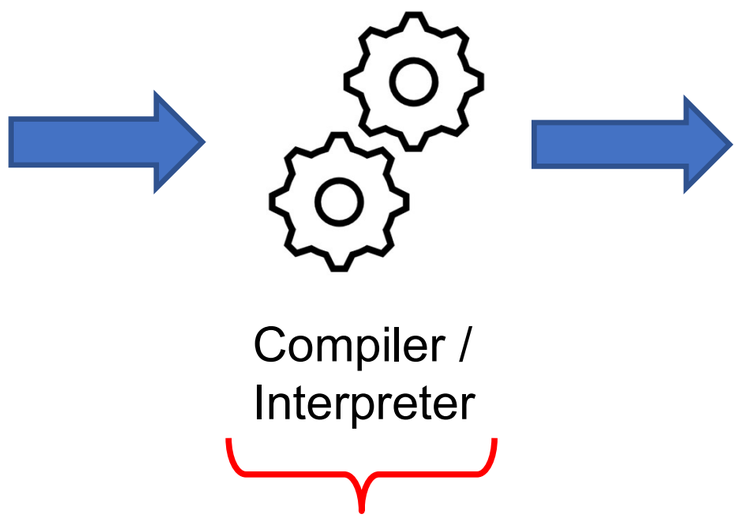
The Programming Languages

The programming languages are user friendly computer language that programmers use to write codes.

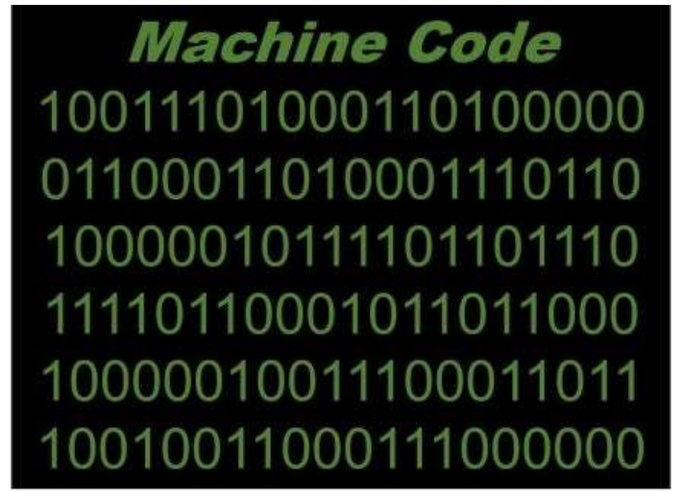
Note that computer doesn't comprehend programming languages as well. There is an intermediate process which converts the programming language into machine readable format (i.e. machine code)



Code in Programming Language



Compiler / Interpreter



Machine Code

It is a program which converts instructions written in programming language into machine code

Difference Between Compiler and Interpreter?

Compiler

A compiler is computer software that readily translates *high level* programming languages into machine code. It also checks for syntactical errors in the program.

Note that both compiler and interpreter are language specific.

Interpreter

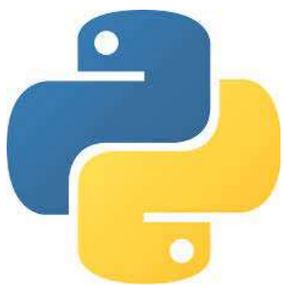
An interpreter is a computer program that converts program statements line by line into machine code. It also checks for syntactical error in program statement.

Difference Between Compiler and Interpreter

Basis	Compiler	Interpreter
Analysis	The entire program is analysed in a compiler.	Line by Line of the program is analysed in an interpreter.
Machine Code	Stores machine code in the disk storage.	Machine code is not stored anywhere.
Execution	The execution of the program happens only after the entire program is compiled. Errors are shown at the end of compilation.	The execution of the program takes place after every line is evaluated and hence the error is raised line by line.
Run Time	Compiled program usually runs faster.	Interpreted program runs slower.
Generation	The compilation gives an output program called Object Code that runs independently from the source file.	The interpreter does not give any output program and is thus evaluated on every execution.
Programming Languages	C, C++, C#, JAVA are compiler based programming languages.	PHP, PERL, Python are interpreter based programming languages.

Programming Languages

Python



There are bunch of programming languages available.

Now the question is *why there are so many Programming Languages?*

The purpose of different programming languages are different.

For example:

- R, Juila, Matlab are used for scientific computations, where R is specifically used for statistical computations.
- C, C++, JAVA, Python are General Purpose Programming Language
- HTML, CSS, Javascript are used for building websites.

