Data Structures Using Python

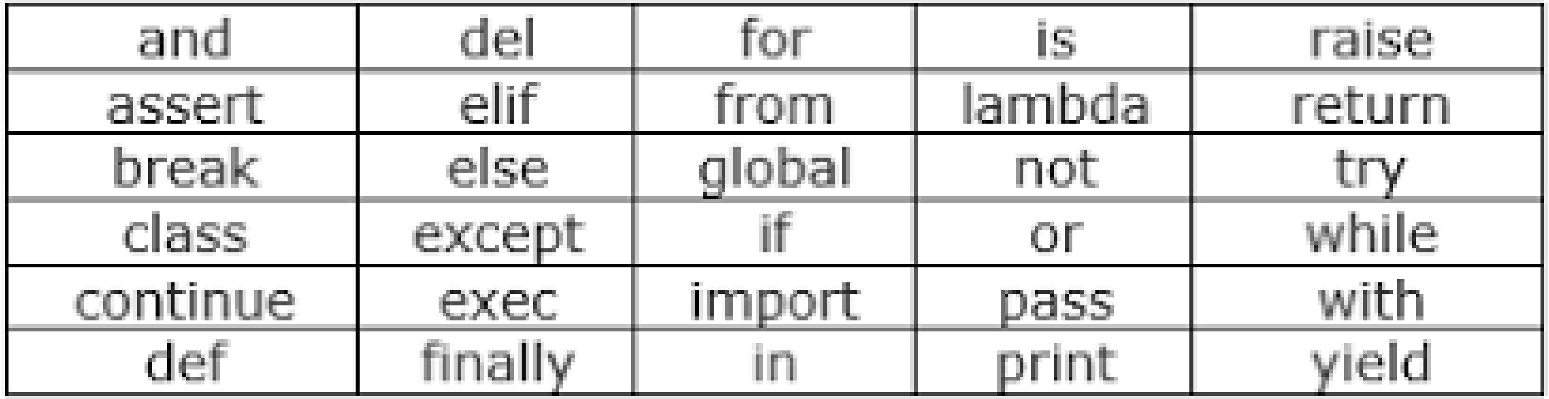
Makkapati Bhavana

{Y20EC117}

Python

- **Python is a high-level Programming language like C, C++, Perl, and Java.**
- **Computers can execute programs written only in low-level languages.**
- **So programs written in a high-level language have to be processed before they can run. This extra processing takes some time, which is a small disadvantage of high-level languages.**
- **The advantages of high-level language are enormous. First, it is much easier to program in a high-level language as they are shorter and easier to read, and they are more likely to be correct.**
- **Second, high-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications.**
- **Python is considered as an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter: interactive mode and script mode.**

Key Words:

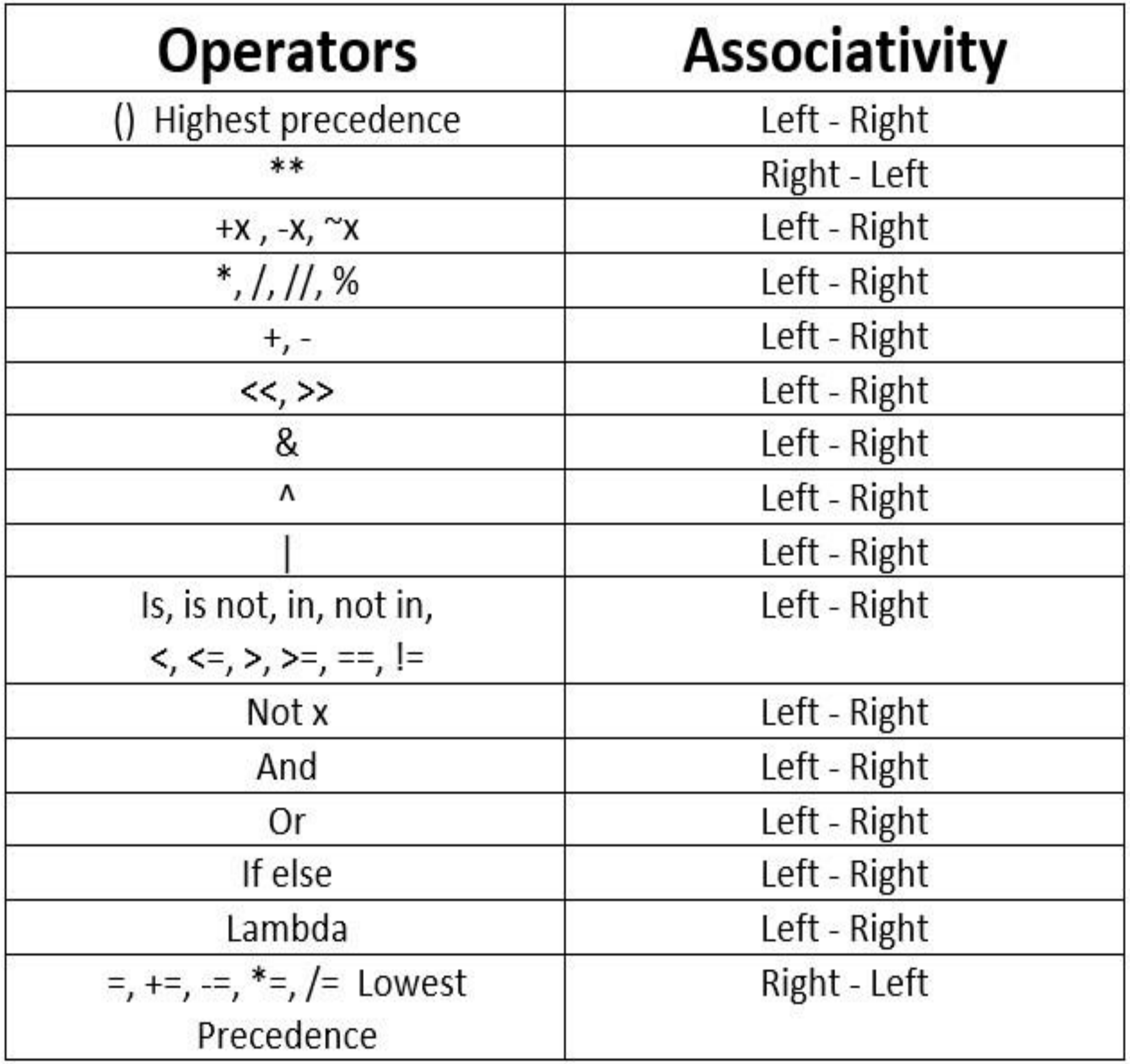


and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	with
def	finally	in	print	yield

Variable: A variable is a name that refers to a value,Key words are not used as variable names

Operators:

- **Arithmetic operators : +, -, *, /, //, %, ****
- **Assignment operators : =, +=, -=, *=, /=, %= , etc**
- **Comparison operators: ==, !=, >, <, >=, <=**
- **Logical operators: and , or, not**
- **Bitwise operator : &, |, ^, >>, <<**
- **Membership operators : in**
- **Identity operators: is , is not**



Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

Control Statements:

Sequential Control Statements

Selection Control Statements

- If
- If Else
- Nested If
- If..Elif..Else

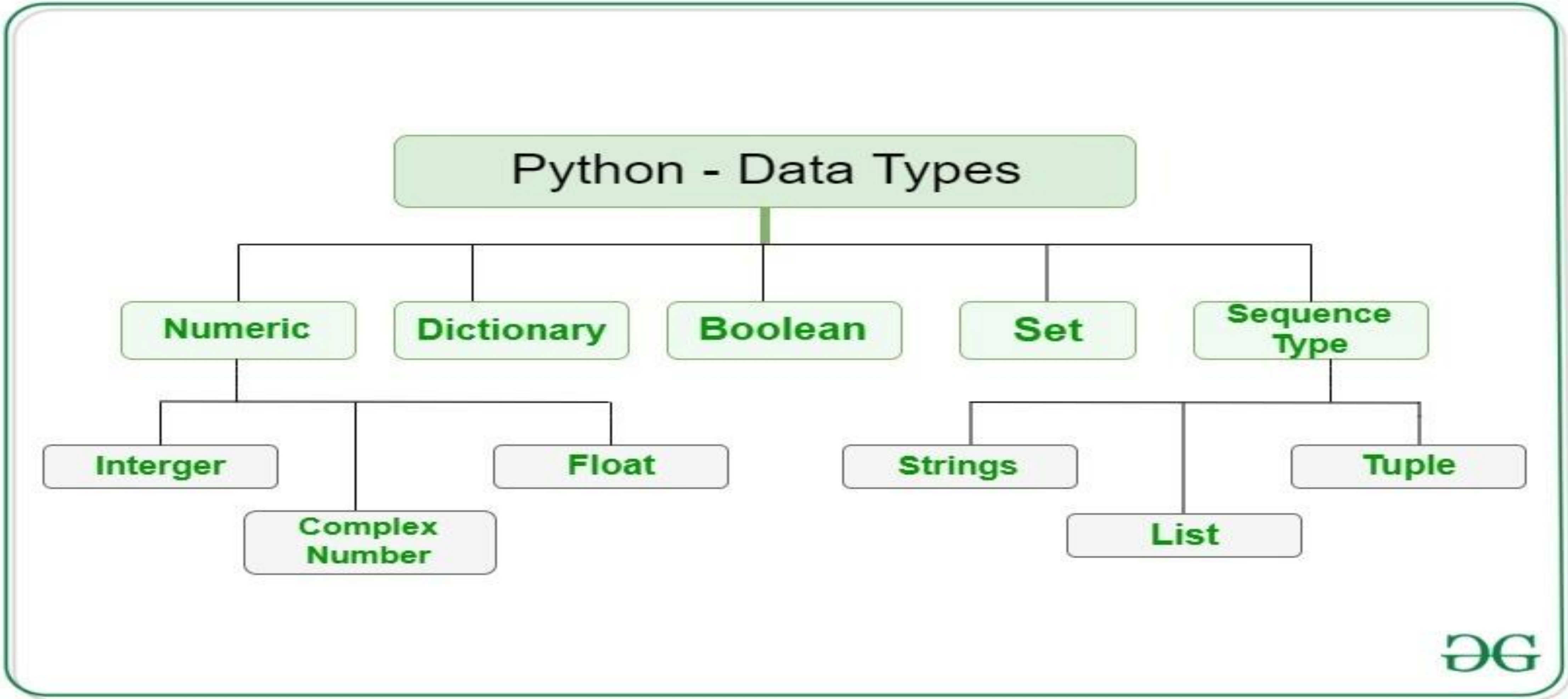
Loop Control Statements

- For
- While

Jump Statements:

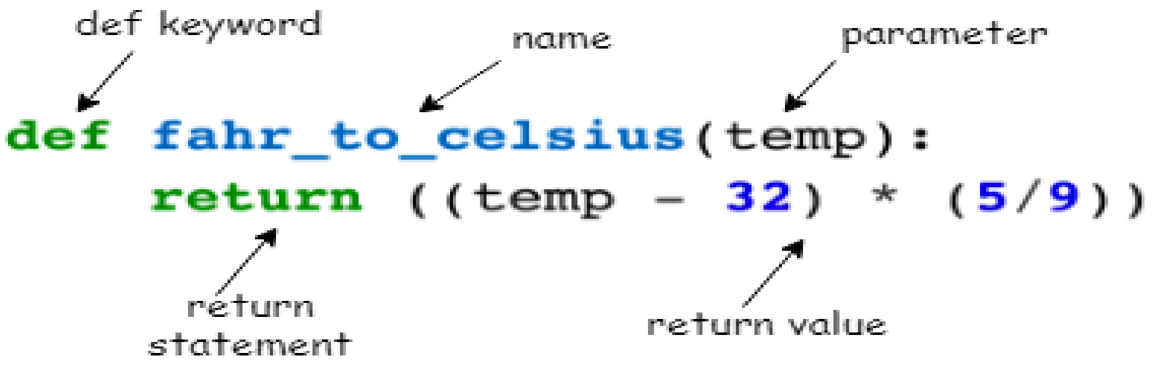
- Continue
- Break
- Pass

Data Types:


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Interger
Complex Number
Float
Strings
List
Tuple

Functions:

It is a block of code which only runs when it is called.You can pass data known as parameters, into a function.



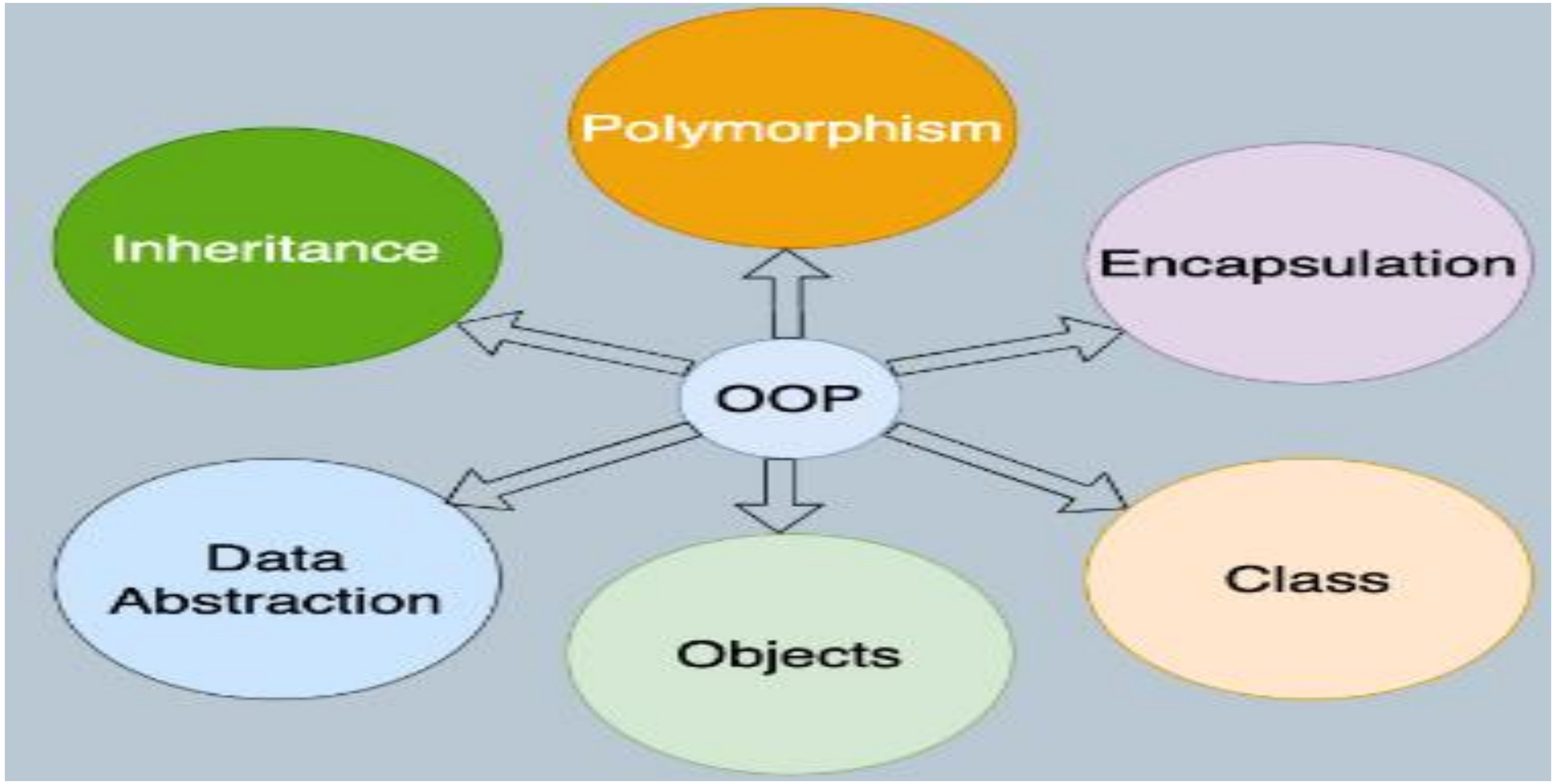
```
def fahr_to_celsius(temp):
    return ((temp - 32) * (5/9))
```

Types of Python:

1. Built-in Functions.
2. Recursion Functions
3. Lambda Functions.
4. User-defined Functions

OOP’s Concepts:

Object-Oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming.
The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.



