3. SCOPING

2 Marks

1. What is a scope?

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

2. Why scope should be used for variable. State the reason.

Scope should be used for **variable** because **every part of the program** can **access the variable**.

3. What is Mapping?

- The process of **binding a variable name** with an **object** is called **mapping.**
- **= (equal to sign)** is used in programming languages to **map the variable and object.**

4. What do you mean by Namespaces?

Namespaces are **containers** for **mapping names of variables to objects**.

(name := object)

Example

a := 5

5. How Python represents the private and protected Access Specifiers?

Python prescribes a convention of **prefixing the name of the variable/method** with **single or double underscore** to emulate the behavior of **protected and private access specifiers.**

3 Marks

1. Define Local scope with an example.

- Local scope refer to **variable defined in current function.**
- A function will **first 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( )

Output

7
```

2. Define Global scope with an example.

- 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

```
a := 10
Disp( ):
    a := 7
    print a
Disp( )
print a
Output
7
10
```

3. Define Enclosed scope with an example.

- 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**.

Example

```
Disp( ):
    a:=10
    Disp1( ):
        print a
    Disp1( )
    print a
Disp( )
Output
10
10
```

4. Why access control is required?

- Access control is a **security technique** that regulates **who or what can view** or **use resources** in a **computing environment**.
- It is a fundamental concept in **security** that **minimizes risk** to the **object.**
- Access control is a **selective restriction of access to data**.
- Access control is **private, protected, public**.

5. Identify the scope of the variables in the following pseudo code and write its Output

```
color:= 'Red'
mycolor( ):
    b:= 'Blue'
    myfavcolor( ):
        g:= 'Green'
        print color, b, g
    myfavcolor( )
    print color, b
mycolor( )
print color
```

Ans:

Output

```
Red Blue Green
Red Blue
Red
```

Scope of the variables

Variables	**Scope**
color:= ‘Red’	Global Scope
b:= ‘Blue’	Enclosed Scope
g:= ‘Green’	Local Scope

5 Marks

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

LEGB rule is used to **decide the order** in which the **scopes** are to be **searched for scope resolution**.

Local Scope

Enclosed Scope

Global Scope

Built-in Scope

Local Scope

- Local scope refer to **variable defined in current function.**
- A function will **first 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( )

Output

        7
```

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

```
    a := 10
    Disp( ):
        a := 7
        print a
    Disp( )
    print a
Output
    7
    10
```

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**.

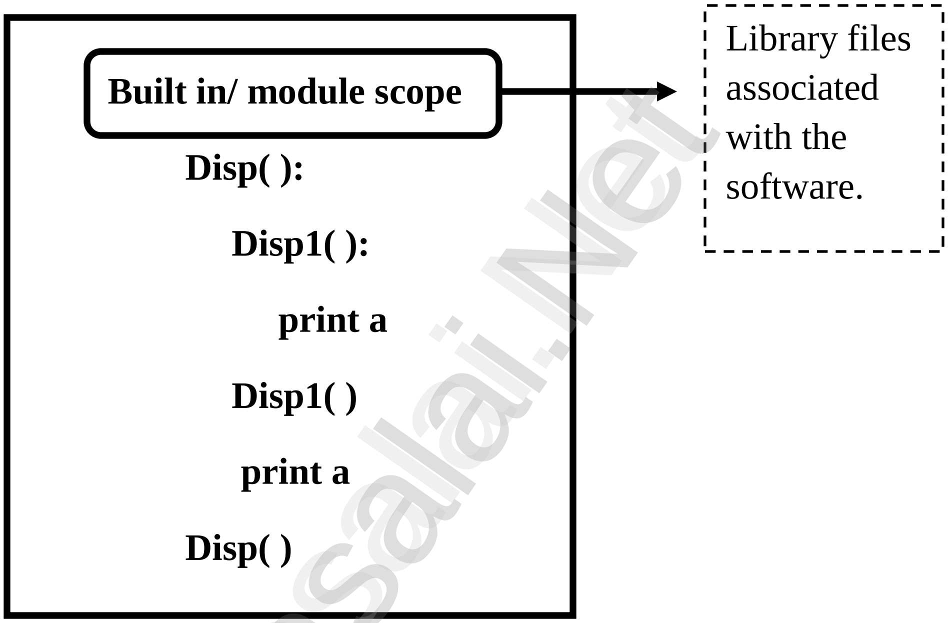
Example

```
    Disp( ):
        a:=10
        Disp1( ):
            print a
        Disp1( )
        print a
    Disp( )
Output

    10
    10
```