What is an Algorithm?

Before jumping into learning the programming language of our choice let's first understand what is an **algorithm**.

An **Algorithm** is defined as a sequence of *well defined steps* which aims to provide a solution to a *specific problem* using computer.

An algorithm is problem specific, however a problem can have multiple algorithms.

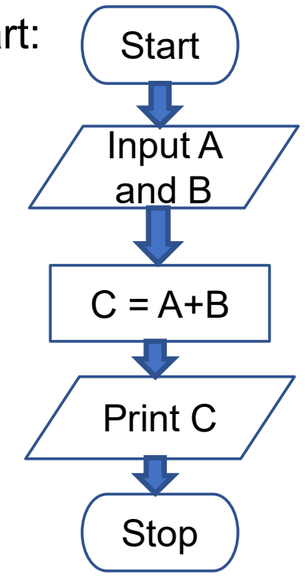
Algorithms are represented using *Natural Languages* like English or using *Flowcharts* or using *Pseudo-Codes.*

Example: Take input of two numbers and print their sum.

In plain English:

1. Take the Input of First Number and store it in A
2. Take the Input of Second Number and store it in B
3. Compute A+B and store it in C
4. Print the value of C

Flowchart:



In Pseudo-Code:

```
Begin
   A = Input("Enter first number: ")
   B = Input("Enter second number: ")
   C = A+B
   Print (C)
End
```

Flowchart

Flowchart is the pictorial representation of the step by step procedure (or algorithm)

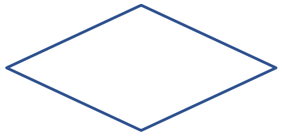
There are multiple shapes used to create a Flowchart. Each shape defines a specific purpose.

Start / Stop

Input / Output

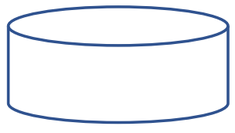
Processing

Decision



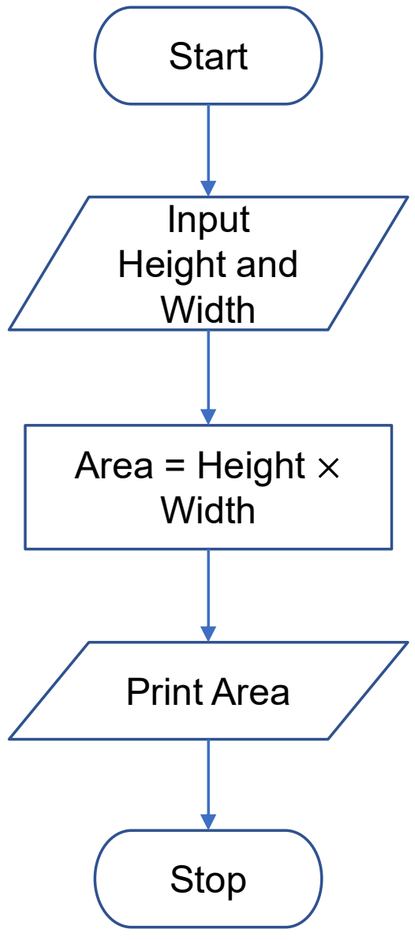
Flow Arrow

Storage (Data)



Documents (File)

Example: Take input of Height and Width of a rectangle and print Area



Pseudo-Code

- Though Flowcharts are great visual tool to describe an Algorithm, they become very complicated when the algorithm is lengthy.
- Complicated programs are often modularized into functions or sub-routines, each function doing a specific task. Describing those functions using Flowchart is cumbersome.

Pseudo-Codes are written in the form of a high level programming language (like C or Python) and provides an excellent way to write algorithm without writing an actual program (hence the name pseudo-code).

