# VIRUDHUNAGAR DISTRICT COMMON EXAMINATIONS COMMON QUARTERLY EXAMINATION – SEPTERMBER 2024

STD: XII COMPUTER SCIENCE

## PART – I

**I. Answer all and Choose the best answer:**

1. d. Interpreter

2. d. Abstract data type

3. b. Protected members

4. a. Algorithmic solution

5. a.#

6. a. for

7. b.Recursive

8. a. Required

9. c. Immutable

10. d. third argument of slice operation

11. b.8

12. c. extend()

13. d. __del__( )

14. d. Instantiation

15. b. else

## PART – II

**II. Answer ANY 6 of the following and question no. 33 is compulsory: 6 X 2 = 12**

**16. What is Subroutine?**

- ☯ Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly.

**17. Differentiate Constructors and Selectors.**

| CONSTRUCTORS | SELECTORS |
|---|---|
| Constructors are functions that build the abstract data type. | Selectors are functions that retrieve information from the data type. |
| Constructors create an object, bundling together different pieces of information | Selectors extract individual pieces of information from the object. |

**18. How Python represents the private and protected Access specifiers?**

- Python prescribes a convention of adding a prefix **__(double underscore)** results in a variable name or method becoming **private**.
  **Example: self.__n2=n2**
- Adding a prefix _ **(single underscore)** to a variable name or method makes it protected.
  **Example: self._sal = sal**

**19. Write the phases of performance evaluation of an algorithm.**

**A Priori estimates:** This is a theoretical performance analysis of an algorithm. Efficiency of an algorithm is measured by assuming the external factors.

**A Posteriori testing:** This is called performance measurement. In this analysis, actual statistics like running time and required for the algorithm executions are collected.

20. **What are the different operators that can be used in Python ?**

**Operators that can be used in Python:**

- Operators are special symbols which represent computations, conditional matching in programming.
- Operators are categorized as Arithmetic, Relational, Logical, Assignment and Conditional.

21. **Write note on break statement.**

**break statement :**

- The **break** statement terminates the loop containing it.

Control of the program flows to the statement immediately after the body of the loop.

**22. What are the main advantages of function?**

**Main advantages of functions are ,**

- It avoids repetition and makes high degree of code reusing.
- It provides better modularity for your application.

**23. How will you delete a string in Python?**

- ☯ Python will not allow deleting a particular character in a string.
- ☯ Whereas you can remove entire string variable using **del** command.

**Example:**

del str1[2]

**24. Write the syntax of creating a Tuple with n number of elements.**

Syntax:

```
Tuple_Name = (E1, E2, E2 ……. En)              # Tuple with n number elements
Tuple_Name = E1, E2, E3 ….. En           # Elements of a tuple without parenthesis
```

**PART – III**

**III. Answer ANY 6 of the following and question no. 33 is compulsory: 6 X 3 = 18**

**25. Differentiate concrete data type and abstract data type.**

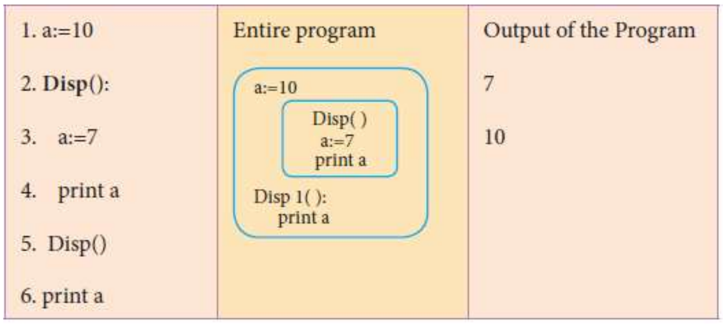
| **Pure functions** | **Impure functions** |
|---|---|
| Pure functions will give exact result when the same arguments are passed. | Impure functions never assure you that the function will behave the same every time it's called. |
| Pure function does not cause any side effects to its output. | Impure function causes side effects to its output. |
| The return value of the pure functions solely depends on its arguments passed. | The return value of the impure functions does not solely depend on its arguments passed. |
| They do not modify the arguments which are passed to them. | They may modify the arguments which are passed. |
| Example: strlen(), sqrt() | Example: random(), Date() |

**26. . Define Global scope with an example.**

**Global scope**

- ☯ A variable which is declared outside of all the functions in a program is known as global variable.
- ☯ Global variable can be accessed inside or outside of all the functions in a program.

