**WEEK NO**:01

**DATE**:05 December ,2025

**SUBJECT**: Python Programming Lab

**1.QUESTION**

To write a python script to illustrate data types (int, float, char, string).

**PROCEDURE:**

- Initialize variables with different literal values (integer, float, string).
- Use the print() function to display the values.
- Use the built-in type() function to identify and print the data class of each variable.

**PROGRAM:**

```
a=25
b=12.74
c='A'
p='Hello'
print("Integer value:",a)
print(type(a))
print("Float value:",b)
print(type(b))
print("character:",c)
print(type(c))
print("string:",p)
print(type(p))
```

**OUTPUT:**

```
Integer value: 25
<class 'int'>
Float value: 12.74
<class 'float'>
Character: A
<class 'str'>
String value: Hello
```

<class 'str'>

**INFERENCES/CONCLUSION:**

- The type() function outputs the class name in the format <Class ‘int’>.
- Python automatically detects the data type based on the value assigned(e.g : 25 as int ,12.75 as float).

**2.QUESTION**

To write a python program to perform the following expressions using operator precedence.

(1) 5+3*2

(2) 2*3**2

(3)2**3**2

(4)(2**3)**2

**PROCEDURE:**

- Mathematical expressions follows BODMAS rules.
- In python ** (exponentiation) has higher precedence than * (multiplication).
- Exponentiation is right-associative (calculated from right to left).

**PROGRAM:**

```
print(5 + 3 * 2)

print(2 * 3 ** 2)    # a ** b = a^b

print(2 ** 3 ** 2)

print((2 ** 3) ** 2)
```

**OUTPUT:**

11

18

512

64

**INFERENCE/CONCLUSION:**

- Exponent has highest priority.
- Using brackets () forces the operation inside to happen first, overriding standard precedence.

**3.QUESTION**

To write a python program to illustrate type conversion functions**.**

**PROCEDURE:**

- ❖ Use float() to convert an integer to a decimal.
- ❖ Use int() to truncate a float or convert a numeric string to an integer.
- ❖ Use str() to convert a number to text, allowing it to be concatenated (joined) with other strings using the + operator.

**PROGRAM:**

```
a = 10

b = 12.5

print("Float value of a:", float(a))

print("Integer value of b:", int(b))

str_val = "45"

print("String converted to int:", int(str_val))

print("String concatenation " + str(a))
```

**OUTPUT:**

Float value of a: 10.0

Integer value of b: 12

String converted to int: 45

String concatenation 45

**INFERENCES/CONCLUSION:**

- Truncation means converting float to int by removing the decimal part .
- You cannot add an integer to string directly. You must use str() to convert the number before concatenation.

**4.Question**

To write a python program to illustrate pi, sqrt, cos, sin functions of math module.

**PROCEDURE:**

- ❖ Import the math library using an alias m.
- ❖ Access the constant pi.
- ❖ Call sqrt() for square roots and trigonometric functions. Note: Math module trig functions expect arguments in radians, not degrees.

**PROGRAM:**

```
import math as m

print("Value of pi:", m.pi)

print("Square root of 5:", m.sqrt(5))

print("Value of cos 0 degree:", m.cos(0))

print("Value of sin 0 degree:", m.sin(0))
```

**OUTPUT:**

Value of pi: 3.1415926535...

Square root of 5: 2.236...

Value of cos 0 degree: 1.0

Value of sin 0 degree: 0.0

**INFERENCES/CONCLUSION:**

- You must import math before using pi or sqrt.
- Functions like sin() and cos() expect values in radians . To use degrees, you must convert them using math.radians().

**WEEK NO**:02

**DATE**:12 December ,2025

**SUBJECT**: Python Programming Lab

**1.QUESTION**

To write a program to calculate simple intrest.

**PROCEDURE:**

- Accept user input and convert to float or int.
- Apply the formula SI = P * T * R/100.

**PROGRAM:**

```
P = float(input("Enter principle value: "))

t = float(input("Enter time: "))

r = int(input("Enter rate of interest: "))

SI = P * t * r / 100

print("Simple Interest is:", SI)
```

**OUTPUT:**

Enter principle value: 1000.25

Enter time: 4.2

Enter rate of interest: 10

Simple Interest is: 420.105

**INFERENCES/CONCLUSION:**

- Using float(input()) is necessary to handle decimal values for principal and time.
- If rate is entered as float but converted to int – loss of precision.
- If we use very large values the new may get floating point precision issues.

**2.QUESTION**

To write a python program to calculate compound interest.