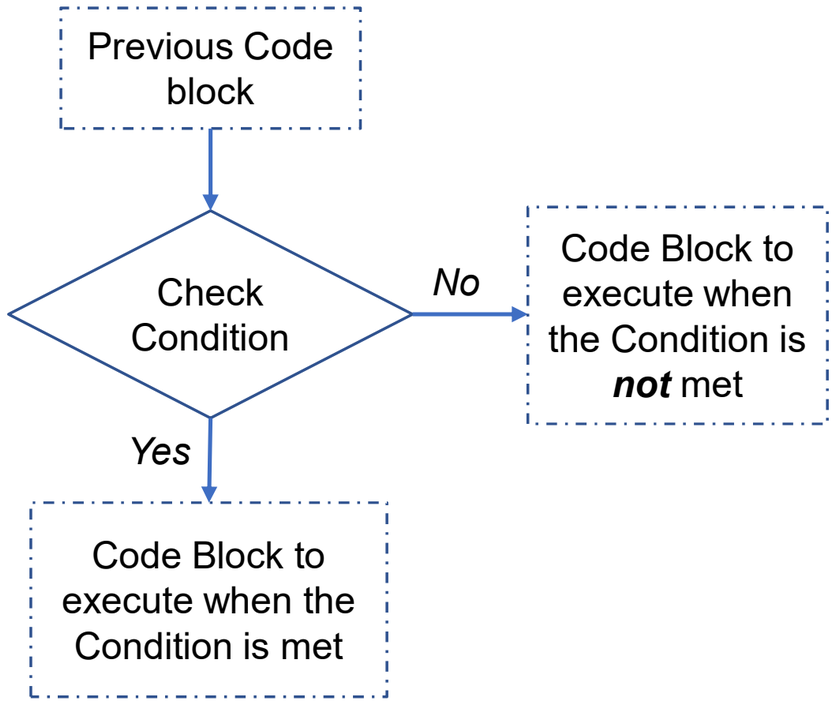
Though a Pseudo-code structurally resembles a high level programming language, they may not be a working code because we don't exactly follow all the syntactical details of a programming language while writing pseudo-code.

Example: Take input of Height and Width of a rectangle and print Area.

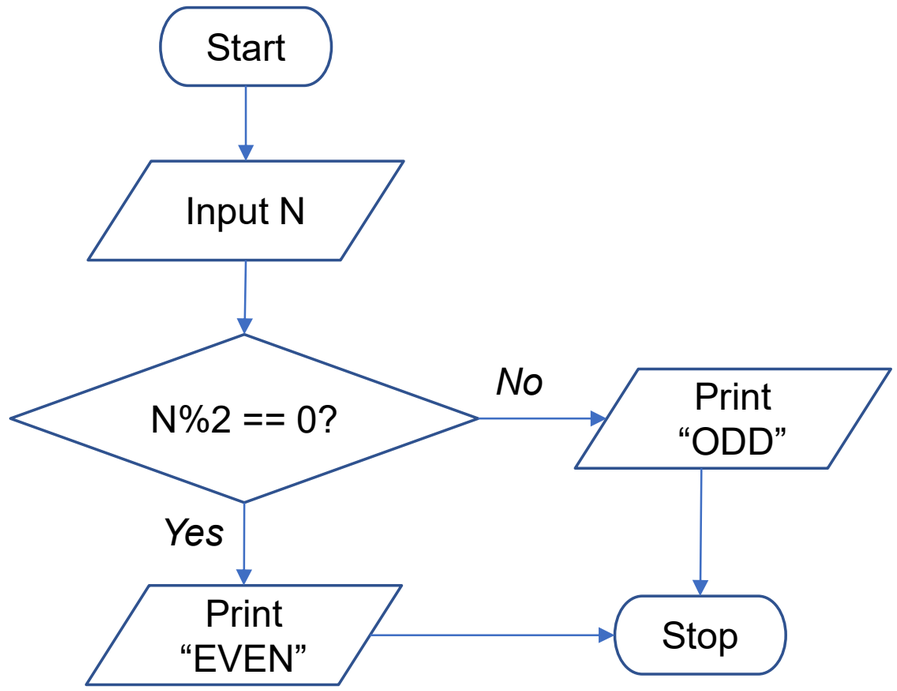
```
BEGIN
    READ Height, Width
    Area = Height × Width
    PRINT (Area)
END
```

Decision Making

In Flowchart



Example: Given a number check whether it is even or Odd



In Pseudo-Code

```
IF (Condition):
        Codes to execute when the Condition is met
ELSE:
        Codes to execute when the Condition is not met
END IF
```

```
BEGIN
        READ N
        IF N % 2 == 0:
                PRINT “Even”
        ELSE:
                PRINT “Odd”
        END IF
END
```

Decision Making: Multiple Conditions

Each condition returns a Boolean Value. It is **TRUE** if the condition is met else it is **FALSE**.

Multiple Conditions can be suitably combined using AND / OR clause.

AND Conditions

The usual form of AND condition is following:

Condition1 AND *Condition2* AND *Condition3* AND …

When all the conditions are TRUE then only the expression is TRUE.

Example:

$$0 \leq Age < 18$$

Age is ≥ 0 AND Age is < 18

There can be combination of AND / OR in a clause.