Applications Using Python

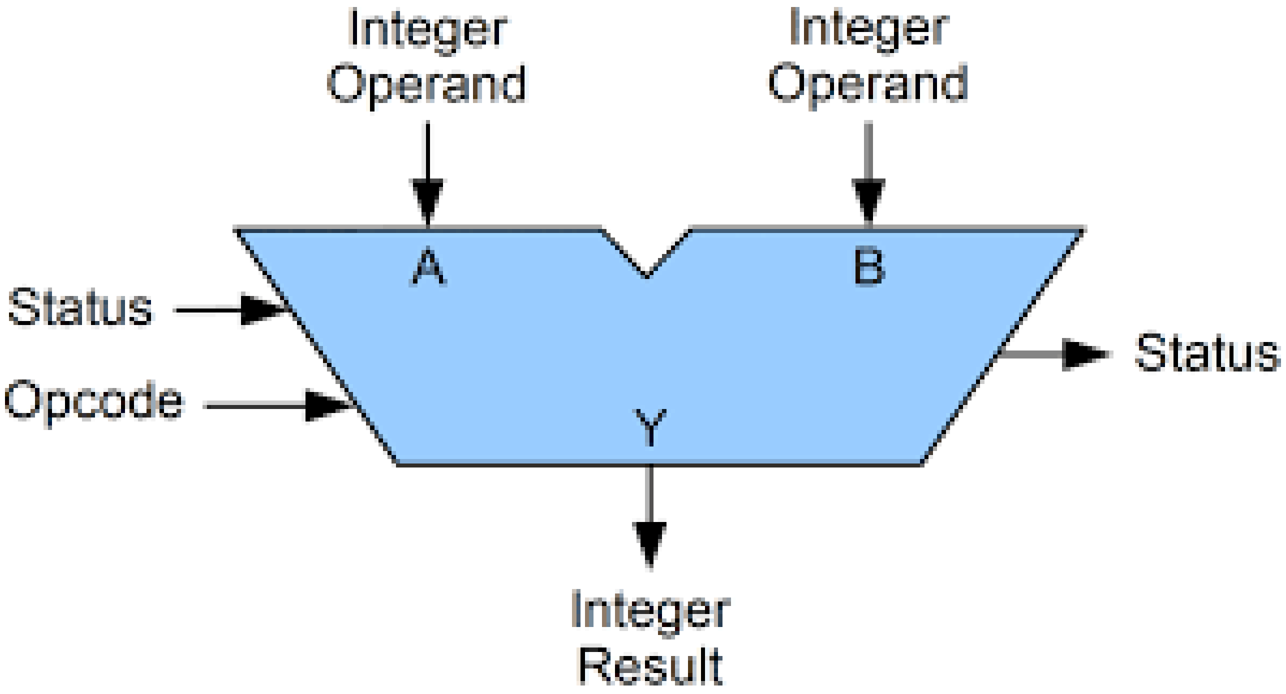
- Implementing ALU using Python

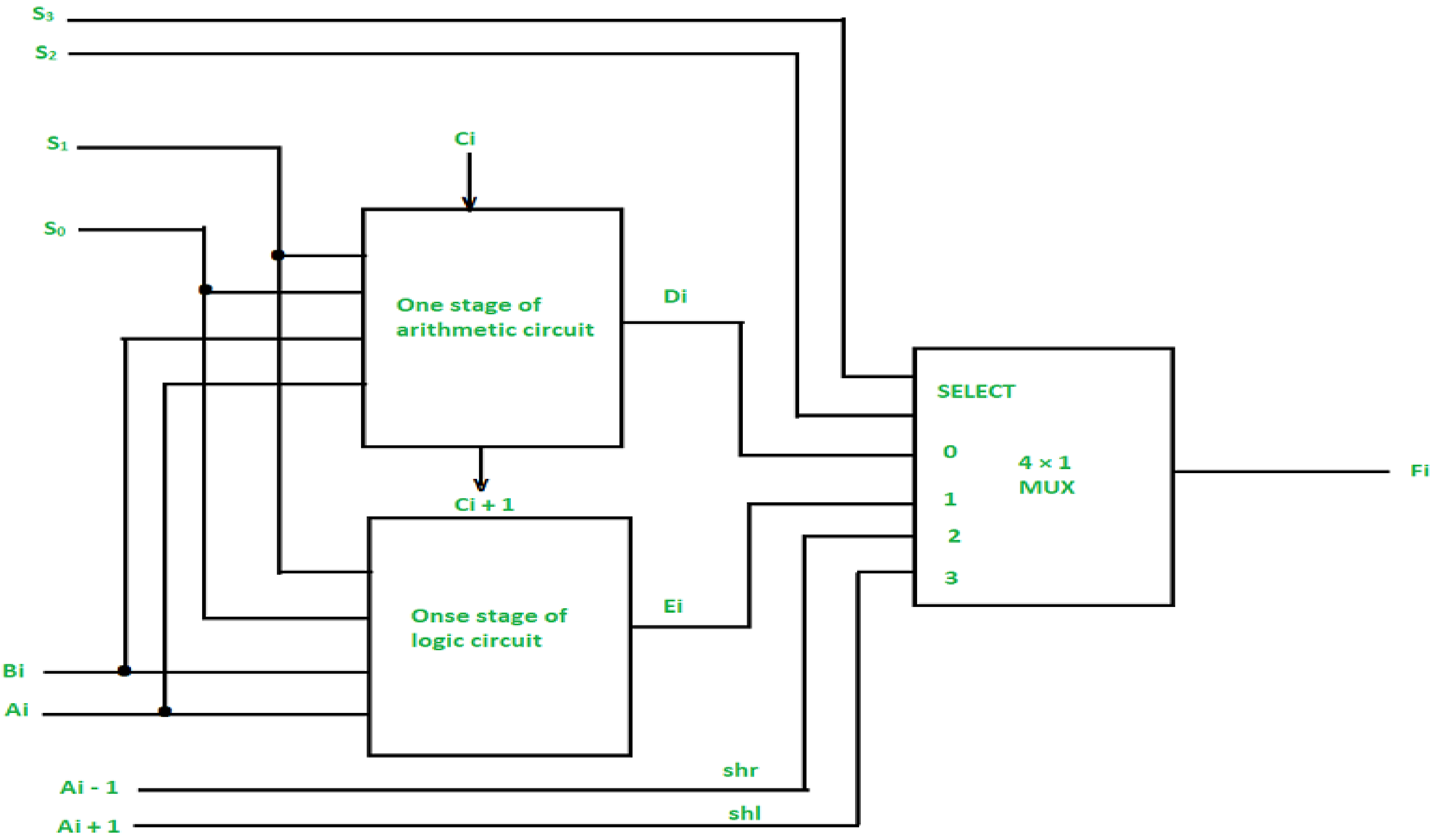
Arithmetic Logic Unit

An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands in computer instruction words. In some processors, the ALU is divided into two units: an arithmetic unit (AU) and a logic unit (LU). ALUs serve as a combinational digital circuit that performs arithmetic and bitwise operations on binary numbers. This is a foundational building block of arithmetic logic circuits for numerous types of control units and computing circuits including central processing units.

Operations that are performed by ALU

- Addition
- Subtraction
- Multiplication
- Division
- Modulo Division
- Increment
- Decrement
- And
- Or
- Xor
- LeftShift
- RightShift...




S₃
S₂
S₁
S₀
Ci
One stage of arithmetic circuit
Di
SELECT
0
4 × 1 MUX
1
2
3
Fi
Ci + 1
Onse stage of logic circuit
Ei
Bi
Ai
shr
Ai - 1
shl
Ai + 1

Algorithm

Step1: Create a class[ALU]

Step2: Define All operation by using def Function

Step3: Create two variables to store inputs

Step4: Assign unique address for every operation

Step5: By calling the function using its address perform the required operation

Code:

```
class ALU():
    def __init__(self,a,b):
        self.a = a
        self.b = b
    def add(self):
        return self.a+self.b
    def sub(self):
        return self.a-self.b
    def multiply(self):
        return self.a*self.b
    def divide(self):
        return self.a/self.b
    def modulo_division(self):
        return self.a%self.b
    def OR(self):
        return self.a or self.b
    def AND(self):
        return self.a and self.b
    def Not(self):
        return not(self.a)
    def Increment(self):
        return self.a+1
    def Decrement(self):
