

Lab Practical File
For
Computer Programming Basic with Python (CPBP)
by
Name: Sahil Sandal
Roll No: 24901320
Submitted to
Dr. Diksha
Centre for Artificial Intelligence



INDEX

<u>S. NO.</u>	<u>EXPERIMENT</u>	<u>DATE OF SUBMISSION</u>	<u>PAGE NO.</u>	<u>REMARKS</u>
1.	Install Python and write basic programs to explore its syntax and functionality	12 AUG 2024	3	
2.	Demonstrate python operators and develop code for given problem statements	19 AUG 2024	5	
3.	Demonstrate conditional and loop statements and develop code for given problem statements	02 SEPT 2024	9	
4.	Demonstrate list operations and develop code for given problem statements	09 SEPT 2024	15	
5.	Demonstrate arrays and tuples and develop code for given problem statements	23 SEPT 2024	18	
6.	Demonstrate functions and modules and develop code for given problem statements	30 SEPT 2024	23	
7.	Demonstrate Set operations and develop code for given problem statements	14 OCT 2024	26	
8.	Demonstrate dictionary operations and develop code for given problem statements	21 OCT 2024	29	
9.	Demonstrate strings and its related operations and develop code for given problem statements	04 NOV 2024	32	
10.	Demonstrate file handling and develop code for given problem statements	11 NOV 2024	38	
11.	Classes, objects and inheritance	18 NOV 2024	40	
12.	Polymorphism, Error and Exception handling	25 NOV 2024	45	

EXPERIMENT 1

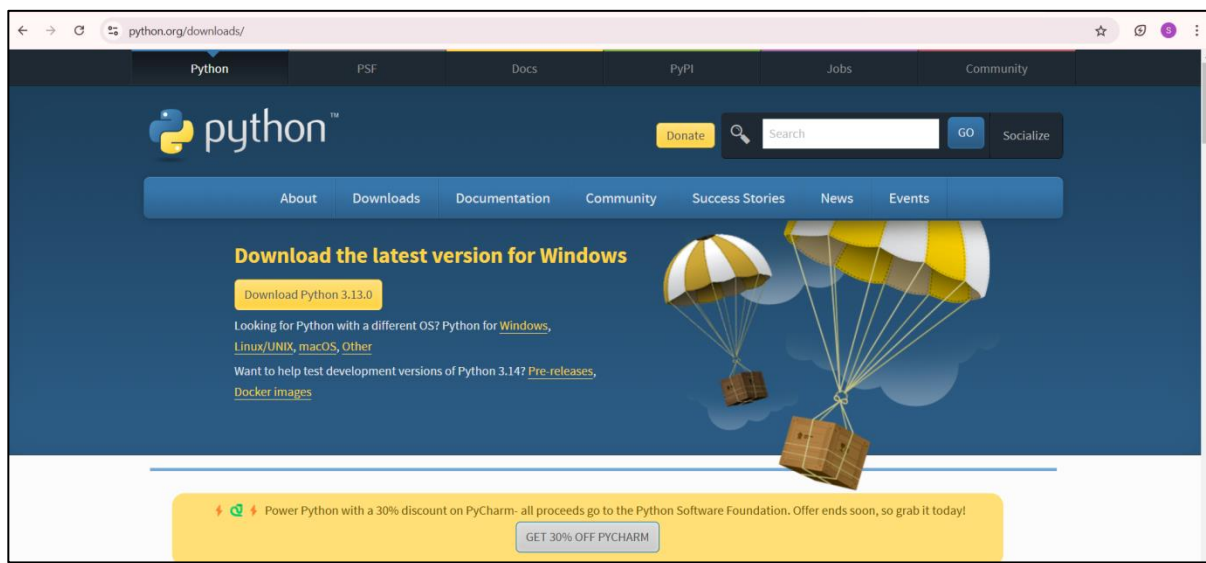
OBJECTIVE: Install Python and write basic programs to explore its syntax and functionality.

THEORY:

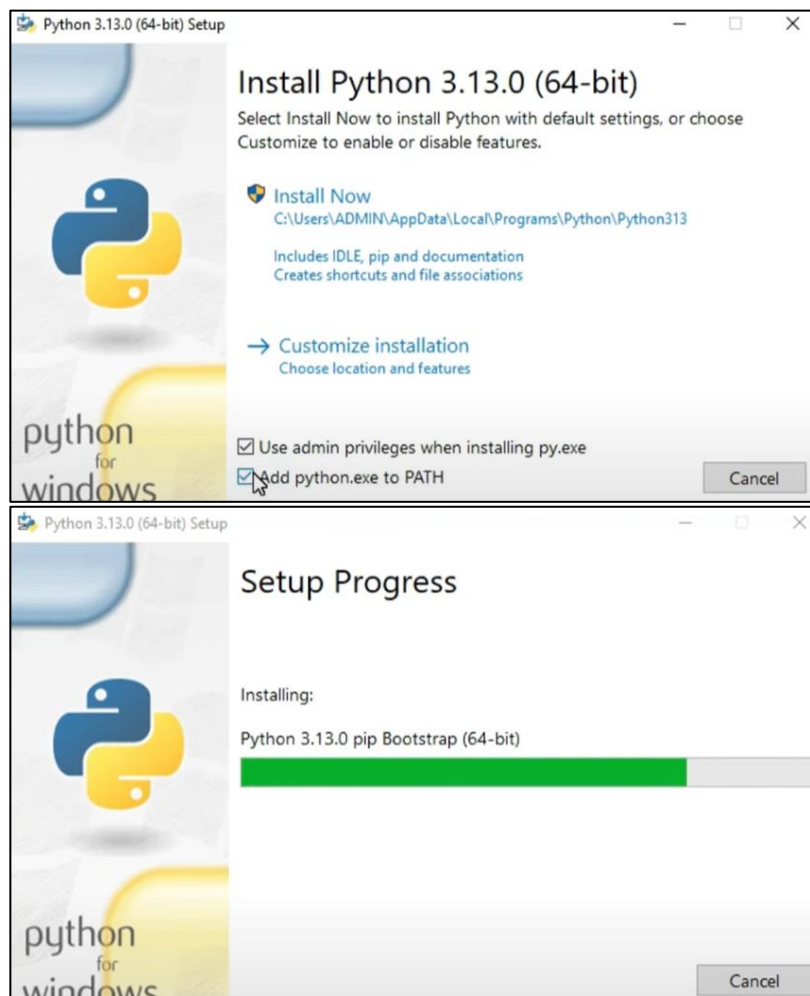
Python is a high-level, interpreted programming language created by Guido van Rossum in 1991. Known for its simplicity and readability, Python uses indentation for defining code blocks, making it beginner-friendly. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is dynamically typed and has an extensive standard library that simplifies complex tasks like file handling, data manipulation, and web development. Its versatility makes it widely used in various fields such as web development, data science, artificial intelligence, automation, and game development. Python's large community and open-source nature further enhance its adaptability and resource availability.