OR Conditions

The usual form of OR condition is following:

Condition1 OR *Condition2* OR *Condition3* OR …

If any one of the conditions is TRUE then the whole expression is TRUE.

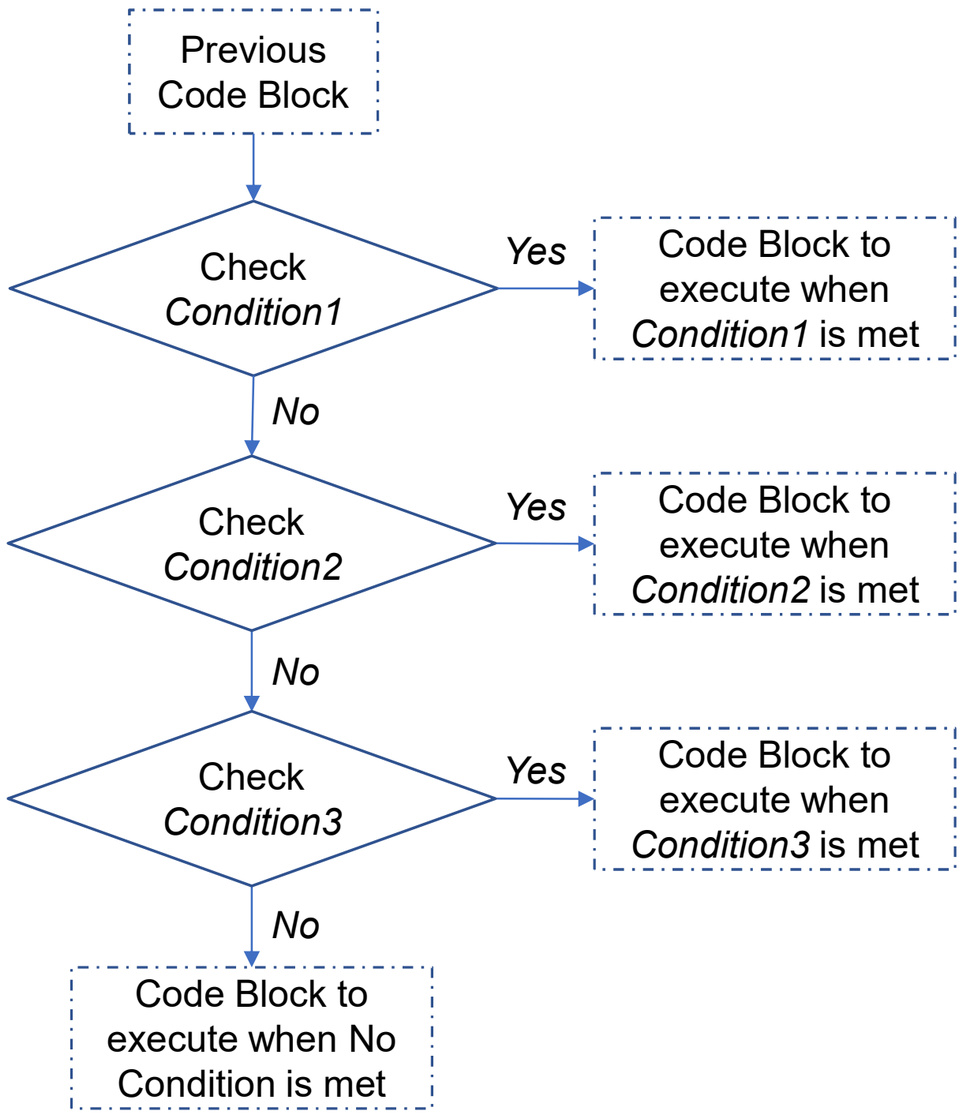
Example:

Vowel if the Letter is A, E, I, O, U

Letter == ‘A’ OR Letter == ‘E’ OR Letter == ‘I’
OR Letter == ‘O’ OR Letter == ‘U’

Decision Making: Multiple Decisions

In Flowchart



In Pseudo-Code

```
IF (Condition-1):
        Codes to execute when Condition-1 is met
ELSE IF (Condition-2):
                Codes to execute when Condition-2 is met
        ELSE IF (Condition-3):
                        Codes to execute when Condition-3 is met
                ELSE:
                        Codes to execute when no conditions met
                END IF
        END IF
END IF
```

This is also called **nested** IF-ELSE blocks.