        return self.a-1
    def BitWise_AND(self):
        return self.a & self.b
    def BitWise_OR(self):
        return self.a | self.b
    def BitWise_XOR(self):
        return self.a ^ self.b
    def BitWise_LeftShift(self):
        return self.a << self.b
    def BitWise_RightShift(self):
        return self.a >> self.b
 a=int(input('Enter First Number: '))
b=int(input('Enter First Number: '))
operator=ALU(a,b)
```

```
x = (' 1.Add \n 2.Sub \n 3.Multiply \n 4.Divide \n 5.modulo division \n 6.or \n 7.and \n 8.not \n 9.Increment \n 10.Decrement 11.Bitwise And \n 12.Bitwise Or \n 13.Bitwise Xor \n 14.Leftshift \n 15.Rightshift')
print(x)
while True:
    choice = int(input('please select one of the following:'))
    if choice ==1:
        print("result: ",operator.add())
    elif choice ==2:
        print("result: ",operator.sub())
    elif choice ==3:
        print("result: ",operator.multiply())
    elif choice ==4:
        print("result: ",operator.divide())
    elif choice ==5:
        print("result: ",operator.modulo_division())
    elif choice ==6:
        print("result: ",operator.OR())
```

```
elif choice ==7:
       print("result: ",operator.AND())
   elif choice ==8:
       print("result: ",operator.Not())
   elif choice ==9:
       print("result: ",operator.Increment())
   elif choice ==10:
       print("result: ",operator.Decrement())
   elif choice ==11:
       print("result: ",operator.BitWise_AND())
   elif choice ==12:
       print("result: ",operator.BitWise_OR())
   elif choice ==13:
       print("result: ",operator.BitWise_XOR())
   elif choice ==14:
       print("result: ",operator.BitWise_LeftShift())
   elif choice ==15:
       print("result: ",operator.BitWise_RightShift())
   else:
       print('Invalid Option')
       break
   print()
```