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**.



```
Disp( ):
    Disp1( ):
        print a
    Disp1( )
    print a
Disp( )
```

2. Write any Five Characteristics of Modules.

- Modules contain **instructions, processing, logic, and data**.
- Modules can be **separately compiled** and **stored in a library**.
- Modules can be **included in a program.**
- Module segments can be used by invoking a **name** and **some parameters.**
- Module **segments** can be used by **other modules**.

3. Write any five benefits in using modular programming.

- ➔ Modular programming allows **many programmers** to **collaborate** on **the same application**
- ➔ **Less code** to be **written.**
- ➔ The code is **stored** across **multiple files**.
- ➔ Code is **short, simple and easy to understand.**
- ➔ Errors can **easily be identified**, as they are localized to a **subroutine or function.**
- ➔ The **same code** can be used in **many applications**.
- ➔ The scoping of **variables can easily be controlled**.

4. ALGORITHMIC STRATEGIES

2 Marks

1. What is an Algorithm?

- An algorithm is a **finite set of instructions** to accomplish a **particular task.**
- It is **a step-by-step procedure** for solving a **given problem**.

2. Define Pseudo code.

- Pseudo code is a **methodology** that allows the **programmer** to represent the **implementation of an algorithm**.
- It has **no syntax** like programming languages and thus **can't be compiled or interpreted by the computer**.

3. Who is an Algorist?

- A person skilled in the **design of algorithms** are called as **Algorist**.
- An algorithmic **artist**.

4. What is Sorting?

- Sorting is defined as an **arrangement of data** in a **ascending or descending** order.
- Sorting techniques are used to **Bubble Sort, Selection Sort, Insertion Sort.**

5. What is searching? Write its types.

- Searching Algorithms are designed to **check for an element** or **retrieve** an **element** form any data structure where it is **stored.**
  - ☑ Linear Search
  - ☑ Binary Search

3 Marks

1. List the characteristics of an algorithm.

- Input
- Output
- Finiteness
- Definiteness
- Effectiveness
- Correctness
- Simplicity
- Unambiguous
- Feasibility
- Portable
- Independent

2. Discuss about Algorithmic complexity and its types.

- Algorithmic complexity is concerned about **running time** and **storage space** required by the **algorithm**. **f(n)** – ***n*** *as the* ***size of input data***.

<u>Time Complexity</u>

- The Time complexity of an algorithm is given by the **number of steps taken by the algorithm** to **complete the process**.

<u>Space Complexity</u>

- Space complexity of an algorithm is **the amount of memory required** to **run to its completion**.

3. What are the factors that influence time and space complexity.

- **Time Factor** is measured by **counting the number of key operations** like **comparisons** in the **sorting algorithm.**
- **Space Factor** is measured by the **maximum memory space required** by the **algorithm**.
- Algorithmic complexity is concerned about **running time** and **storage space** required by the **algorithm**. **f(n)** – ***n*** *as the* ***size of input data***.

4. Write a note on Asymptotic notation.

- **Asymptotic Notations** are languages that uses **meaningful statements** about **time and space complexity**.
- **Big O** - Big Oh is used to describe the **upper bound (worst case)** of a asymptotic function **- O(n log n)**
- **Big Ω** - Big Omega is the **reverse** Big O. It is used to describe the **lower bound (best case)** of a asymptotic function **- Ω(n log n)**
- **Big Θ** - Big Theta is used to describe the **lower bound = upper bound (best case and worst case)** of a asymptotic function - **Θ(n log n)**

5. What do you understand by Dynamic programming?

- Dynamic programming is an **algorithmic design method** is used when the **solution to a problem** can be **viewed** as the **result** of a **sequence of decisions.**
- Dynamic programming approach is similar to **divide and conquer.** The problem can be **divided into smaller sub-problems**.
- **Results** can be **re-used** to **complete the process**.
- Dynamic programming approaches are used to **find the solution in optimized way.**

5 Marks

1. Explain the characteristics of an algorithm.

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**.
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**.