Decision Making: An Example

Write a Pseudo-Code to determine the age group of a person given the person's age. The age group is described as following:

$0 \leq Age < 18$: Not Adult
$18 \leq Age < 30$: Young Adult
$30 \leq Age < 45$: Mid Adult
$45 \leq Age < 60$: Middle Aged
$60 \leq Age$: Senior Citizen

The Pseudo-code looks like following:

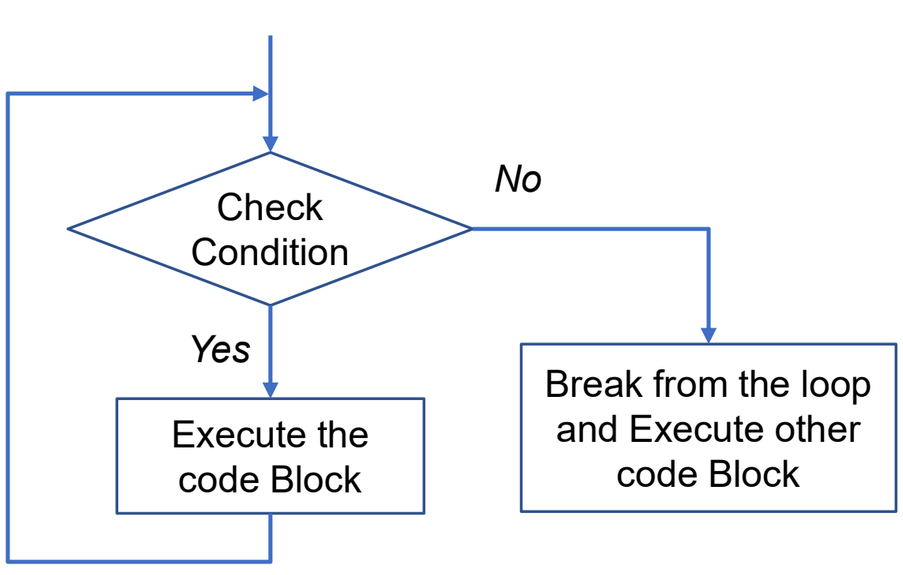
```
BEGIN
    IF Age ≥ 0 AND Age < 18 :
        PRINT "Not Adult"
    ELSE IF Age ≥ 18 AND Age < 30 :
        PRINT "Young Adult"
    ELSE IF Age ≥ 30 AND Age < 45 :
        PRINT "Mid Adult"
    ELSE IF Age ≥ 45 AND Age < 60:
        PRINT "Middle Aged"
    ELSE:
        PRINT "Senior Citizen"
    END IF
END IF
END IF
END IF
END
```

Looping

Looping is fundamentally very important concept of Programming. It comes very handy when we have to perform repetitive tasks.

There are usually two types of Looping that programmers use. **While Loop** and **For Loop**.

While Loop executes a code block if the condition is met. After execution of the code block the condition is again checked, if that's still true it will execute the code block under it else it will break from the loop.

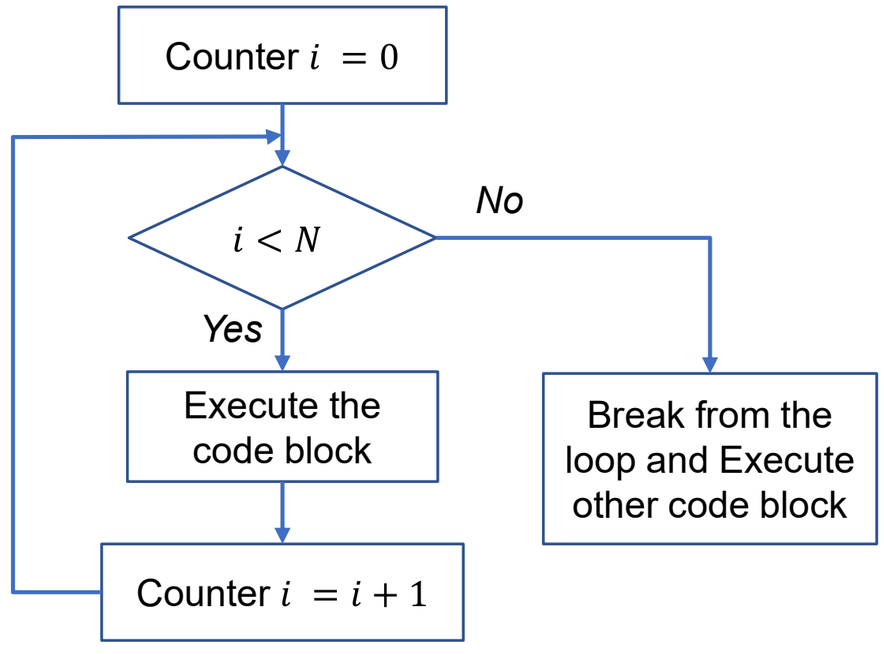


WHILE (Condition):

Execute the Code Block

END WHILE

For Loop makes use of a counter and executes the code block under it for predefined number of times.