**Example:**



| | | |
|---|---|---|
| 1. a:=10<br>2. Disp():<br>3. a:=7<br>4. print a<br>5. Disp()<br>6. print a | Entire program<br>a:=10<br>Disp( )<br>a:=7<br>print a<br>Disp 1( ):<br>print a | Output of the Program<br>7<br>10 |

### 27. List the characteristics of an algorithm.

| Characteristics | Meaning |
|---|---|
| Input | Zero or more quantities to be supplied. |
| Output | At least one quantity is produced. |
| Finiteness | Algorithms must terminate after finite number of steps. |
| Definiteness | All operations should be well defined. |
| Effectiveness | Every instruction must be carried out effectively. |
| Correctness | The algorithms should be error free. |
| Simplicity | Easy to implement. |
| Unambiguous | Algorithm should be clear and unambiguous. Each of its steps should be clear and must lead to only one meaning. |
| Feasibility | Should be feasible with the available resources. |
| Portable | An algorithm should be generic, independent and able to handle all range of inputs. |
| Independent | An algorithm should have step-by-step directions, which should be independent of any programming code. |

### 28. Explain Ternary operator with examples.

**Ternary operator:**

- Ternary operator is also known as **conditional operator** that evaluates something based on a condition being true or false.
- It simply allows testing a condition in a single line replacing the multiline if-else making the code compact.

**Syntax:**

***Variable Name = [on_true] if [Test expression] else [on_false]***

**Example :**

min = 50 if 49<50 else 70        # Output: **min = 50**

**29. Write a program to display.**

```
A
A   B
A   B   C   D
A   B   C   D   E
```

**CODE:**

```
a=['A','B','C','D','E']
for i in range(0,6):
        for j in range(0,i):
                print(a[j],end=" ")
        else:
                print()
```

**30. Write the rules of local variable.**

**Rules of local variable:**

- A variable with local scope can be accessed only within the function/block that it is created in.
- When a variable is created inside the function/block, the variable becomes local to it.
- A local variable only exists while the function is executing.
- The formal arguments are also local to function.

**31. Write a note about count( ) function in python.**

**Count( ) function in python:**

- Returns the number of substrings occurs within the given range.
- Remember that substring may be a single character.
- Range (beg and end) arguments are optional. If it is not given, python searched in whole string.
- Search is case sensitive.

**SYNTAX:**

```
Count(str, beg, end)
```

**EXAMPLE:**

```
>>> str1="Raja Raja Chozhan"
>>> print(str1.count('Raja'))
```

**OUTPUT:**

```
2
```

**32. How do define constructor and destructor in Python?**

**CONSTRUCTOR:**

- **init”** is a special function begin and end with double underscore in Python act as a Constructor.
- Constructor function will automatically executed when an object of a class is created.

**General format of constructor:**

```
def __init__(self, [args ……..]):
<statements>
```

**DESTRUCTOR:**

- Destructor is also a special method gets executed automatically when an object exit from the scope.

In Python, __del__( ) method is used as destructor.

**General format of destructor:**

```
def __del__(self):
<statements>
```

**33. what will be the output of the following code?**

```
list=[2**x for x in range(5)]

print(list)
```

**output;**

```
1    2    4    8    16
```

**PART - IV**

34. a. **1. What are called Parameters and write a note on**

(i) Parameter without Type (ii) Parameter with Type

- **Parameters** are the variables in a function definition
- ***Arguments*** are the values which are passed to a function definition.

Two types of parameter passing are,

1. Parameter Without Type

2. Parameter With Type

**1. Parameter Without Type:**

- Lets see an example of a function definition of Parameter Without Type:

***(requires: b>=0 )***

***(returns: a to the power of b)***

***let rec pow a b:=***

```
if b=0 then 1
else a * pow a (b-1)
```

- In the above function definition **variable „ b‟** is the **parameter** and the **value** passed **to** the variable *„b‟* is the **argument**.
- The precondition ***(requires)*** and postcondition ***(returns)*** of the function is given.
- We have not mentioned any types: ***(data types)***. This is called parameter without type.
- In the above function definition the expression has type ***„int‟***, so the function's return type also be ***„int‟*** *by implicit*.