Output:

```
Enter First Number: 8
Enter First Number: 4
 1.Add
 2.Sub
 3.Multiply
 4.Divide
 5.modulo division
 6.or
 7.and
 8.not
 9.Increment
 10.Decrement
 11.Bitwise And
 12.Bitwise Or
 13.Bitwise Xor
 14.Leftshift
 15.Rightshift
```

```
please select one of the following:1
result:  12
please select one of the following:2
result:  4
please select one of the following:3
result:  32
please select one of the following:4
result:  2.0
please select one of the following:5
result:  0
please select one of the following:6
result:  8
please select one of the following:7
result:  4
please select one of the following:8
result:  False
```

```
please select one of the following:9
result:  9
please select one of the following:10
result:  7
please select one of the following:11
result:  0
please select one of the following:12
result:  12
please select one of the following:13
result:  12
please select one of the following:14
result:  128
please select one of the following:15
result:  0
please select one of the following:16
Invalid Option
```

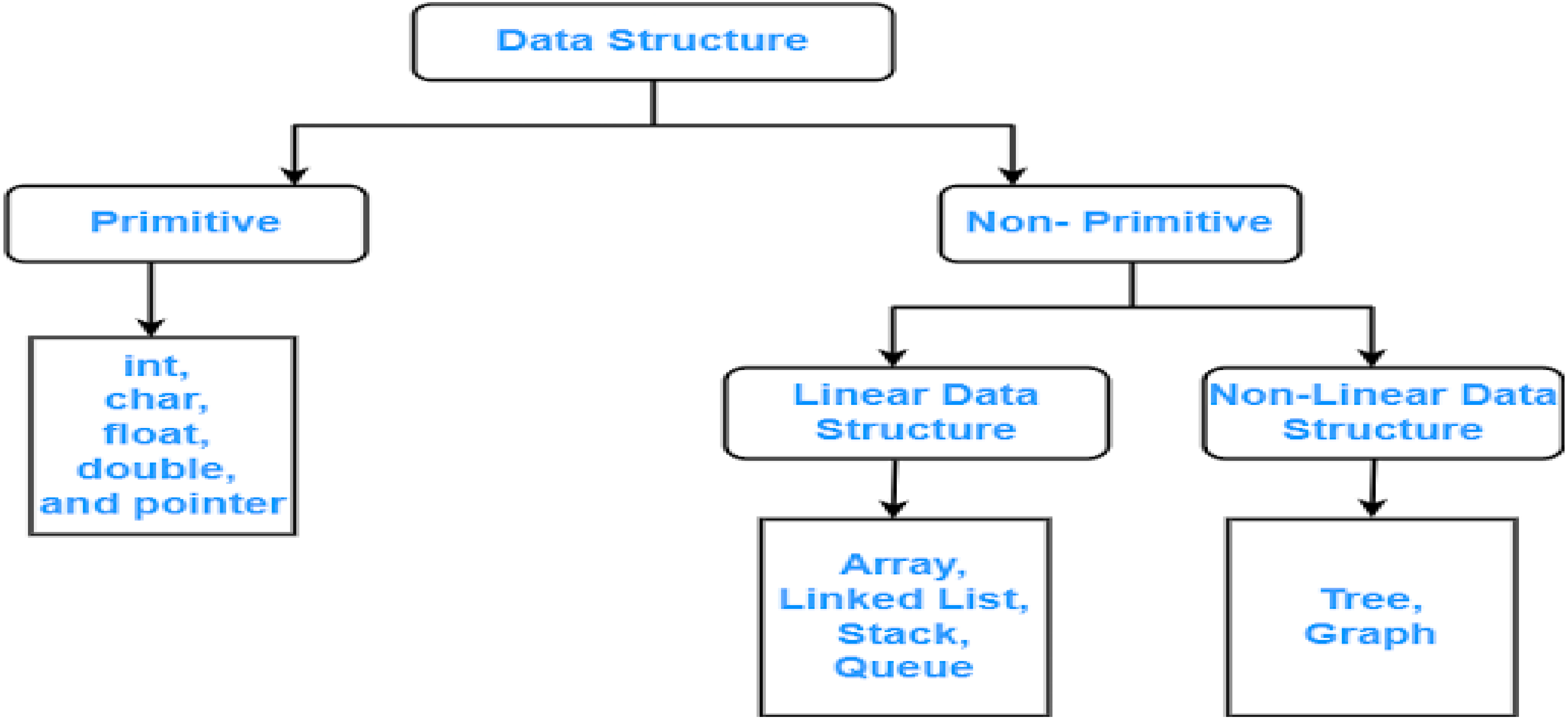
Data Structures

- Data structure is representation of the logical relationship existing between individual elements of data.
- a data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
- Data structure affects the design of both structural & functional aspects of a program.