FOR (i = 0 ; i < N ; i ++) :

Execute the Code Block

END FOR

FOR i FROM 0 to N-1:

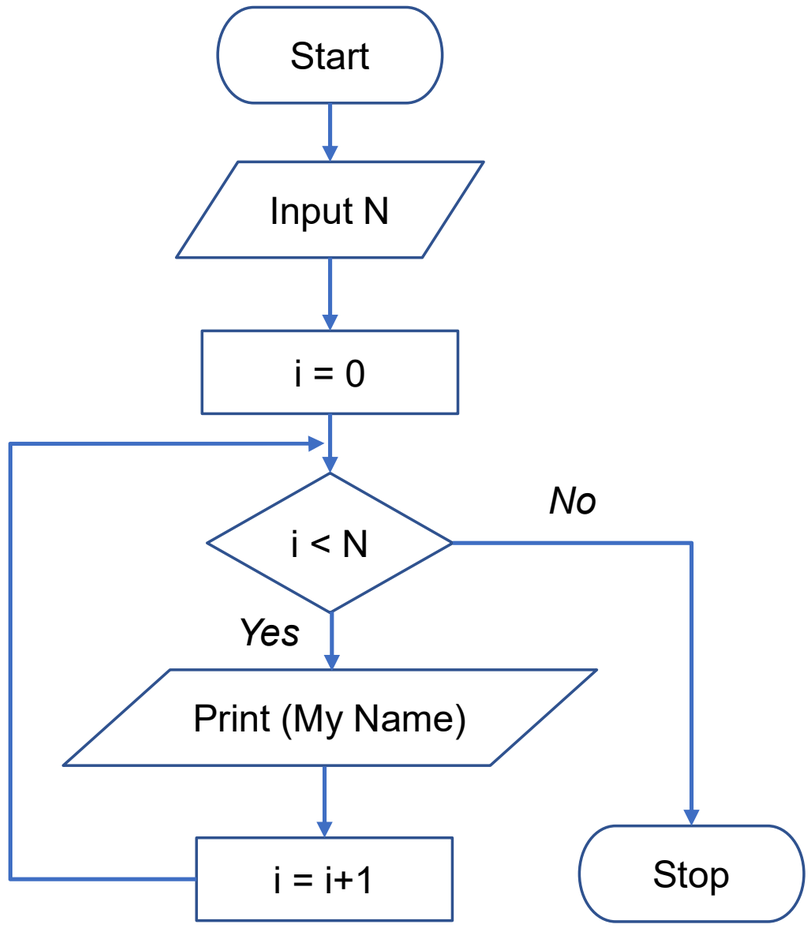
Execute the Code Block

END FOR

Looping

Let’s take a very Lucid example:

Print your name for given number of times.




```
BEGIN
        FOR i FROM 0 TO N-1:
                PRINT (My Name)
        END FOR
END
```

```
BEGIN
        i = 0
        WHILE (i < N):
                PRINT (My Name)
                i = i + 1
         END WHILE
END
```

Loop Control Statements

BREAK Statement

Sometimes we can use BREAK statement to break out of the loop if certain condition is met.

Consider the Example: Write a Program that allows user to input integer until he/she enters a negative number

```
BEGIN
    WHILE (TRUE):
        READ N
        IF (N < 0):
            BREAK
        END IF
    END WHILE
END
```

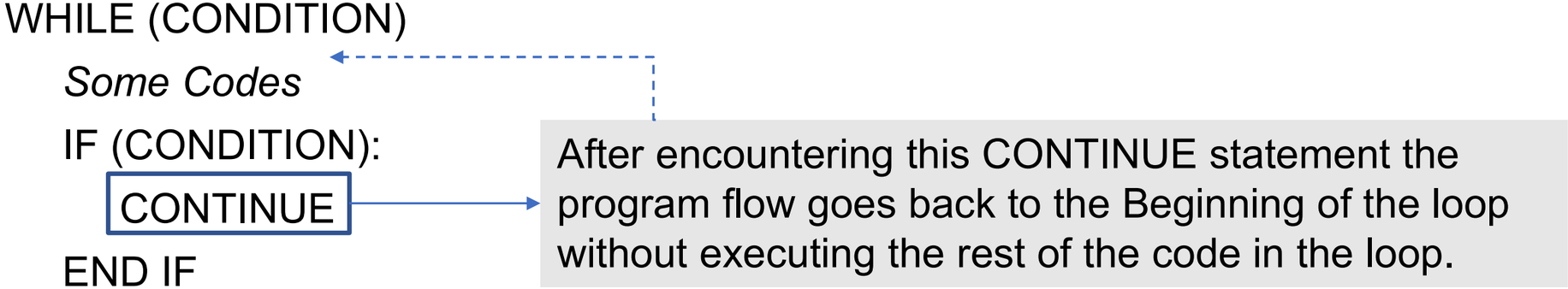
WHILE (TRUE): The condition inside the While statement has kept TRUE. This must be accompanied with a **BREAK** statement, else it will continue to run. This is also called Infinite loop.