**PROCEDURE:**

- Collect Principal, Rate, and Time.
- Apply the formula A = P(1 + r/100)^t and subtract the principal to find interest.

**PROGRAM:**

```
P = float(input("Enter principle amount: "))
r = int(input("Enter rate of interest: "))
t = float(input("Enter time: "))
CI = P * (1 + r / 100) ** t - P
print("Compound Interest:", CI)
```

**OUTPUT:**

Enter principle amount: 1028.63

Enter rate of interest: 12

Enter time: 3.5

Compound interest: 500.7741549

**INFERENCES/CONCLUSION:**

- For compound interest, the power operator ** must be used. Using ^ will result in a bitwise error.
- If we didn't use exponent then it may get wrong calculation.

**3.QUESTION**

To write a python program to print ASCII value of a character.

**PROCEDURE:**

- Use ord() to convert a single character into its integer ASCII code.
- Use chr() to convert an integer ASCII code back into its character representation.

**PROGRAM:**

```
ch = input("Enter a character: ")
v = int(input("Enter a value: "))
print("ASCII value of a character:", ord(ch))
print("ASCII character for a value:", chr(v))
```

**OUTPUT:**

Enter a character: A

Enter a value: 70

ASCII value of a character: 65

ASCII value of a value: F

**INFERENCES/CONCLUSION:**

- If user enter more than one character then it cause TypeError:ord() expected a character , but string of length>1 found.
- If value is out of ASCII range then we get error called valueError:chr() arg not in range.

**4.QUESTION**

To write a python program to find the area of a circle.

**PROCUDEURE:**

- Use pi*r**2 formula for finding the area of circle.
- Take input radius from user.

**PROGRAM:**

```
Pi=3.1415

R=float(input("Enter radius value:"))

a=pi*r**2

Print("Area of circle:",a)
```

**OUTPUT:**

Enter radius value: 4.2

Area of circle: 55.41606

**INFERENCES/CONCLUSION:**

- Using r^2 instead of r**2 gives Error: Gives incorrect result(bitwise xor).
- Negative radius gives logically incorrect value.

**5.QUESTION**

To write a python program to find the area of triangle.

**PROCEDURE:**

- To calculate area of triangle we should use the formula, ½*b*h.
- We should take inputs like breadth and height from the user.

**PROGRAM:**

```
b=float(input("Enter breadth value:"))

h=float(input("Enter height value:"))

a=1/2*b*h

print("Area of triangle:",a)
```

**OUTPUT:**

Enter breadth value: 4.0

Enter height value: 5.2

Area of triangle: 10.4

**INFERENCES/CONCLUSION:**

- If the input is non- numeric then it may give valueError: could not convert string to float.
- We can take values of breadth and height as either integer or float values.

**6.QUESTION**

To write a python program to perform string concatenation.

**PROCEDURE:**

- Take two strings from user or you can directly give two strings.
- By using '+' operator we can concatenate two strings.

**PROGRAM:**

a="Hello"

b=" World"

print("Add two strings:",a+b)

**OUTPUT:**

Add two strings: Hello World

**INFERENCE/CONCLUSION:**

- We can use either " or '.
- If one variable is not a string then we get Error:TypeError: Can only concatenate str( not "int" to str).

**WEEK NO**:03

**DATE**:19 December ,2025

**SUBJECT**: Python Programming Lab

**1.QUESTION**

To write a python program to work with 1D array operations including indexing and slicing.

**PROCEDURE:**

- Use import numpy as np to bring in the numerical python library.
- Use the np.array() function to convert a Python list into a NumPy array object.
- Access the first element using [0].
- Use the format [start:stop]. Note that the stop index is **exclusive** (it stops just before that number). For example, [1:4] gives elements at positions 1, 2, and 3.
- Use [-1] to quickly access the final element of the array without knowing its length.

**PROGRAM:**

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])

print("Original array:", arr)

print("First element:", arr[0])

print("Last element:", arr[-1])

print("Elements from index 1 to 3:", arr[1:4])

print("First three elements:", arr[:3])
```

**OUTPUT:**

Original array: [10 20 30 40 50]

First element: 10

Last element: 50

Elements from index 1 to 3: [20 30 40]

First three element: [10 20 30]

**INFERENCES/CONCLUSION:**

- Missing bracket means we get syntax error.
- If we given arr[10] means invalid index( index error).
- If numpy is not installed we get ModuleNotFound Error.

**2.QUESTION**

To write a python program to work with 2D array operations.

**PROCEDURE:**

- ❖ Define a list of lists inside np.array() to create rows and columns.
- ❖ Access a specific value using the format matrix[row, column]. Remember, counting starts at 0.
- ❖ Use a colon : to select "all":
- matrix[1, :] means "Row 1, all columns."
- matrix[:, 2] means "All rows, Column 2."

**PROGRAM:**