Program = algorithm + Data Structure

- Algorithm is a step by step procedure to solve a particular function.
- Data structure are normally divided into two broad categories:

1. Primitive Data Structure
2. Non-Primitive Data Structure



Data Structure

Primitive

int,
char,
float,
double,
and pointer

Non- Primitive

Linear Data Structure

Array,
Linked List,
Stack,
Queue

Non-Linear Data Structure

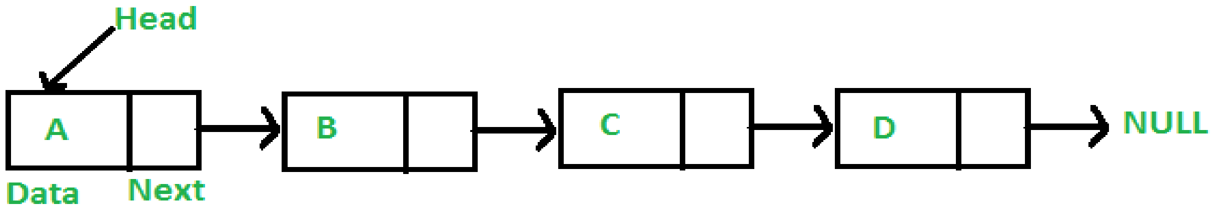
Tree,
Graph

Array:

- An array is defined as a set of finite number of homogeneous elements or same data items.
- It means an array can contain one type of data only, either all integer, all float-point number or all character.
- The elements of array will always be stored in the consecutive (continues) memory location.

Linked Lists:

- A Linear linked list can be defined as a collection of variable number of data items.
- Lists are the most commonly used non-primitive data structures. An element of list must contain at least two fields, one for storing data or information and other for storing address of next element.



Stack:

- A stack is also an ordered collection of elements like arrays, but it has a special feature that deletion and insertion of elements can be done only from one end called the top of the stack (TOP).
- Due to this property it is also called as last in first out type of data structure (LIFO).
- Insertion of element into stack is called PUSH and deletion of element from stack is called POP.

Queue:

- Queue are first in first out type of data structure (i.e. FIFO)
- In a queue new elements are added to the queue from one end called REAR end and the element are always removed from other end called the FRONT end.

Both Stack and queue can be implemented into two ways:

- Using arrays (Static implementation)
- Using pointer (Dynamic implementation)

Trees:

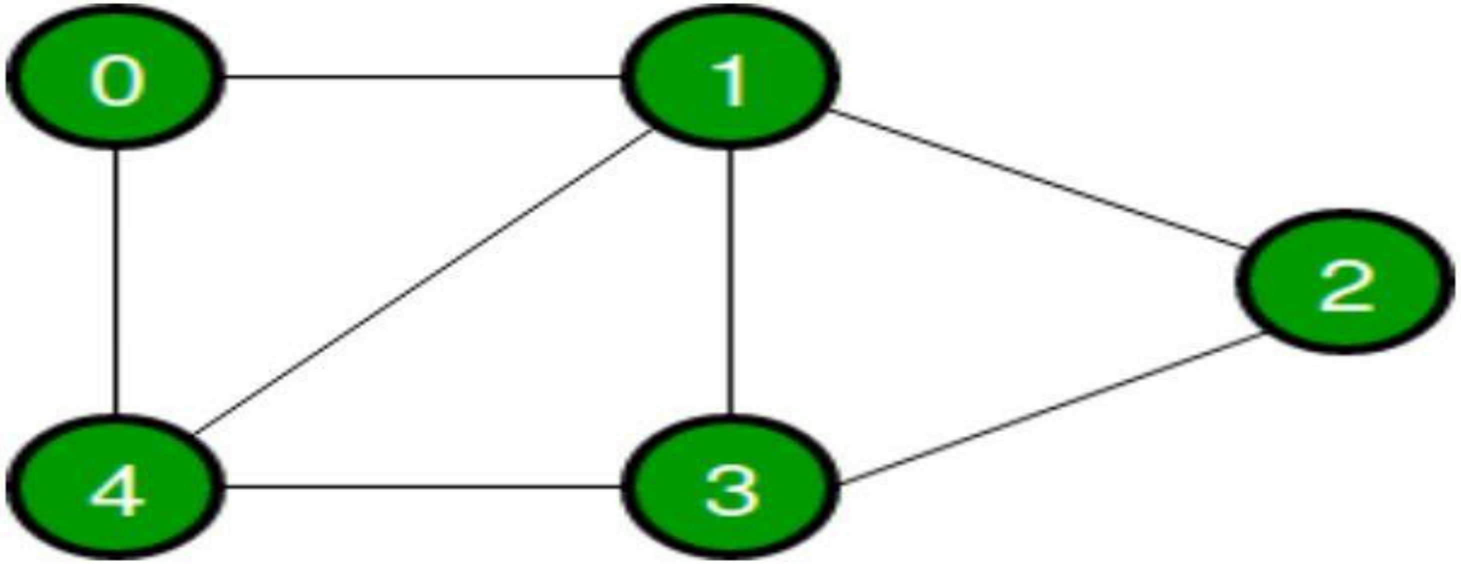
- A tree can be defined as finite set of data items (nodes).
- Tree is non-linear type of data structure in which data items are arranged or stored in a sorted sequence.
- Tree represent the hierarchical relationship between various elements.
- A Tree contain Root,Children,sibling,leaf nodes..