2. Discuss about Linear Search algorithm.

Linear Search

- Linear search also called **sequential search** is a sequential method for **finding a particular value in a list.**
- The **search element** with **each element** is **found or the list exhausted**.
- List need **not be ordered**.

Pseudo code

```
1. Traverse the array using for loop.
2. In every iteration, compare the target, search key value with the current
   value of the list.
       If the values match, display the current index and value of the array.
       If the values do not match, move on to the next array element.
3. If no match is found display the search element not found.
```

Index	0	1	2	3	4
values	10	12	20	25	30

Example 1

Input:

values[ ] = {10, 12, 20, 25, 30}

target = 25

Output:

3

Example 2

Input:

values[ ] = {10, 12, 20, 25, 30}

target = 50

Output:

-1 (not found)

3. What is Binary Search? Discuss with example.

- Binary Search also called **half-interval search** algorithm.
- It finds the position of a search element with a **sorted array**.
- It can be done as **divide and conquer** search algorithm .

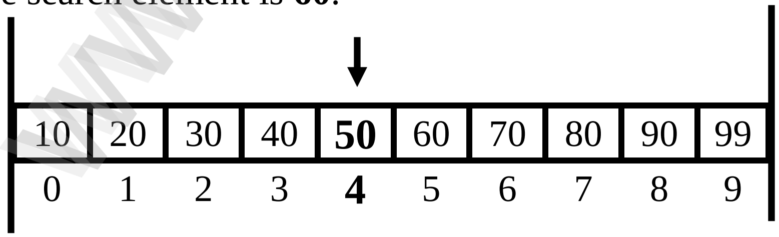
Pseudo code

```
1. Start with the middle element:
        middle value = number of elements in array/2.
2. If not, then compare the middle element with the search value,
3. If the search element is greater than the number in the middle index, then
   select the elements to the right side of the middle index, and go to step 1.
4. If the search element is less than the number in the middle index, then
   select the elements to the left side of the middle index, and start with
   step1.
5.  when a match is found, display success message with the index of the
   element matched.
6. If no match is found for all comparisons, then display unsuccessful
   message.
```

Example

- List of elements in an array must be **sorted first** for Binary Search.
- The search element is **60**.

60>50

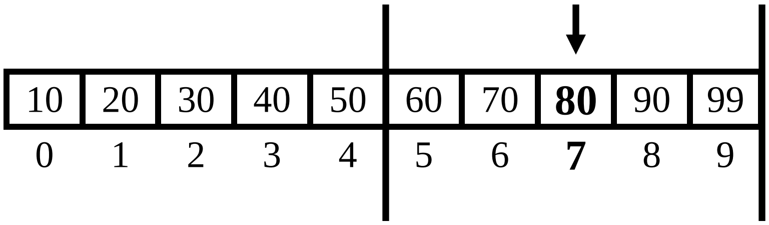


10	20	30	40	**50**	60	70	80	90	99
0	1	2	3	**4**	5	6	7	8	9

$$\boxed{\textbf{mid= low + (high – low)/2}}$$

- Here it is, 0 + (9-0)/2 = **4.5** (fractional part ignored)

60<80



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

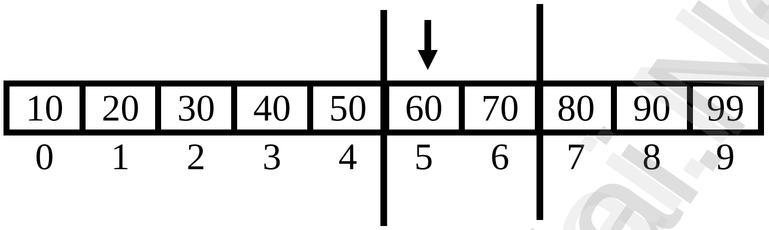
low = mid + 1 // low=5

mid= low + (high – low)/2

mid =5 + (9-5)/2 = **7 is a mid value**

Which is not match with search element.