```
import numpy as np

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print("Matrix:\n", matrix)

print("Element at (1, 2):", matrix[1, 2])

print("Second row:", matrix[1, :])

print("Third column:", matrix[:, 2])

print("Sub-matrix (rows 0-1, cols 0-1):\n", matrix[0:2, 0:2])
```

**OUTPUT:**

Matrix:

```
[[1 2 3]
 [4 5 6]
 [7 8  9]]
```

Element at (1,2): 2

Second row: [4 5 6]

Third column: [3 6 9]

Sub-matrix(row 0-1, cols 0-1):

```
[[1 2]
 [4 5]]
```

**INFERENCES/CONCLUSION:**

- Matrices are created using numpy arryas.
- Format for accessing: matrix [row, column].

**WEEK NO**:04

**DATE**:26 December ,2025

**SUBJECT**: Python Programming Lab

**1.QUESTION**

To write python program to find the power of a number without using built-in functions.

**PROCEDURE:**

- Take the base and the exponent from the user as integers.
- Set a variable result to **1**. (Multiplicative identity).
- Use a for loop that runs exactly exponent times.
- In each loop, update the result by multiplying it by the base (result = result ^ base).
- Display the final accumulated result.

**PROGRAM:**

```
base = int(input("Enter the base of number: "))

exponent = int(input("Enter the exponent: "))

result = 1

for i in range(exponent):

    result = result * base

print("Result:", result)
```

**OUTPUT:**

Enter the base number: 2

Enter the exponent: 3

Result: 8

**INFERENCES/CONCLUSION:**

- Power is computed using repeated multiplication.
- If we give string instead of number we get valueError.

**2.QUESTION**

To write a python program to count the number of even and odd numbers upto the given range.

**PROCEDURE:**

- Ask the user how many numbers they want to check (n).
- Initialize even = 0 and odd = 0 to keep track of the totals.
- Use for i in range(n) to prompt the user to enter a number (n) times.

- ❖ Use the **modulo operator** (%).
- If num % 2 == 0, the number is even; increment the even counter.
- Otherwise, increment the odd counter.
- ❖ Print the final counts for both categories.

**PROGRAM:**

```
n = int(input("Enter number of elements: "))
even = 0
odd = 0
for i in range(n):
    num = int(input("Enter number: "))
    if num % 2 == 0:
        even += 1
    else:
        odd += 1
print("Even numbers count:", even)
print("Odd numbers count:", odd)
```

**OUTPUT:**

Enter number of elements: 3

Enter number: 8

Enter number: 9

Enter number: 3

Even numbers count: 1

Odd numbers count: 2

**INFERENCES/CONCLUSION:**

- Loop checks each number using % operator by which we can see whether the given number is even or odd.
- If we miss colon(: ) after if condition then we get syntaxError: expected ':'
- If no space/tab is used for even/odd we get Identation Error.

**3.QUESTON**

To write a python program to print the multiplication table for a given number.

**PROCEDURE:**

- Ask the user which number they want a table for (n).
- Use range(1, 11) to iterate through numbers 1 to 10.
- Inside the loop, multiply n by the current loop index i.
- Print the result in a readable format using string formatting.

**PROGRAM:**

```
n = int(input("Enter a number: "))

for i in range(1, 11):

    print(n, "x", i, "=", n * i)
```

**OUTPUT:**

Enter a number: 3

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

3 x 10 = 30

**INFERENCES/CONCLUSION:**

- Works for positive, zero, negative numbers.
- We should give range from (1,11) not (1,10) because ending value is not incremented.
- If range is given as (0,11) then it will start printing from 0 so we should start with 1.

**4.QUESTION**

To write a python program to display minimum and maximum among three numbers.

**PROCEDURE:**

- Start by picking the first number (a) and storing it in a variable called max_num (and another for min_num).
- The program checks the second number (b). If b is larger than our current max_num, the max_num variable is updated to hold b.

- Then, the program checks the third number (c). If c is larger than the current max_num (regardless of whether it was a or b), it updates again.

❖ The exact same logic is followed for the minimum, but we check if the next number is **smaller** (<) than our current stored min_num.

**PROGRAM:**

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))
max_num = a  # Assume the first number is the maximum
if b > max_num:
   max_num = b
if c > max_num:
   max_num = c
min_num = a  # Assume the first number is the minimum
if b < min_num:
   min_num = b
if c < min_num:
   min_num = c
print("Maximum:", max_num)
print("Minimum:", min_num)
```

**OUTPUT:**

Enter first number: 50

Enter second number: 100

Enter third number: 75

Maximum: 100

Minimum: 50

**INFERENCES/CONCLUSION:**

- If we give string instead of number it gives valueError:invalid literal for int().
- If all three numbers are the same (e.g., 5, 5, 5), the if conditions (b > max_num) will always be false. The program will correctly output 5 as both the max and min.