**2. Parameter With Type:**

- Now let us write the same function definition with types,
- In this example we have explicitly annotating the types of argument and return type as ***„int‟***.
- Here, when we write the type annotations for „a‟ and „b‟ the parantheses are mandatory.
- This is the way passing parameter with type which helps the compiler to easily infer them.

Or

**How will you facilitate data abstraction? Explain it with suitable example.**

- Data abstraction is supported by defining an abstract data type (ADT), which is a collection of constructors and selectors.
- To facilitate data abstraction, you will need to create two types of functions:
  - **Constructors**
  - **Selectors**

**a) Constructor:**

- Constructors are functions that build the abstract data type.
- Constructors create an object, bundling together different pieces of information.

For example, say you have an abstract data type called city.

- This city object will hold the city‟s name, and its latitude and longitude.
- To create a city object, you‟d use a function like **city = makecity (name, lat, lon).**
- Here makecity (name, lat, lon) is the constructor which creates the object city.

**b) Selectors:**

- Selectors are functions that retrieve information from the data type.
- Selectors extract individual pieces of information from the object.
- To extract the information of a city object, you would use functions like
  - ✓ **getname(city)**

- ✓ **getlat(city)**
- ✓ **getlon(city)**

These are the selectors because these functions extract the information of the city object.

### 35. Explain the types of scopes for variable or LEGB rule with example.

**SCOPE:**

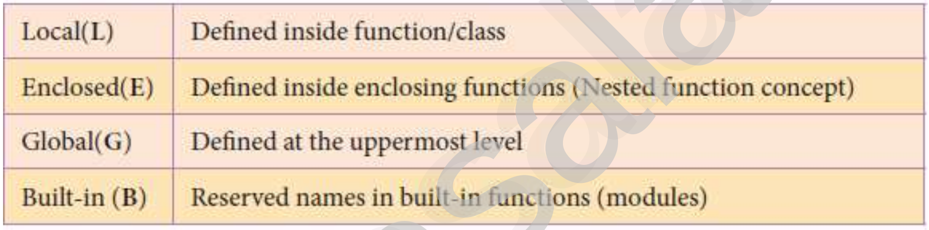
- ☯ Scope refers to the visibility of variables, parameters and functions in one part of a program to another part of the same program.

**TYPES OF VARIABLE SCOPE:**

- ☯ Local Scope
- ☯ Enclosed Scope
- ☯ Global Scope
- ☯ Built-in Scope

**LEGB RULE:**

- ☯ The **LEGB** rule is used to decide the order in which the scopes are to be searched for scope resolution.
- ☯ The scopes are listed below in terms of hierarchy (highest to lowest).

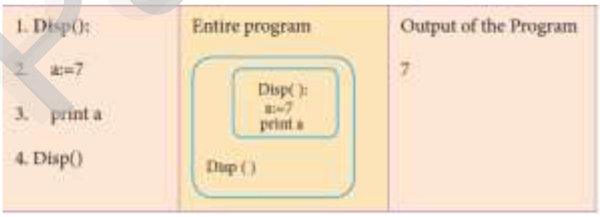


| | |
|---|---|
| Local(L) | Defined inside function/class |
| Enclosed(E) | Defined inside enclosing functions (Nested function concept) |
| Global(G) | Defined at the uppermost level |
| Built-in (B) | Reserved names in built-in functions (modules) |

**i) LOCAL SCOPE:**

- ☯ Local scope refers to variables defined in current function.
- ☯ A function will always look up for a variable name in its local scope.
- ☯ Only if it does not find it there, the outer scopes are checked.