Graphs:

- Graph is a mathematical non-linear data structure capable of representing many kind of physical structures.
- It has found application in Geography, Chemistry and Engineering sciences.
- Definition: A graph G(V,E) is a set of vertices V and a set of edges E.

Applications Using Data Structures

- ❖ **Representation of Graph using Arrays**



Representation of Graph:

We can represent a graph using Adjacency list,which can be implemented by using Arrays.
An array of lists is used. The size of the array is equal to the number of vertices.
An entry array[i] represents the list of vertices adjacent to the i-th vertex.
This representation can also be used to represent a weighted graph.
The weights of edges can be represented as lists of pairs.
Following is the adjacency list representation of the above graph.

```
class AdjNode:
    def __init__(self, data):
        self.vertex = data
        self.next = None
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [None] * self.V
    def add_edge(self, src, dest):
        node = AdjNode(dest)
        node.next = self.graph[src]
        self.graph[src] = node
        node = AdjNode(src)
        node.next = self.graph[dest]
        self.graph[dest] = node
    def print_graph(self):
        for i in range(self.V):
            print("Adjacency list of vertex {}\n head".format(i), end="")
            temp = self.graph[i]
            while temp:
                print(" -> {}".format(temp.vertex), end="")
                temp = temp.next
            print(" \n")
```

```
if __name__ == "__main__":
    V = 5
    graph = Graph(V)
    graph.add_edge(0, 1)
    graph.add_edge(0, 4)
    graph.add_edge(1, 2)
    graph.add_edge(1, 3)
    graph.add_edge(1, 4)
    graph.add_edge(2, 3)
    graph.add_edge(3, 4)

    graph.print_graph()
```

Output:

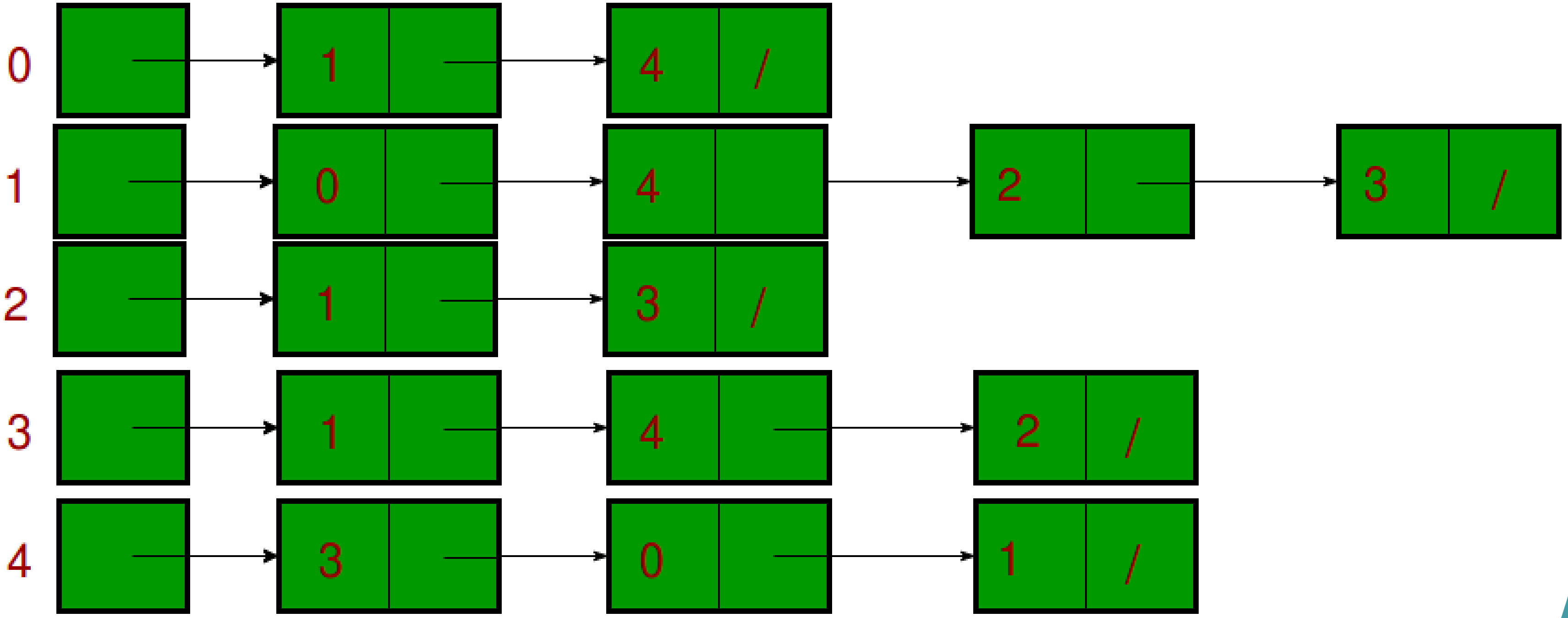
```
Adjacency list of vertex 0
 head -> 4 -> 1

Adjacency list of vertex 1
 head -> 4 -> 3 -> 2 -> 0

Adjacency list of vertex 2
 head -> 3 -> 1

Adjacency list of vertex 3
 head -> 4 -> 2 -> 1

Adjacency list of vertex 4
 head -> 3 -> 1 -> 0
```



Thank You