INSTALLATION STEPS:

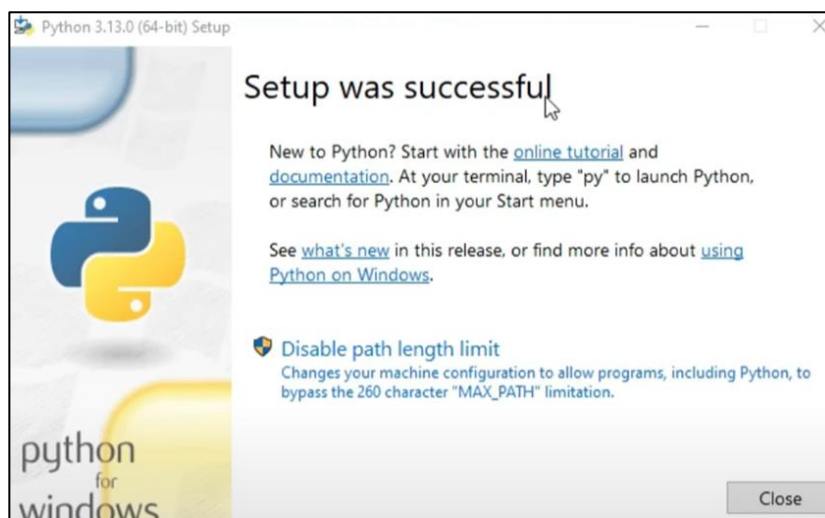
1. Download Python:
 - a. Visit the official Python website.
 - b. Go to the Downloads section and select the appropriate installer for your operating system (Windows, macOS, or Linux).



2. Install Python:
 - a. Run the downloaded installer.
 - b. During installation, check the box Add Python to PATH. This ensures you can use Python from the command line.
 - c. Click on Customize Installation if needed, but the default settings work for most users.



3. Verify Installation:



- a. Open a terminal or command prompt and type:
 - i. `python --version`

```
C:\Users\ADMIN>python --version
Python 3.13.0
```

CODE:

Simple Program to perform arithmetic operations on two numbers

a = 7

b = 2

```
print("sum: ", a+b)

print("diff: ", a-b)

print("mult: ", a*b)

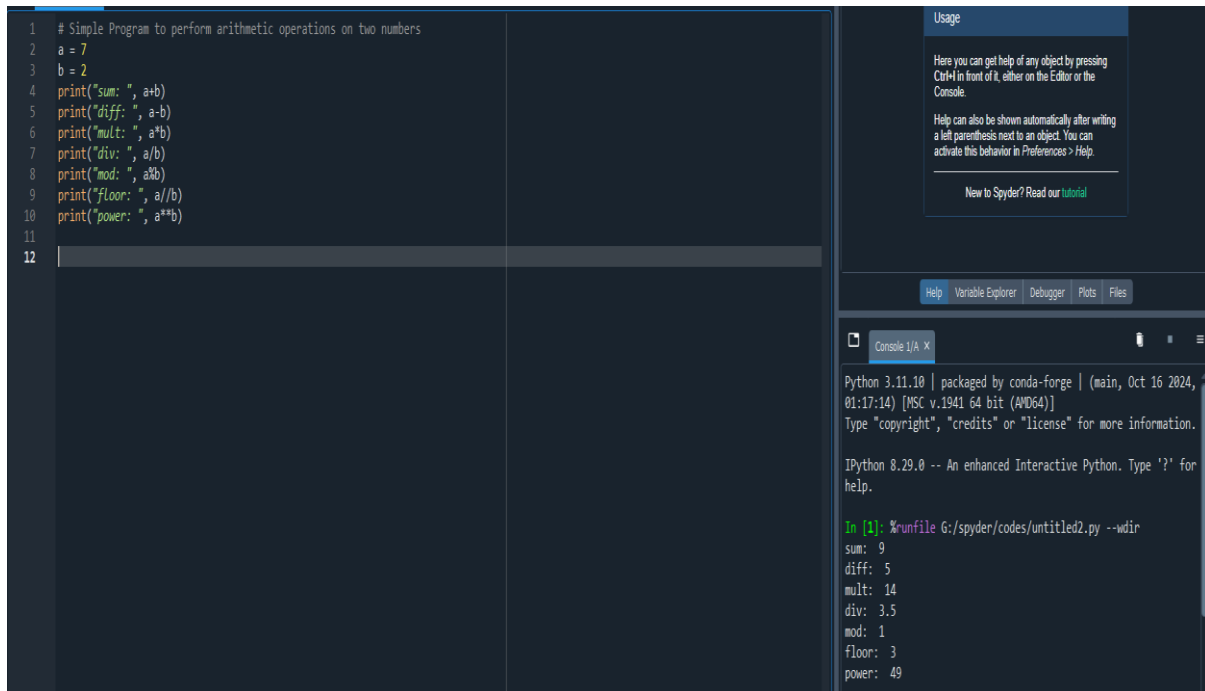
print("div: ", a/b)

print("mod: ", a%b)

print("floor: ", a//b)

print("power: ", a**b)
```

RESULTS:



```
1 # Simple Program to perform arithmetic operations on two numbers
2 a = 7
3 b = 2
4 print("sum: ", a+b)
5 print("diff: ", a-b)
6 print("mult: ", a*b)
7 print("div: ", a/b)
8 print("mod: ", a%b)
9 print("floor: ", a//b)
10 print("power: ", a**b)
11
12
```

Usage

Here you can get help of any object by pressing
Ctrl+H in front of it, either on the Editor or the
Console.

Help can also be shown automatically after writing
a left parenthesis next to an object. You can
activate this behavior in [Preferences > Help](#).