**Example:**



```
Disp():
    a:=7
    print a
Disp()
```

Entire program

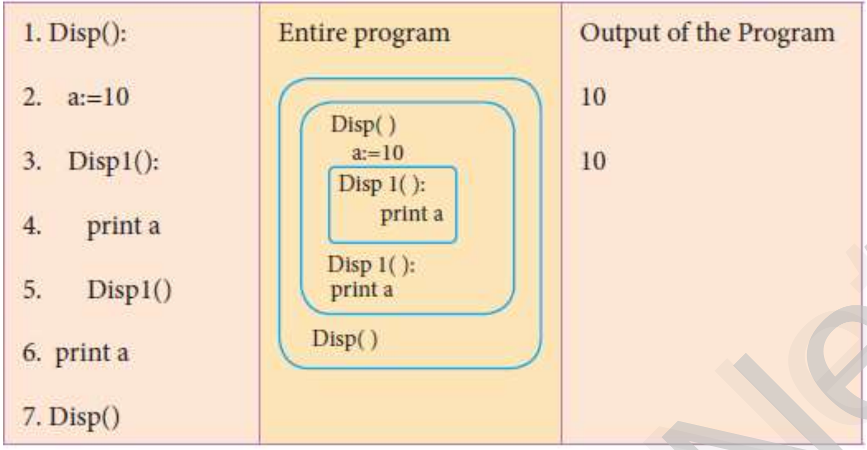
Output of the Program

```
7
```

- ☯ On execution of the above code the variable **a** displays the value 7, because it is defined and available in the local scope.

**ii) ENCLOSED SCOPE:**

- A variable which is declared inside a function which contains another function definition with in it, the inner function can also access the variable of the outer function. This scope is called enclosed scope.
- When a compiler or interpreter searches for a variable in a program, it first search Local, and then search Enclosing scopes.



| 1. Disp(): | Entire program | Output of the Program |
|---|---|---|
| 2. a:=10 | | 10 |
| 3. Disp1(): | | 10 |
| 4. print a | | |
| 5. Disp1() | | |
| 6. print a | | |
| 7. Disp() | | |

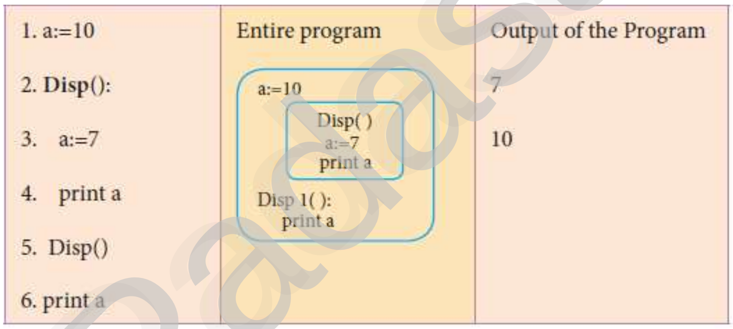
Disp( )
a:=10
Disp 1( ):
print a
Disp 1( ):
print a
Disp( )

- In the above example Disp1() is defined within Disp(). The variable „a‟ defined in Disp() can be even used by Disp1() because it is also a member of Disp().

### iii) GLOBAL SCOPE:

- A variable which is declared outside of all the functions in a program is known as global variable.
- Global variable can be accessed inside or outside of all the functions in a program.

**Example:**



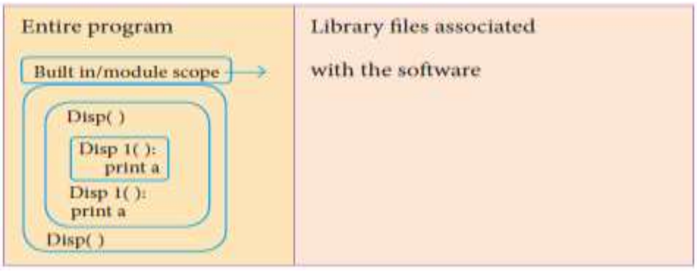
| 1. a:=10 | Entire program | Output of the Program |
|---|---|---|
| 2. Disp(): | | 7 |
| 3. a:=7 | | 10 |
| 4. print a | | |
| 5. Disp() | | |
| 6. print a | | |

a:=10
Disp( )
a:=7
print a
Disp 1( ):
print a

- On execution of the above code the variable **a** which is defined inside the function displays the value 7 for the function call Disp() and then it displays 10, because **a** is defined in global scope.