60=60

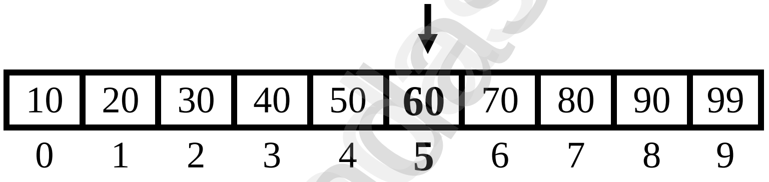


10	20	30	40	50	60	70	80	90	99
0	1	2	3	4	5	6	7	8	9

high = mid - 1 // high =6

mid= low + (high – low)/2

mid = 5 + (6 – 5)/2 = **5.5** (fractional part ignored)



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

- The search element **60** is found at location or index **5**.

4. Explain the Bubble sort algorithm with example.

- Bubble sort is a **simple sorting algorithm**.
- It compares each **pair of adjacent elements** and **swaps them** If they are in the **unsorted order**.
- This **comparison and passed** to be continued **until no swaps are needed**.

Pseudo code

1. Start with the first element **index = 0**, compare the **current element** with the **next element** of the array.

2. If the **current element is greater than** the **next element** of the array, **swap them.**

3. If the **current element is less than** the **next or right side of the element**, **move to the next** element. **Go to step 1** and repeat until **end of the index in reached**.

Example

An array with values {15, 11, 16, 12, 14, 13}

15 > 11

So interchange

15 < 16

No swapping

16 > 12

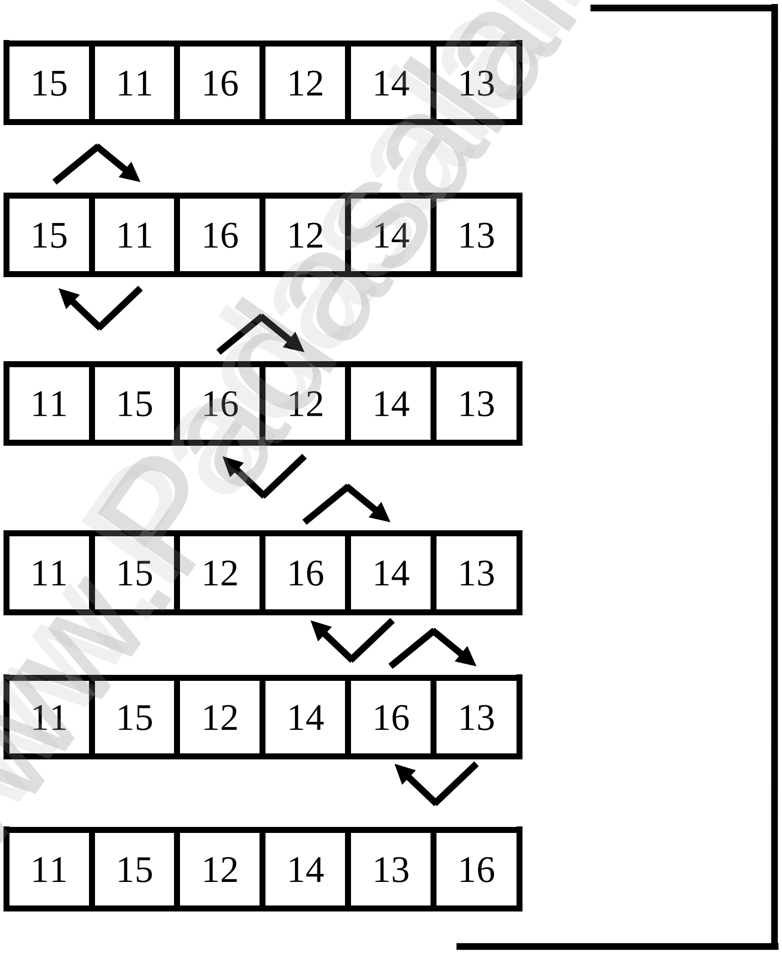
So interchange

16 > 14

So interchange

16 > 13

So interchange



- The final iteration will give the **sorted array.**
- At the end of **all the iterations we will get** the sorted values in an array