New to Spyder? Read our [tutorial](#)

Help Variable Explorer Debugger Plots Files

Console 1/A X

Python 3.11.10 | packaged by conda-forge | (main, Oct 16 2024,
01:17:14) [MSC v.1941 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.29.0 -- An enhanced Interactive Python. Type '?' for
help.

```
In [1]: %runfile G:/spyder/codes/untitled2.py --wdir
sum: 9
diff: 5
mult: 14
div: 3.5
mod: 1
floor: 3
power: 49
```

EXPERIMENT 2

OBJECTIVE: Demonstrate python operators and develop code for given problem statements:

- 1) Datatype Conversion:
 - a. convert char to int, and find octal, hex value of given value
 - b. convert string to tuple, set and list
- 2) Types of operators:
 - a. perform arithmetic operations on 2 numbers
 - b. demonstrate use of comparison, logical, identity, membership operators

THEORY:

Operators are used to perform operations on variables and values. Python divides the operators in the following groups:

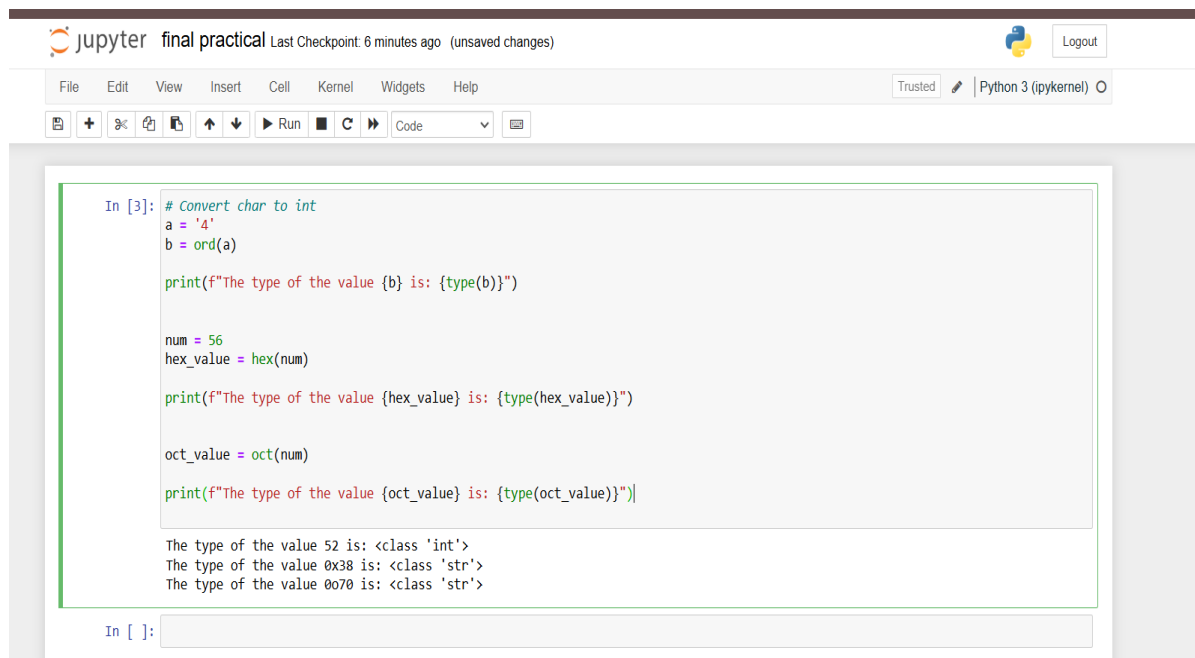
- Arithmetic operators - Arithmetic operators are used with numeric values to perform common mathematical operations
- Assignment operators - Assignment operators are used to assign values to variables
- Comparison operators - Comparison operators are used to compare two values
- Logical operators - Logical operators are used to combine conditional statements
- Identity operators - Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location
- Membership operators - Membership operators are used to test if a sequence is presented in an object
- Bitwise operators - Bitwise operators are used to compare (binary) numbers

CODE:

1. Convert char to int, and find octal, hex value of given value

```
# Convert char to int
a = '4'
b = ord(a)
print(f"The type of the value {b} is: {type(b)}")
num = 56
hex_value = hex(num)
print(f"The type of the value {hex_value} is: {type(hex_value)}")
oct_value = oct(num)
print(f"The type of the value {oct_value} is: {type(oct_value)}")
```

OUTPUT:



```
In [3]: # Convert char to int
a = '4'
b = ord(a)

print(f"The type of the value {b} is: {type(b)}")

num = 56
hex_value = hex(num)

print(f"The type of the value {hex_value} is: {type(hex_value)}")

oct_value = oct(num)

print(f"The type of the value {oct_value} is: {type(oct_value)}")

The type of the value 52 is: <class 'int'>
The type of the value 0x38 is: <class 'str'>
The type of the value 0o70 is: <class 'str'>
```

2. Convert string to tuple, set and list

Define a string

```
x = 'PythonRocks'
```

Convert string to tuple

```
y = tuple(x)
```

```
print(f"The string '{x}' converted to a tuple: {y}")
```

Convert string to set

```
y = set(x)
```


```
print(f"The string '{x}' converted to a set (unique characters only, unordered): {y}")
```

Convert string to list

```
y = list(x)
```

```
print(f"The string '{x}' converted to a list: {y}")
```

OUTPUT:



```
In [4]: # Define a string
x = 'PythonRocks'

# Convert string to tuple
y = tuple(x)
print(f"The string '{x}' converted to a tuple: {y}")

# Convert string to set
y = set(x)
print(f"The string '{x}' converted to a set (unique characters only, unordered): {y}")

# Convert string to list
y = list(x)
print(f"The string '{x}' converted to a list: {y}")

The string 'PythonRocks' converted to a tuple: ('P', 'y', 't', 'h', 'o', 'n', 'R', 'o', 'c', 'k', 's')
The string 'PythonRocks' converted to a set (unique characters only, unordered): {'o', 'k', 'n', 'R', 'c', 's', 'y', 'P', 't', 'h'}
The string 'PythonRocks' converted to a list: ['P', 'y', 't', 'h', 'o', 'n', 'R', 'o', 'c', 'k', 's']
```

3. Perform arithmetic operations on 2 numbers

```
# Define two numbers
```

```
a = 7
```

```
b = 2
```

```
# Perform arithmetic operations
```

```
print(f"The sum of {a} and {b} is: {a + b}")
```

```
print(f"The difference when {b} is subtracted from {a} is: {a - b}")
```

```
print(f"The product of {a} and {b} is: {a * b}")
```