### iv) BUILT-IN-SCOPE:

- The built-in scope has all the names that are pre-loaded into the program scope when we start the compiler or interpreter.
- Any variable or module which is defined in the library functions of a programming language has Built-in or module scope.



**OR**

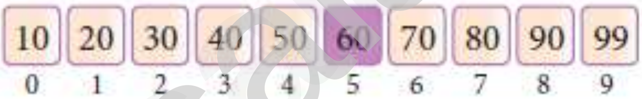
**What is Binary search? Discuss with example.**

**BINARY SEARCH:**

- Binary search also called half-interval search algorithm.
- It finds the position of a search element within a sorted array.
- The binary search algorithm can be done as divide-and-conquer search algorithm and executes in logarithmic time.

**Binary Search Working principles with example**:

- List of elements in an array must be sorted first for Binary search.
- The array is being sorted in the given example and it is suitable to do the binary search algorithm.
- Let us assume that the **search element is 60** and we need to search the location or index of search element 60 using binary search.

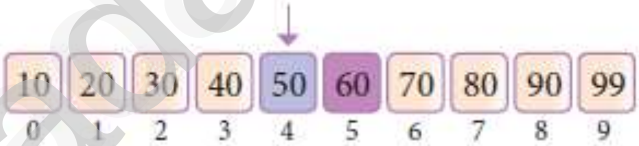


| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- First, we find index of middle element of the array by using this formula :

**mid = low + (high - low) / 2**

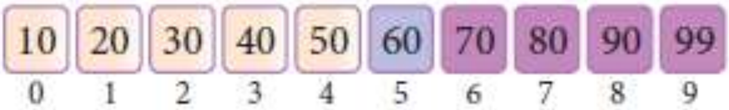
- Here it is, 0 + (9 - 0 ) / 2 = 4. So, 4 is the mid value of the array.



| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

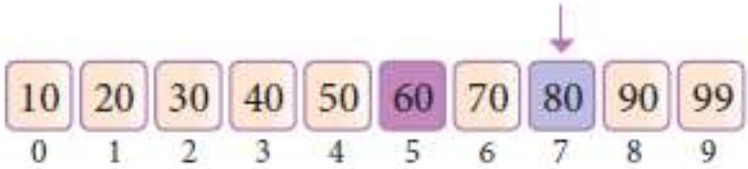
- Compare the value stored at index 4 with target value, which is not match with search element.

As the search value 60 > 50.



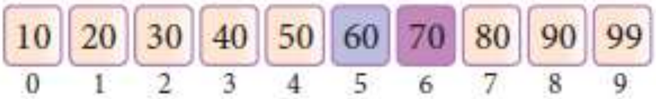
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- Now we change our search range **low to mid + 1** and find the new mid value as index 7.
- We compare the value stored at index 7 with our target value.
- Element not found because the value in index 7 is greater than search value . ( 80 > 60)



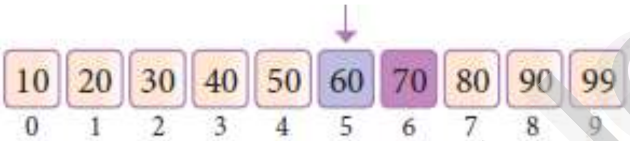
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- So, the search element must be in the lower part from the current mid value location



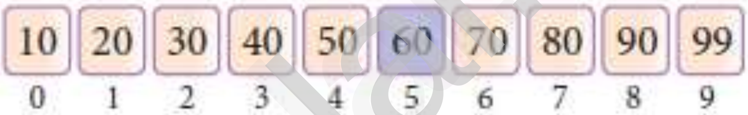
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- Now we change our search range **low to mid - 1** and find the new mid value as index 5



| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- Now we compare the value stored at location 5 with our search element.
- We found that it is a match.



| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- We can conclude that the search element 60 is found at location or index 5.