**WEEK NO**:05

**DATE**:02 January,2025

**SUBJECT**: Python Programming Lab

**1.QUESTION**

To write a python program to find if a number is prime or not with and without recursion.

**PROCEDURE:**

- A function is_prime(n) is created to handle the logic.
- It first checks if the number is less than 2 (since 0 and 1 are not prime).
- A for loop iterates through every integer $i$ starting from 2 up to $n-1$.
- Inside the loop, the modulo operator % checks if the remainder is 0 ($n \pmod i = 0$). If it is, the number has a factor other than 1 and itself, so it returns False.
- If the loop finishes without finding any factors, the function returns True.
- The main program takes an integer input and calls the function to display the final result.

**PROGRAM: Without recursion**

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
num = int(input("Enter a number: "))
if is_prime(num):
    print("Prime number!")
else:
    print("Not a prime number!")
```

**OUTPUT:**

Enter a number: 7

Prime number!

**INFERENCES/CONCLUSION:**

- In python , if we use true or false first letter must be capital(True/False) otherwise it cause NameError.
- The return True statement must be placed **outside** the for-loop. If it is inside the loop, the function will stop after checking only the number 2 and give incorrect results for odd numbers like 9.

**PROGRAM: With recursion**

```
def is_prime_recursive(n, divisor=2):
    if n < 2:
        return False
    if n == divisor:
        return True
    if n % divisor == 0:
        return False
    return is_prime_recursive(n, divisor + 1)
num = int(input("Enter a number: "))
if is_prime_recursive(num):
    print("Prime number!")
else:
    print("Not a prime number!")
```

**OUTPUT:**

Enter a number: 15

Not a prime number!

**2.QUESTION**

To write a python program to display Fibonacci series using iteration and recursion.

**PROCEDURE:**

- The program asks the user for the number of terms (n) they want to see.
- Two variables, a and b, are set to the starting values of the series, **0** and **1**.
- A for loop runs exactly n times.
- In each iteration, the current value of a is printed. The end=" " part ensures the numbers are printed on the same line with a space and a, b = b, a + b it is used for updating both variables.

**PROGRAM: Iterative**

```
n = int(input("Enter number of terms: "))
a, b = 0, 1
print("Fibonacci series:")
for i in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

**OUTPUT:**

Enter number of terms: 6

Fibonacci series:

0 1 1 2 3 5

**INFERENCES/CONCLUSION:**

- Using end=" " is crucial here to display the series as a single horizontal line.
- In other languages, you might need a "temp" variable to update these values. But in Python a, b = b, a + b handles the swap and addition simultaneously without needing a third variable.

**PROGRAM: Recursive**

```
def fibonacci_recursive(n):
    if n <= 1:
        return n
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
terms = int(input("Enter number of terms: "))
if terms <= 0:
    print("Please enter a positive integer")
else:
    print("Fibonacci series:")
    for i in range(terms):
        print(fibonacci_recursive(i), end=" ")
```

**OUTPUT:**

Enter number of terms: 4

Fibonacci series:

0 1 1 2

**3.QUESTION**

To write a python program to find the factorial of a number with and without recursion.

**PROCEDURE:**

- The program takes an integer input from the user and stores it in variable n.
- Inside the function, a variable result is set to 1.
- A for loop runs from 1 up to (and including) n.
- In each cycle, the current value of result is multiplied by the loop variable i ($result = result \times i$).
- Once the loop completes, the final product is returned and printed.

**PROGRAM: without Recursion**

```
n=int(input("Enter a number:"))
def fact(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
print("Result:",fact(n))
```

**OUTPUT:**

Enter a number:5

Result: 120

**PROGRAM: With recursion**

```
n = int(input("Enter a number: "))
def fact_rec(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * fact_rec(n - 1)
print("Result:", fact_rec(n))
```

**OUTPUT:**

Enter a number: 6

Result: 720

**INFERENCES/CONCLUSION:**

- It can handle very large numbers without crashing the program.
- If you enter 0, the loop range(1, 1) doesn't execute, so it correctly returns the initial result of 1.