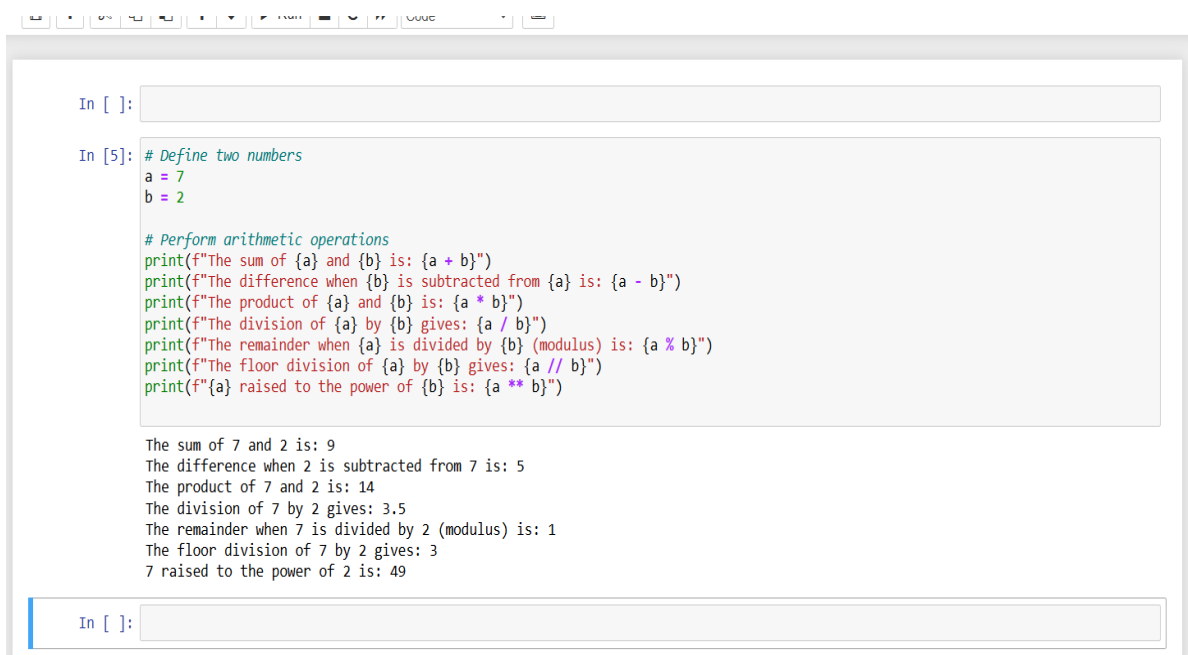
```
print(f"The division of {a} by {b} gives: {a / b}")
```

```
print(f"The remainder when {a} is divided by {b} (modulus) is: {a % b}")
```

```
print(f"The floor division of {a} by {b} gives: {a // b}")
```

```
print(f"{a} raised to the power of {b} is: {a ** b}")
```

OUTPUT:



```
In [ ]:
```

```
In [5]: # Define two numbers
a = 7
b = 2

# Perform arithmetic operations
print(f"The sum of {a} and {b} is: {a + b}")
print(f"The difference when {b} is subtracted from {a} is: {a - b}")
print(f"The product of {a} and {b} is: {a * b}")
print(f"The division of {a} by {b} gives: {a / b}")
print(f"The remainder when {a} is divided by {b} (modulus) is: {a % b}")
print(f"The floor division of {a} by {b} gives: {a // b}")
print(f"{a} raised to the power of {b} is: {a ** b}")
```

```
The sum of 7 and 2 is: 9
The difference when 2 is subtracted from 7 is: 5
The product of 7 and 2 is: 14
The division of 7 by 2 gives: 3.5
The remainder when 7 is divided by 2 (modulus) is: 1
The floor division of 7 by 2 gives: 3
7 raised to the power of 2 is: 49
```

```
In [ ]:
```

4. Demonstrate use of comparison, logical, identity, membership operators

```
# Demonstrate Comparison Operators
```

```
a = 5
```

```
b = 2
```

```
print(f"Is {a} equal to {b}? {a == b}")
```

```
print(f"Is {a} not equal to {b}? {a != b}")
```

```
print(f"Is {a} greater than {b}? {a > b}")
```

```
print(f"Is {a} less than {b}? {a < b}")
```

```
print(f"Is {a} less than or equal to {b}? {a <= b}")
```

```
print(f"Is {a} greater than or equal to {b}? {a >= b}")
```

```
# Demonstrate Logical Operators
```

```
a = 5
```

```
b = 6
```

```
print(f"Do both conditions hold: {a > 2} and {b >= 6}? {(a > 2) and (b >= 6)}")
```

```
print(f"Does at least one condition hold: {a > 2} or {b >= 6}? {(a > 2) or (b >= 6)}")
```

```
# Demonstrate Identity Operators
```

```
x1 = 5
```

```
y1 = 5
```

```
x2 = 'Hello'
```

```
y2 = 'Hello'
```

```
x3 = [1, 2, 3]
```



```

y3 = [1, 2, 3]
print(f"Are {x1} and {y1} not the same object in memory? {x1 is not y1}")
print(f"Are the strings '{x2}' and '{y2}' the same object in memory? {x2 is y2}")
print(f"Are the lists {x3} and {y3} the same object in memory? {x3 is y3}")

```

OUTPUT:

```

In [ ]:

In [6]: # Demonstrate Comparison Operators
a = 5
b = 2
print(f"Is {a} equal to {b}? {a == b}")
print(f"Is {a} not equal to {b}? {a != b}")
print(f"Is {a} greater than {b}? {a > b}")
print(f"Is {a} less than {b}? {a < b}")
print(f"Is {a} less than or equal to {b}? {a <= b}")
print(f"Is {a} greater than or equal to {b}? {a >= b}")

# Demonstrate Logical Operators
a = 5
b = 6
print(f"Do both conditions hold: {a > 2} and {b >= 6}? {(a > 2) and (b >= 6)}")
print(f"Does at least one condition hold: {a > 2} or {b >= 6}? {(a > 2) or (b >= 6)}")

# Demonstrate Identity Operators
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1, 2, 3]
y3 = [1, 2, 3]
print(f"Are {x1} and {y1} not the same object in memory? {x1 is not y1}")
print(f"Are the strings '{x2}' and '{y2}' the same object in memory? {x2 is y2}")
print(f"Are the lists {x3} and {y3} the same object in memory? {x3 is y3}")

Is 5 equal to 2? False
Is 5 not equal to 2? True
Is 5 greater than 2? True
Is 5 less than 2? False
Is 5 less than or equal to 2? False
Is 5 greater than or equal to 2? True
Do both conditions hold: True and True? True
Does at least one condition hold: True or True? True
Are 5 and 5 not the same object in memory? False
Are the strings 'Hello' and 'Hello' the same object in memory? True
Are the lists [1, 2, 3] and [1, 2, 3] the same object in memory? False