**36. Explain input () and print() function with examples.**

**Input and Output Functions**

- A program needs to interact with the user to accomplish the desired task; this can be achieved using

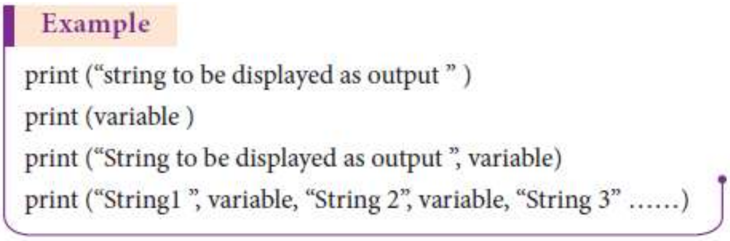
**Input-Output functions.**

- The **input()** function helps to enter data at run time by the user
- The output function **print()** is used to display the result of the program on the screen after execution.

**1) print() function**

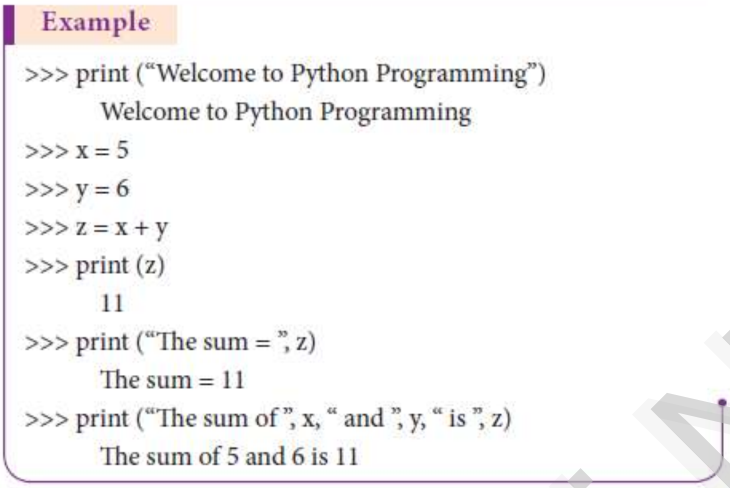
- In Python, the **print()** function is used to display result on the screen.

**Syntax for print():**



Example

```
print (“string to be displayed as output ” )
print (variable )
print (“String to be displayed as output ”, variable)
print (“String1 ”, variable, “String 2”, variable, “String 3” ……)
```



Example

```
>>> print (“Welcome to Python Programming”)
        Welcome to Python Programming
>>> x = 5
>>> y = 6
>>> z = x + y
>>> print (z)
        11
>>> print (“The sum = ”, z)
        The sum = 11
>>> print (“The sum of ”, x, “ and ”, y, “ is ”, z)
        The sum of 5 and 6 is 11
```

- The **print ( )** evaluates the expression before printing it on the monitor.
- The print () displays an entire statement which is specified within print( ).
- **Comma ( , )** is used as a separator in **print ( )** to print more than one item.

**2) input() function**

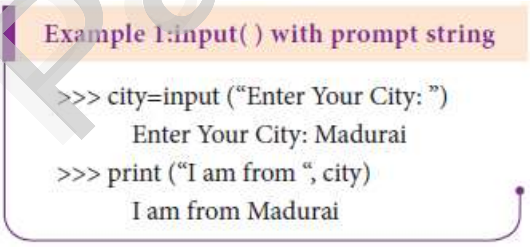
- In Python, **input( )** function is used to accept data as input at run time.

The syntax for **input()** function is,

Variable=input(“prompt string)

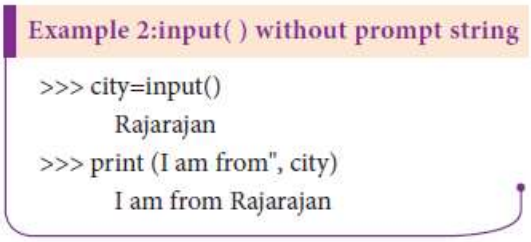
**“Prompt string”** in the syntax is a message to the user, to know what input can be given.

- If a prompt string is used, it is displayed on the monitor; the user can provide expected data from the input device.
- The **input( )** takes typed data from the keyboard and stores in the given variable.
- If prompt string is not given in **input( )**, the user will not know what is to be typed as input.