After executing the BREAK statement the program will break from the loop

Loop Control Statements

CONTINUE Statement

The basic structure of CONTINUE statement is described in the following flowchart.



Example: Print all the odd numbers between 0 to 20

```
BEGIN
    FOR i FROM 0 TO 20:
        IF i % 2 == 0:
            CONTINUE
        ELSE:
            PRINT i
        END IF
    END FOR
END
```

Looping

Let's take a another example: Given a number check whether it is Prime number or not. *(A number is a Prime number if it is divisible only by 1 and that number)*

Intuition: Given the number N, we have to check all the numbers till N/2 whether any of them divides N.

```
BEGIN
    READ N
    i = 2
    Flag = 0
    WHILE (i ≤ N/2) :
        IF (N % i == 0) :
            PRINT "Not Prime"
            Flag = 1
            BREAK
        ELSE :
            i = i + 1
        END IF
    END WHILE
    IF Flag == 0:
        PRINT "Prime"
    END IF
END
```

Here Flag is a variable whose status gives the indication whether the number is prime or not.

First we have initialized Flag with 0

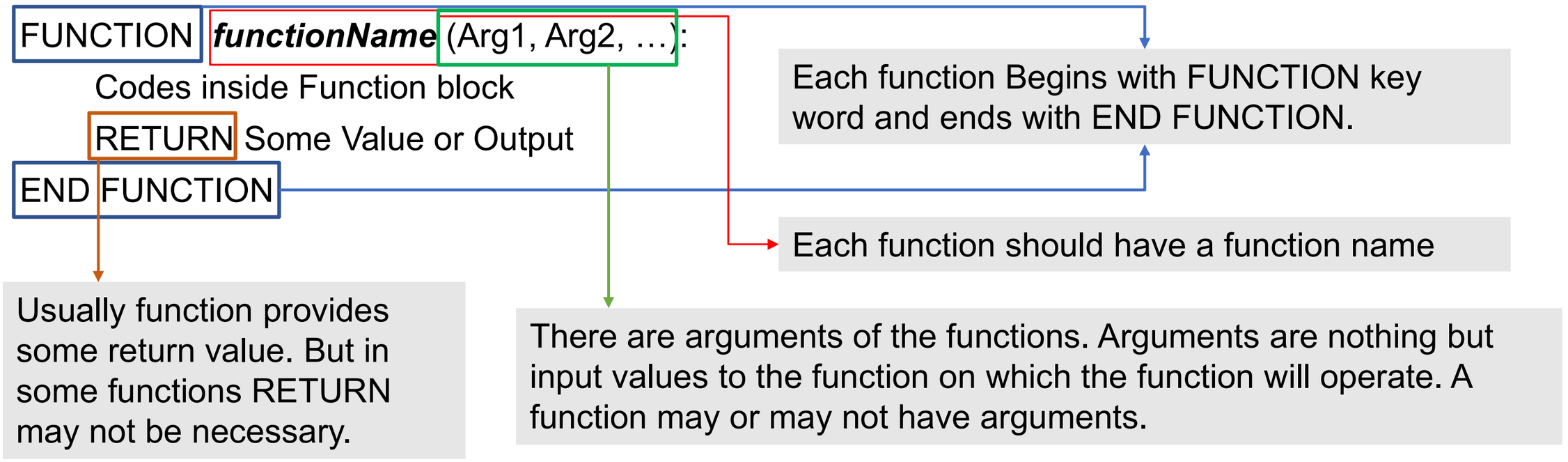
It is set to 1 if the number is not prime

It remains 0 if the number is prime

Finally we check the condition of the Flag to check if the number is Prime

Functions

Complicated Programs are often modularized into functions. Now lets see how this function works.