11	12	13	14	15	16

5. Explain the concept of Dynamic programming with suitable Example.

- Dynamic programming is an **algorithmic design method** is used when the **solution to a problem** can be **viewed** as the **result** of a **sequence of decisions.**
- Dynamic programming approach is similar to **divide and conquer.** The problem can be **divided into smaller sub-problems**.
- **Results** can be **re-used** to **complete the process**.
- Dynamic programming approaches are used to **find the solution in optimized way.**

Steps to do Dynamic Programming

- The given problem will be **divided into smaller** overlapping **sub-problems**.
- An **optimum solution** for the **given problem** can be **achieved** by using **result** of **smaller sub-problem.**
- Dynamic algorithms uses **Memoization**.

Example: Fibonacci Iterative Algorithm

- Fibonacci series Initialize **f0 = 0, f1 = 1**.

Step 1: Print the initial values of Fibonacci **f0 and f1**	**// f0,f1 = 1**
Step 2: Calculate Fibonacci **fib ← f0 + f1**	**// fib =2**
Step 3: Assign **f0 ←f1, f1←fib**	**// f0 = 1 , f1 = 2**
Step 4: Print the next consecutive value of Fibonacci **fib**	**// fib = 3**

Step 5: Go to step 2 and **repeat** until the **specified number of terms generated**.

Example: If we generate Fibonacci series **upto10 digits**,

The Fibonacci series is : **0 1 1 2 3 5 8 13 21 34 55**

5. PYTHON – VARIABLES AND OPERATORS

2 Marks

1. What are the different modes that can be used to test Python Program?

- □ Interactive mode
- □ Script mode

2.Write short notes on Tokens.

Python breaks each logical line into a **sequence of elementary lexical components** known as Tokens.

1. Identifiers
2. Keywords
3. Operators
4. Delimiters
5. Literals

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

Operators are special symbols which represent **computations, conditional matching** in programming.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Conditional Operators

4. What is a literal? Explain the types of literals?

- ○ Literal is a **raw data** given in a **variable or constant**.
  1. Numeric Literals – consists of **digits** and are immutable.
  2. String Literals – sequence of **characters** surrounded by **quotes.**
  3. Boolean Literals – any of two values **True or False**.

5. Write short notes on Exponent data?

An Exponent data contains decimal digit part, decimal point, exponent part followed by **one or more digits.**

Example: 12.E04, 24.e04

3 Marks

1. Write short notes on Arithmetic operator with examples.

An arithmetic operator is a **mathematical operator** used for simple arithmetic. It takes two operands and performs a calculation on them.

a=100, b= 10	Example	Output
+ (Addition)	>>> a+b	110
- (Subtraction)	>>>a-b	90
*(Multiplication)	>>>a*b	1000
/(Division)	>>>a/b	10.0
%(Modulus)	>>>a%30	10
**(Exponent)	>>>a**2	10000
// (Floor Division)	>>>a//30 (Integer Division)	3

2. What are the assignment operators that can be used in Python?

- '=' is a simple assignment operator to assign **values to variable**.
- There are various **compound operators** in Python like +=, -=, *=, /=, %=, **=, and //=.

Example

```
a = 5
a, b = 5, 10
a+=2
```

3. Explain Ternary operator with examples.

Ternary operator is also known as **conditional operator** that evaluates something based on a condition being **true or false.**

Syntax

variable_name = [on_true] if[Test expression] else [on_false]

Example

```
min = 49 if 49<50 else 50
```

Output

```
49
```

4. Write short notes on Escape sequences with examples.

In Python strings, the **backslash** “\” is a special character, also called the “escape” character. It is used in representing certain whitespace characters.