```

EXPERIMENT 3

OBJECTIVE: Demonstrate conditional and loop statements and develop code for given problem statements:

1. Conditional Statements –
 - 1) WAP to take input from a user and then check whether it is a number or a character. If it is a char, determine whether it is Upper case or lower case
 - 2) WAP that displays the user to enter a number between 1 to 7 and then displays the corresponding day of the week
2. Looping -
 - 1) Demonstrate nested looping
 - i. Nested loop to print given pattern


```

*

* *

* * *

* * * *
              
```
 - 2) Demonstrate while loop inside for loop
 - 3) WAP to print the pattern

3 3 3

44 4 4

5 5 5 5 5

- 4) WAP using for loop to calculate factorial of a number
- 5) WAP that displays all leap years from 1900 to 2101
- 6) WAP to sum the series numbers - $1 + 1/2 + \dots + 1/n$ using for loop

THEORY:

Conditional Statements

Conditional statements are used to execute a block of code based on whether a condition evaluates to True or False. These are also known as decision-making statements. The main conditional statements in Python are:

1. **if statement:** Executes a block of code if the condition is true.
 2. **if-else statement:** Executes one block if the condition is true, and another block if the condition is false.
 3. **if-elif-else statement:** Used to test multiple conditions sequentially.
-

Looping Statements

Loops are used to repeat a block of code multiple times. Python provides the following looping constructs:

1. **for loop:** Iterates over a sequence (e.g., list, tuple, string).
2. **while loop:** Repeats a block of code as long as a condition evaluates to True.
3. **Nested loops:** A loop inside another loop, used for multi-dimensional tasks like generating patterns.

CODE:

```
# WAP to take input from a user and then check whether it is a number or a character.  
# If it is a char, determine whether it is Upper case or lower case
```

```
user_input = input("Enter a character or number: ")  
if user_input.isdigit():  
    print(f"The input '{user_input}' is a number.")  
elif user_input.isalpha():  
    if user_input.isupper():  
        print(f"The input '{user_input}' is an uppercase letter.")  
    else:  
        print(f"The input '{user_input}' is a lowercase letter.")  
else:  
    print(f"The input '{user_input}' is invalid.")
```

OUTPUT:

```
+  ⏮  ⏭  📄  📄  ⬆  ⬆  ▶ Run  ⏹  ↺  ⏭  Code  🗨
```

```
In [ ]:
```

```
In [9]: user_input = input("Enter a character or number: ")
        if user_input.isdigit():
            print(f"The input '{user_input}' is a number.")
        elif user_input.isalpha():
            if user_input.isupper():
                print(f"The input '{user_input}' is an uppercase letter.")
            else:
                print(f"The input '{user_input}' is a lowercase letter.")
        else:
            print(f"The input '{user_input}' is invalid.")

        Enter a character or number: a
        The input 'a' is a lowercase letter.
```

```
In [ ]:
```

WAP that displays the user to enter a number between 1 to 7 and then displays the corr day of the week
print("**** Program that displays the user to enter a number between 1 to 7 and then displays the corr day of the week ****")


```
day_num = int(input("Enter a number (1-7): "))
```

```
days = {1: "Monday", 2: "Tuesday", 3: "Wednesday", 4: "Thursday",
        5: "Friday", 6: "Saturday", 7: "Sunday"}
```

```
result = days.get(day_num, "Invalid input!")
```

```
print(f"The day corresponding to the number {day_num} is: {result}")
```

OUTPUT:

 **jupyter** final practical Last Checkpoint: 27 minutes ago (unsaved changes)

```
File  Edit  View  Insert  Cell  Kernel  Widgets  Help
```

```
📄  +  ⏮  ⏭  📄  📄  ⬆  ⬆  ▶ Run  ⏹  ↺  ⏭  Code  🗨
```

```
In [ ]:
```

```
In [10]: day_num = int(input("Enter a number (1-7): "))
        days = {1: "Monday", 2: "Tuesday", 3: "Wednesday", 4: "Thursday",
                5: "Friday", 6: "Saturday", 7: "Sunday"}

        result = days.get(day_num, "Invalid input!")
        print(f"The day corresponding to the number {day_num} is: {result}")

        Enter a number (1-7): 5
        The day corresponding to the number 5 is: Friday
```

```
In [ ]:
```

```
# Nested loop to print pattern
print("The pattern using a nested loop is:")

for i in range(1, 5):
    for j in range(i):
        print("*", end=" ")
    print() # Move to the next line
```

OUTPUT:

In []:

```
In [11]: print("The pattern using a nested loop is:")
for i in range(1, 5):
    for j in range(i):
        print("*", end=" ")
    print() # Move to the next line
```

```
The pattern using a nested loop is:
*
* *
* * *
* * * *
```

In []:

```
# While loop inside for loop
print("Demonstrating a while loop inside a for loop:")
for i in range(1, 4):
    j = 1
    while j <= i:
        print(f"Outer loop index (i): {i}, Inner loop index (j): {j}")
        j += 1
```

OUTPUT:

In []:

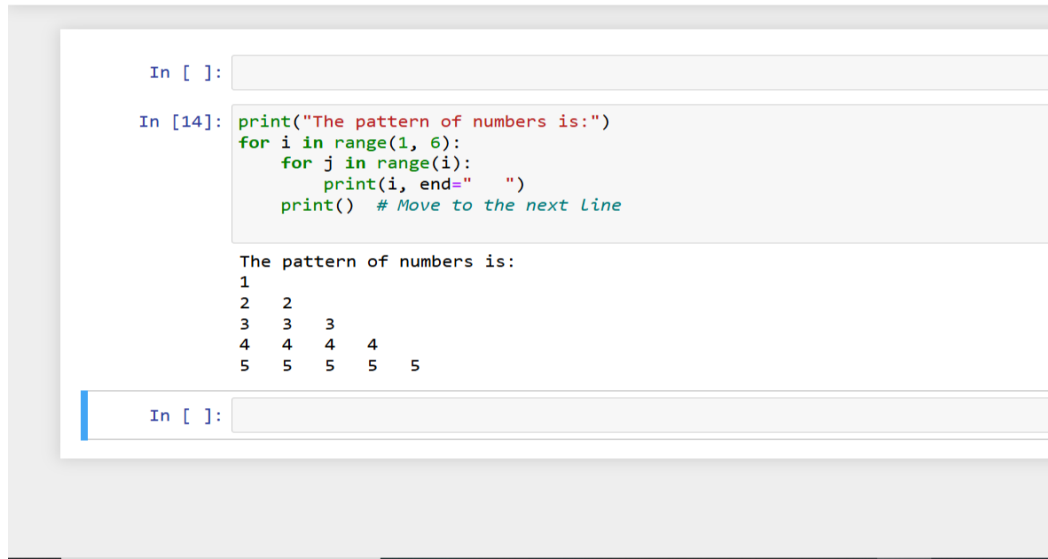
```
In [13]: print("Demonstrating a while loop inside a for loop:")
for i in range(1, 4):
    j = 1
    while j <= i:
        print(f"Outer loop index (i): {i}, Inner loop index (j): {j}")
        j += 1
```

```
Demonstrating a while loop inside a for loop:
Outer loop index (i): 1, Inner loop index (j): 1
Outer loop index (i): 2, Inner loop index (j): 1
Outer loop index (i): 2, Inner loop index (j): 2
Outer loop index (i): 3, Inner loop index (j): 1
Outer loop index (i): 3, Inner loop index (j): 2
Outer loop index (i): 3, Inner loop index (j): 3
```

In []:

```
# WAP to print the pattern
print("The pattern of numbers is:")
for i in range(1, 6):
    for j in range(i):
        print(i, end=" ")
    print() # Move to the next line
```

OUTPUT:



The screenshot shows a Jupyter Notebook interface. The first cell contains the Python code for printing a pattern. The second cell shows the output of the code, which is a pattern of numbers 1 through 5 arranged in a triangular shape. The third cell is empty.

```
In [ ]:
```

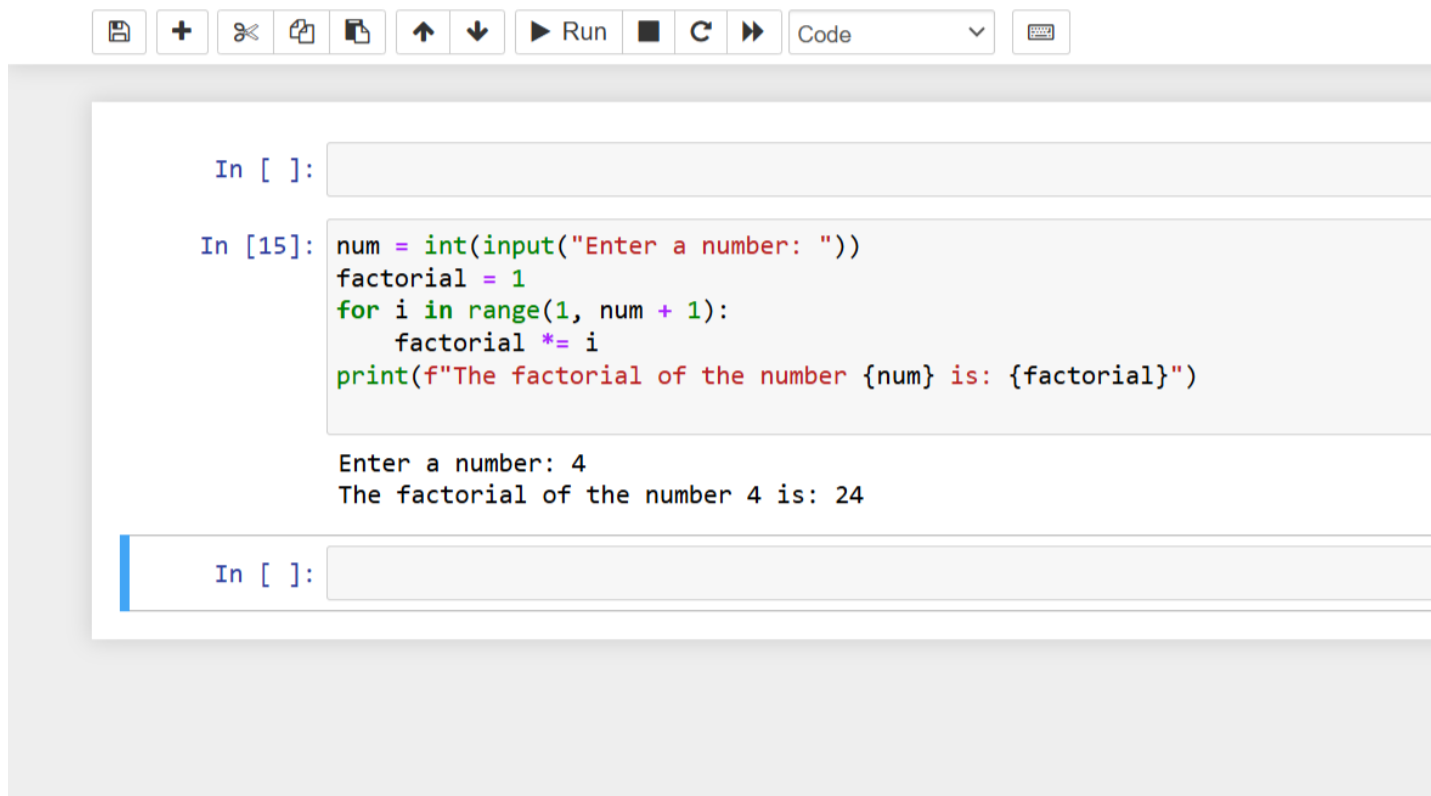
```
In [14]: print("The pattern of numbers is:")
         for i in range(1, 6):
             for j in range(i):
                 print(i, end=" ")
             print() # Move to the next Line
```

```
The pattern of numbers is:
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
In [ ]:
```

```
# Calculating factorial
num = int(input("Enter a number: "))
factorial = 1
for i in range(1, num + 1):
    factorial *= i
print(f"The factorial of the number {num} is: {factorial}")
```