Example 1:input( ) with prompt string

```
>>> city=input (“Enter Your City: ”)
        Enter Your City: Madurai
>>> print (“I am from “, city)
        I am from Madurai
```



Example 2:input( ) without prompt string

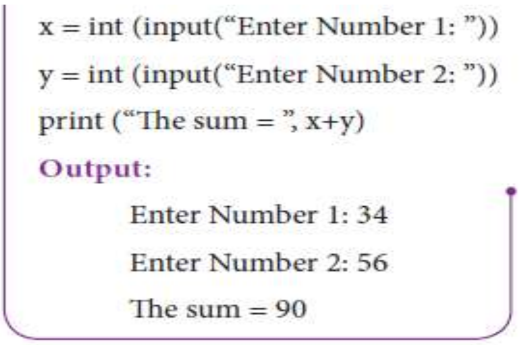
```
>>> city=input()
        Rajarajan
>>> print (I am from", city)
        I am from Rajarajan
```

**Example:**

**Input() using Numerical values:**

- The **input ( )** accepts all data as string or characters but not as numbers.
- The **int( )** function is used to convert string data as integer data explicitly.

**Example:**



```
x = int (input("Enter Number 1: "))
y = int (input("Enter Number 2: "))
print ("The sum = ", x+y)
Output:
        Enter Number 1: 34
        Enter Number 2: 56
        The sum = 90
```

**OR**

**Write a detail note on if..elif.....else statement with suitable example.**

**Nested if..elif...else statement:**

- When we need to construct a chain of **if** statement(s) then **„elif‟** clause can be used instead of **„else‟**.
- **„elif‟** clause combines **if..else-if..else** statements to one **if..elif…else.**
- **„elif‟** can be considered to be abbreviation of **„else if‟**.
- In an **„if‟** statement there is no limit of **„elif‟** clause that can be used, but an **„else‟** clause if used should be placed at the end.

**Syntax:**

```
if <condition-1>:
        statements-block 1
elif <condition-2>:
        statements-block 2
else:
        statements-block n
```

- In the syntax of **if..elif..else** mentioned above, condition-1 is tested if it is true then statementsblock1 is executed.
- Otherwise the control checks condition-2, if it is true statements-block2 is executed and even if it fails statements-block n mentioned in **else** part is executed.

**Example:**

```
m1=int (input("Enter mark in first subject : "))
m2=int (input("Enter mark in second subject : "))
avg= (m1+m2)/2
if avg>=80:
```

```
            print (“Grade : A”)
    elif avg>=70 and avg<80:
            print (“Grade : B”)
    elif avg>=60 and avg<70:
            print (“Grade : C”)
    elif avg>=50 and avg<60:
            print (“Grade : D”)
    else:
             (“Grade : E”)
```

**Output :**

```
Enter mark in first subject : 34
Enter mark in second subject : 78
Grade : D
```

**37. Explain the following built-in functions.**

(a) id() (b) chr() (c) round() (d) type() (e) pow()

| Function | Description | Syntax | Example |
|---|---|---|---|
| id ( ) | Return the “identity” of an object. i.e. the address of the object in memory. | id (object) | x=15<br>y='a'<br>print ('address of x is :',id (x))<br>print ('address of y is :',id (y))<br>**Output:**<br>address of x is : 1357486752<br>address of y is : 13480736 |
| chr ( ) | Returns the Unicode character for the given ASCII value. | chr(i) | c=65<br>print (chr (c))<br>**Output:**<br>A |
| round ( ) | Returns the nearest integer to its input.<br>1. First argument (number) is used to specify the value to be rounded | round (number [,ndigits]) | x= 17.9<br>print ('x value is rounded to', round (x))<br>**Output:**<br>X value is rounded to 18 |
| type ( ) | Returns the type of object for the given single object. | type (object) | x= 15.2<br>print (type (x))<br>**Output:**<br><class |

| | | | |
|---|---|---|---|
| | | | 'float'> |
| pow ( ) | Returns the computation of a,b i.e. (a**b ) a raised to the power of b. | pow(a,b) | a= 5<br>b= 2<br>print (pow (a,b))<br>**Output:**<br>25 |

**OR**

**b. Explain about string operators in python with a suitable example.**

**STRING OPERATORS:**

- Python provides the following string operators to manipulate string.

**(i) Concatenation (+)**

- Joining of two or more strings using plus **(+) operator** is called as **Concatenation**.