Escape Sequence	Example	Output
\\ (Backslash)	>>>print(“\\test”)	\test
\’ (Single-quote)	>>>print(“Doesn\’t”)	Doesn’t
\”(Double-quote)	>>>print(“\”Python\””)	“Python”

5. What are string literals? Explain.

- In Python a string literal is a sequence of characters surrounded by **quotes**.
- Python supports **single, double and triple quotes** for a string.
- A character literal is a **single character** surrounded by single or double quotes.
- The value with triple-quote ‘‘‘ ’’’ is used to give **multi-line string** literal.

Example

```
strings = “This is a Python”
char = “C”
multiline = ‘‘‘ This is a multiline sting with more than one line code.’’’
print(strings)
print(char)
print(multiline)
```

Output

```
This is Python
C
This is a multiline string with more than one line code.
```

5 Marks

1. Describe in detail the procedure Script mode programming.

- A script is a **text file** containing the Python statements.
- Once the Python Scripts is **created**, they are **reusable**.
- The Scripts are **editable**.

Creating Scripts in Python

1. Choose **File** → **New File** or press **Ctrl+N**
2. An **untitled** blank script text editor will be displayed on screen.
3. Type the **code** in script editor

```
a=100

b=350

c=a+b

print(“The Sum=”, c)
```

Saving Python Script

1. Choose **File** → **Save** or press **Ctrl + S**
2. Now, **Save As** dialog box appears on the screen.
   - Select the **location** to save your python code.
   - Type the **file name** in File name box.
   - Python file are by **default** saved with extension **.py**
3. Click **Save** button save your Python script.

Executing Python Script

1. Choose **Run** → **Run Module** or Press **F5**
2. If your code has any error, it will be shown in **red color** in the IDLE window, and Python describes the **type of error occurred**.
3. To correct the errors, save the file and **execute** it again.
4. The **Output** will appear in the IDLE window of Python.

```
The Sum= 450
```

2. Explain input( ) and print( ) functions with examples.

Input and Output Functions

A **program** needs to interact with the **user to accomplish the desired task**; this is called Input-Output Functions.

1. Input( ) Function

- The **input( )** function helps to **enter data at runtime** by the user.
- **"Prompt String"** in the syntax is a **message to the user**.
- It is displayed on the **Monitor**.
- The **input( )** takes **typed data** from the **keyboard.**

Syntax

variable = input("prompt string")

Example

```
x = int(input("Number 1:"))

y = int(input("Number 2:"))

print("Total = ", x+y)
```

Output

```
Number 1: 10

Number 2: 20

Total = 30
```

2. Print( ) Function

- The **print( )** function is used to **display result on the screen**.
- The **print( )** function **displays an entire statement**.
- **Comma** is used to **print more than one item**.

Syntax

```
print(“string”)
print(variable)
print(“string”, variable)
print(“string1”, variable1, “string2” , variable2)
```

Example 1

```
x=10
y=20
z=x+y
print(z)
```

Output

```
30
```

Example 2

```
print(“Welcome to Python”)
```

Output

```
Welcome to Python
```

3. Discuss in detail about Tokens in Python.

Tokens

Python breaks each logical line into a **sequence of elementary lexical components** known as **Tokens**.

Types of Tokens

1. Identifiers
2. Keywords
3. Operators
4. Delimiters
5. Literals

1. Identifiers

- An identifier must start with an **Alphabet** (A to Z or a to z) or **Underscore** (_)
- Identifiers may contain **digits** (0 to 9)
- Identifiers are **case sensitive**.
- Identifiers does **not allow Special character**.
- Identifiers does not allow **Python keyword**.

Identifiers Example:

sum, total_marks, num1

2. Keywords

- Keywords are **special meaning for Interpreter**.
- To recognize the **Structure of the Program**.
- They **cannot be used for any other purpose**.

Keywords Example:

true, false, break, continue etc.,

3. Operators

- Operators are **special symbols** which represent **computations, conditional matching** in Programming.
- Operators are **Arithmetic, Relational, Logical, Assignment and Conditional**

Operators Example:

```
a=100

b=10

print(a+b)

print(a>b)
```

Output

```
110

True
```

4. Delimiters

- Python uses the **symbols** and **symbol combinations** as **delimiters**.
- It is include **expressions, lists, dictionaries and strings**.