OUTPUT:



The screenshot shows a Jupyter Notebook interface with a toolbar at the top. The first cell is empty. The second cell contains the Python code for calculating the factorial of a number. The third cell shows the output of the code, which is the input number 4 and its factorial 24. The fourth cell is empty.

```
In [ ]:
```

```
In [15]: num = int(input("Enter a number: "))
         factorial = 1
         for i in range(1, num + 1):
             factorial *= i
         print(f"The factorial of the number {num} is: {factorial}")
```

```
Enter a number: 4
The factorial of the number 4 is: 24
```

```
In [ ]:
```

```
# WAP that displays all leap years from 1900 to 2101
print("The leap years between 1900 and 2101 are:")
for year in range(1900, 2102):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        print(year, end=" ")
print() # Move to the next line after printing all years
```

OUTPUT:



```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) C
```

```
In [ ]:
```

```
In [16]: print("The Leap years between 1900 and 2101 are:")
for year in range(1900, 2102):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        print(year, end=" ")
print() # Move to the next line after printing all years
```

```
The Leap years between 1900 and 2101 are:
1904 1908 1912 1916 1920 1924 1928 1932 1936 1940 1944 1948 1952 1956 1960 1964 1968 1972 1976 1980 1984 1988 1992 1996 2000 2004 2008 2012 2016 2020 2024 2028 2032 2036 2040 2044 2048 2052 2056 2060 2064 2068 2072 2076 2080 2084 2088 2092 2096
```

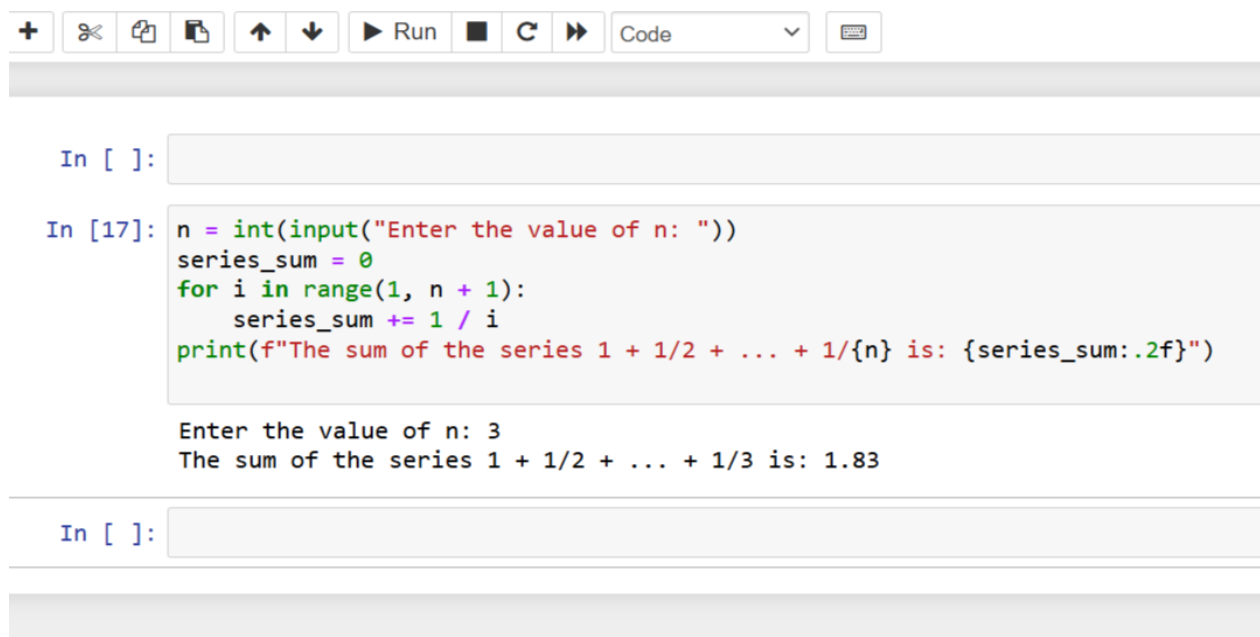
```
# WAP to sum the series numbers - 1 + 1/2 + ... + 1/n using for loop
n = int(input("Enter the value of n: "))

series_sum = 0

for i in range(1, n + 1):
    series_sum += 1 / i

print(f"The sum of the series 1 + 1/2 + ... + 1/{n} is: {series_sum:.2f}")
```

OUTPUT:



```
+ ✂ 📄 📁 ⬆ ⬇ ▶ Run ■ ↺ ⬆ Code ▾ 🗨
```

```
In [ ]:
```

```
In [17]: n = int(input("Enter the value of n: "))
series_sum = 0
for i in range(1, n + 1):
    series_sum += 1 / i
print(f"The sum of the series 1 + 1/2 + ... + 1/{n} is: {series_sum:.2f}")
```

```
Enter the value of n: 3
The sum of the series 1 + 1/2 + ... + 1/3 is: 1.83
```

```
In [ ]:
```

EXPERIMENT 4

OBJECTIVE: Demonstrate list operations and develop code for given problem statements:

1. Demonstrate list slicing and list cloning
2. Demonstrate use of list methods- insert, append, extend, reverse, reversed, remove, pop
3. List comprehension
4. Looping in lists
5. WAP to print index of values in a list
6. Sum and average of elements in list

THEORY:

Lists in Python

A **list** is a mutable, ordered collection of elements. Python provides numerous operations for handling lists efficiently.

1. **List Slicing and Cloning:**
 - Slicing (start:stop:step) extracts portions of a list.
 - Cloning creates a shallow copy of a list using slicing ([:]) or the list() method.
2. **List Methods:**
 - **insert()**: Inserts an element at a specific index.
 - **append()**: Adds an element to the end.
 - **extend()**: Adds elements from another iterable.
 - **reverse()**: Reverses the list in place.
 - **reversed()**: Returns a reversed iterator of the list.
 - **remove()**: Removes the first occurrence of a value.
 - **pop()**: Removes and returns an element by index (default: last).
3. **List Comprehension:**
 - A concise method to create lists using loops and conditions in a single line.
4. **Looping in Lists:**
 - Use for or while loops to iterate through lists.
5. **Calculations:**
 - Use sum() to compute the total and divide by len() for the average.