**Example**

```
>>> "welcome" + "Python"
```

**Output:**

```
'welcomePython'
```

**(ii) Append (+ =)**

- Adding more strings at the end of an existing string using **operator +=** is known as **append**.

**Example:**

```
>>> str1="Welcome to "
>>> str1+="Learn Python"
>>> print (str1)
```

**Output:**

```
Welcome to Learn Python
```

count(str, beg, end

**(iii) Repeating (*)**

- The multiplication operator (*) is used to display a string in multiple number of times.

**Example:**

```
>>> str1="Welcome "
>>> print (str1*4)
```

**Output:**

```
Welcome Welcome Welcome Welcome
```

**(iv) String slicing**

- Slice is a substring of a main string.
- A substring can be taken from the original string by using **[ ] slicing operator** and index values.
- Using slice operator, you have to slice one or more substrings from a main string.

**General format of slice operation:**

str[start:end]

- Where ***start*** is the beginning index and ***end*** is the last index value of a character in the string.
- Python takes the end value less than one from the actual index specified.

**Example: slice a single character from a string**

```
>>> str1="THIRUKKURAL"
>>> print (str1[0])
```

**Output:**

```
 T
```

**(v) Stride when slicing string**

- When the slicing operation, you can specify a third argument as the stride, which refers to the number of characters to move forward after the first character is retrieved from the string.
- The default value of stride is 1.
- Python takes the last value as n-1
- You can also use negative value as stride, to prints data in reverse order.

**Example:**

```
>>> str1 = "Welcome to learn Python"
>>> print (str1[10:16])
>>> print(str1[::-2])
```

**Output:**

```
Learn
nhy re teolW
```

**38. . Explain the different set operations supported by python with suitable example.**

- A Set is a mutable and an unordered collection of elements without duplicates.

**Set Operations:**

- The set operations such as Union, Intersection, difference and Symmetric difference.

**(i) Union:**

- It includes all elements from two or more sets.

- The **operator |** is used to union of two sets.
- The function union( ) is also used to join two sets in python.

**Example:**

```
set_A={2,4,6,8}
set_B={'A', 'B', 'C', 'D'}
U_set=set_A|set_B
print(U_set)
```

**Output:**

```
{2, 4, 6, 8, 'A', 'D', 'C', 'B'}
```

**(ii) Intersection:**

- It includes the common elements in two sets.
- The **operator &** is used to intersect two sets in python.
- The function **intersection( )** is also used to intersect two sets in python.

**Example:**

```
set_A={'A', 2, 4, 'D'}
set_B={'A', 'B', 'C', 'D'}
print(set_A & set_B)
```

**Output:**

```
{'A', 'D'}
```

**(iii) Difference:**

- It includes all elements that are in first set (say set A) but not in the second set (say set B).
- The minus **(-) operator** is used to difference set operation in python.
- The function **difference( )** is also used to difference operation.

**Example:**

```
set_A={'A', 2, 4, 'D'}
set_B={'A', 'B', 'C', 'D'}
print(set_A - set_B)
```

**Output:**

```
{2, 4}
```

**(iv) Symmetric difference**

- It includes all the elements that are in two sets (say sets A and B) but not the one that are common to two sets.
- The caret **(^) operator** is used to symmetric difference set operation in python.
- The function **symmetric_difference( )** is also used to do the same operation.

**Example:**

```
set_A={'A', 2, 4, 'D'}
set_B={'A', 'B', 'C', 'D'}
print(set_A ^ set_B)
```

**Output:**

```
{2, 4, 'B', 'C'}
```

OR

**Write a program using class to accept three sides of a triangle and print its area.**

```
a = float(input('Enter first side: '))
b = float(input('Enter second side: '))
c = float(input('Enter third side: '))
s = (a + b + c) / 2.
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is %0.2f' %area)
```

**output:**

```
Enter first side: 3
Enter second side: 4
Enter third side: 5
The area of the triangle is 6.00
```

**************************

Mrs. GEETHAMARIMUTHU, M.Sc.,B.Ed.

P.G. Computer Science

VMG RR SRI SARADA SAKTHI MHSS

VIRUDHUNAGAR