Delimiters Example:

(	)	[	]	{	}

5. Literals

- Literal is a raw data given in a **variable or constant**.

Types of Literals

1. Numeric Literals – consists of **digits** and are immutable.
2. String Literals – sequence of **characters** surrounded by **quotes.**
3. Boolean Literals – any of two values **True or False**.

6. CONTROL STRUCTURES

2 Marks

1. List the control structures in Python.

- Sequential
- Alternative or Branching
- Iterative or Looping

2. Write note on 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.

Syntax: break

3. Write is the <u>syntax</u> of if..else statement.

Syntax:

if <condition>:

statements-block 1

else:

statements-block 2

4. Define control structure.

- A program statement that causes a jump of **control** from one part of the program to another is called control structure or control statement.

5. Write note on range( ) in loop.

- range( ) generates a list of values starting from **start** till **stop-1** in for loop.

Syntax:

range(start, stop,[step])

3 Marks

1. Write a program to display

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

Ans.:

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()
```

2. Write note on if..else structure.

- The if..else statement provides control to check the **true block** as well as the **false block**.
- if..else statement thus provides two possibilities and the **condition** determines which BLOCK is to be executed.

Syntax:

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

3. Using if..else..elif statement write a suitable program to display largest of 3 numbers.

CODE:

```
n1=int(input("Enter the first number: "))

n2=int(input("Enter the second number: "))

n3=int(input("Enter the third number: "))

if(n1>=n2)and(n1>=n3):

  biggest=n1

elif(n2>=n1)and(n2>=n3):

  biggest=n2

else:

  biggest=n3

print("The biggest number between",  n1, "," ,n2, "and" ,n3, "is" ,biggest)
```

OUTPUT:

```
Enter the first number: 1

Enter the second number: 3

Enter the third number: 5

The biggest number between 1 , 3 and 5 is 5
```

4. Write the syntax of while loop.

Syntax:

```
while <condition>:
        statements block 1
[else:
        statements block 2]
```

5. List the differences between break and continue statements.

break	continue
The break statement terminates the loop containing it.	The continue statement is used to skip the remaining part of a loop.
Control of the program flows to the statement immediately after the body of the loop.	Control of the program flows start with next iteration.
Syntax: break	**Syntax:** continue

5 Marks

1. Write a detail note on for loop.

- for loop is the most **comfortable** loop. It is also an **entry** check loop.
- The condition is checked in the beginning and the body of the loop is executed if it is only true otherwise the loop is not executed.

Syntax:

```
for  counter_variable in sequence:
        statements-block 1
[else:
        statements-block 2]
```

- The counter_variable is the control variable.
- The sequence refers to the initial, final and increment value.
- for loop uses the range() function in the sequence to specify the initial, final and increment values.
- range() generates a list of values starting from **start** till **stop-1.**

Syntax:

range(start, stop, [step])

start – refers to the initial value
stop – refers to the final value
step – refers to increment value, this is optional part

Example:

```
for i in range(2,10,2):
   print(i, end=" ")
else:
   print("\n End of the loop")
```

Output:

```
2 4 6 8
End of the loop
```

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

- To construct a chain of if statements 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 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
```

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:
   print("Grade:E")
```

OUTPUT:

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

3. Write a program to display all 3 digit odd numbers.

CODE:

```
n1=int(input("Enter the First Number: "))
n2=int(input("Enter the Last Number: "))
for i in range(n1,n2+1):
    if(i%2!=0):
        print(i, end=" ")
```

OUTPUT:

```
Enter the First Number: 100
Enter the Last Number: 110
101 103 105 107 109
```

4. Write a program to display multiplication table for a given number.

CODE:

```
n1=int(input("Display Multiplication Table of  "))
for i in range(1,11):
    print(i, 'x', n1, '=',  n1*i)
```

OUTPUT:

```
Display Multiplication Table of  2
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
>>>
```