Example: Calculate Sum of Two Numbers

Option-1

```
FUNCTION CalculateSum (Num1 , Num2):
    Result = Num1 + Num2
    RETURN Result
END FUNCTION
```

Option-2

```
FUNCTION CalculateSum ():
    READ Num1, Num2
    Result = Num1 + Num2
    RETURN Result
END FUNCTION
```

Option-3

```
FUNCTION CalculateSum ():
    READ Num1, Num2
    Result = Num1 + Num2
    PRINT Result
END FUNCTION
```

Functions

Print all the prime numbers between 2 to 100

```
FUNCTION CheckPrime(N):
    i = 2
    WHILE (i ≤ N/2):
        IF (N % i == 0):
            RETURN FALSE
        END IF
    END WHILE
    RETURN TRUE
END FUNCTION
```

When the function *CheckPrime(k)* is called within the main program, the function executes with value N = k.

When this RETURN statement is executed the flow of the program goes back to the main program with RETURN value FALSE

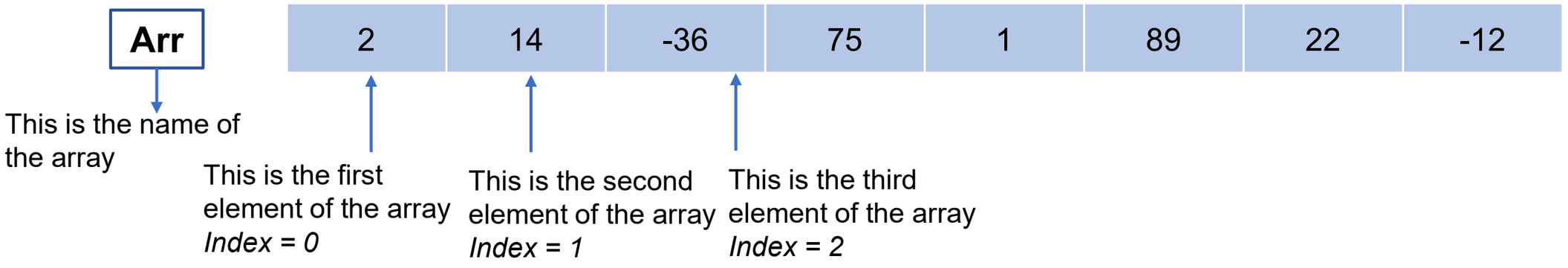
After the complete execution of the WHILE loop if we found no number that divides N, we return TRUE (indicating that the number is Prime)

```
BEGIN
    FOR k FROM 2 TO 100:
        p = CheckPrime(k)
        IF (p == TRUE):
            PRINT(k)
        END IF
    END FOR
END
```

The main program calls the function *CheckPrime()*

Arrays

Arrays are series of numbers stored consecutively.
These numbers are referred using the ***name of the array*** and ***index*** (the position where the number is stored)



In the languages like C, python, Java the array indexing starts from 0. There are programming languages like FORTRAN where it starts from 1. We will consider 0 based indexing throughout our course.

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Arr	2	14	-36	75	1	89	22	-12
	Arr[0]	Arr[1]	Arr[2]	Arr[3]	Arr[4]	Arr[5]	Arr[6]	Arr[7]

If we give index more than 7 then it will result in an error.

Arrays

2	14	-36	75	1	89	22	-12

Length of the Array

- It is the total number of elements an array contains. In the example above there are total 8 elements in the array and the length of the array is 8.
- In Pseudo Code we write length of the array as **LEN(Arr)**
- The index of the array runs from 0 to LEN(Arr)-1

Example: Write a program to print all the values of an array

```
FUNCTION PrintAllValues (Arr):
        FOR i FROM 0 TO LEN(Arr)-1:
                PRINT (Arr[i])
        END FOR
END FUNCTION
```

Arrays

Example: Write a program to find the max value in an array

```
FUNCTION FindMax (Arr):
    MAX = Arr[0]
    FOR i FROM 1 to LEN(Arr)-1:
        IF (MAX < Arr[i]):
            MAX = Arr[i]
        END IF
    END FOR
    RETURN MAX
END FUNCTION
```

Example: Write a program to check whether a value exist in an array. If the value exist return the index of the array else return -1.

```
FUNCTION SearchValue (Arr, value):
    FOR i FROM 0 TO LEN(Arr)-1:
        IF (Arr[i] == value):
            RETURN i
        END IF
    END FOR
    RETURN -1
END FUNCTION
```

Thank You