CODE:

List Slicing

```
list2 = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
print(f"Elements from index 1 to 4 of list2: {list2[1:5]}")
```

```
# -----
```

List Methods (Insert, Append, Extend, Reverse, Slicing)

```
List = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
```

```
print(f"\nOriginal List: {List}")
```

```
Sliced_list = List[:-6]
```

```
print(f'List after slicing out the last 6 elements: {Sliced_list}')

l2 = List[-6:-1]

print(f'Elements from index -6 to -2: {l2}')

l3 = List[::-1]

print(f'Reversed list: {l3}')
```

```
# -----
```

```
# List Comprehension
```

```
l1 = [x**2 for x in range(1, 11) if x % 2 == 1]

print(f'\nSquares of odd numbers from 1 to 10: {l1}')
```

```
# -----
```

```
# Looping in Lists
```

```
ls = [1, 'a', "abc", [2, 3, 4, 5], 8.9]

i = 0

print("\nElements in the list with their indices:")

while i < len(ls):

    print(f'Element at index {i}: {ls[i]}')

    i += 1
```

```
# -----
```

```
# Program to Print Index of Values in a List
```

```
l1 = [1, 2, 3, 4, 5]

print("\nValue and index of each element in the list:")

for i in range(len(l1)):

    print(f'Value: {l1[i]} is at index: {i}')
```

```
# -----
```

```
# Sum and Average of List Items
```

```
l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

s = 0

for i in l1:

    s += i
```



```
print(f"\nSum of the list elements: {s}")
```

```
print(f"Average of the list elements: {s / len(l1)}")
```

OUTPUT:

```
Elements from index 1 to 4 of list2: [2, 3, 4, 5]

Original List: ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
List after slicing out the last 6 elements: ['G', 'E', 'E', 'K', 'S', 'F', 'O']
Elements from index -6 to -2: ['R', 'G', 'E', 'E', 'K']
Reversed list: ['S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E', 'G']

Squares of odd numbers from 1 to 10: [1, 9, 25, 49, 81]

Elements in the list with their indices:
Element at index 0: 1
Element at index 1: a
Element at index 2: abc
Element at index 3: [2, 3, 4, 5]
Element at index 4: 8.9

Value and index of each element in the list:
Value: 1 is at index: 0
Value: 2 is at index: 1
Value: 3 is at index: 2
Value: 4 is at index: 3
Value: 5 is at index: 4

Sum of the list elements: 55
Average of the list elements: 5.5
```

EXPERIMENT 5

OBJECTIVE: Demonstrate arrays and tuples and develop code for given problem statements:

1. Operations in array - Create array in python, Demonstrate functions in arrays - insert(), append(), Slicing in array, updating elements in array
2. Create an empty tuple, create tuple using string, create tuple using list, and create a tuple with mixed datatypes
3. Write a program to demonstrate use of nested tuples. Also, WAP that has a nested list to store toppers details. Edit the details and reprint the details.
4. Creating a tuple using Loop
5. WAP to swap two values using tuple assignment
6. WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument
7. WAP that scans an email address and forms a tuple of username and domain

THEORY:

Arrays and Tuples in Python

1. Arrays

Arrays are data structures that store elements of the same data type in contiguous memory locations. Python provides arrays through the array module, supporting basic operations like insertion, updating, slicing, and appending.

- **array():** Used to create an array.
- **insert(index, value):** Adds an element at a specific index.
- **append(value):** Adds an element to the end of the array.
- **Slicing:** Access a subset of elements using start:stop:step.
- **Updating Elements:** Modify values by accessing their index.

2. Tuples

Tuples are immutable, ordered collections of elements. Unlike lists, tuples cannot be modified after creation, which makes them ideal for storing fixed data.

- **Creating Tuples:** Tuples can be created using parentheses () or directly from other data types like strings and lists.
- **Mixed Data Types:** Tuples can hold elements of different types, including nested structures like lists and other tuples.

CODE:

Array Operations in Python

```
import array as arr
```

1. Creating integer and float arrays

```
a = arr.array('i', [1, 2, 3])
```

```
print("Array of integers: ", a)
```

```
for i in range(0, len(a)):
```

```
    print(f"Element {i}: {a[i]}", end=" ")
```

2. Demonstrate insert() and append()

```
print("\n\nBefore Insertion: ", a)
```

```
a.insert(1, 4) # Insert value at index 1
```

```
print("After Insertion: ", a)
```

```
b = arr.array('d', [1.0, 2.0, 3.0])
```

```
print("Before Appending: ", b)
```

```
b.append(4.4) # Append a float value
```

```
print("After Appending: ", b)
```

3. Slicing an array

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
a = arr.array('i', l)
```

```
print("\nOriginal Array: ", list(a))
```

```
print("Slice [3:8]: ", a[3:8])
```

```
print("Slice [5:]: ", a[5:])
```

```
print("All elements using slice: ", a[:])
```

4. Array Updation

```
arr_upd = array.array('i', [1, 2, 3, 1, 2, 5])
```

```
print("\nBefore Update: ", arr_upd)
```

```
arr_upd[2] = 6 # Update index 2 with value 6
```

```
print("After Update: ", arr_upd)
```

Create empty tuple:

```
tuple1 = ()
```

```
print(tuple1)
```

Create tuple using string:

```
tuple1 = ('Hello', 'Sam')
```

```
print(tuple1)
```

Create tuple using list:

Tuple Operations in Python

OUTPUT:

```
Array of integers: array('i', [1, 2, 3])
Element 0: 1 Element 1: 2 Element 2: 3

Before Insertion: array('i', [1, 2, 3])
After Insertion: array('i', [1, 4, 2, 3])
Before Appending: array('d', [1.0, 2.0, 3.0])
After Appending: array('d', [1.0, 2.0, 3.0, 4.4])
```

```
Original Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Slice [3:8]: array('i', [4, 5, 6, 7, 8])
Slice [5:]: array('i', [6, 7, 8, 9, 10])
All elements using slice: array('i', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
Before Update: array('i', [1, 2, 3, 1, 2, 5])
After Update: array('i', [1, 2, 6, 1, 2, 5])
```

1.

1. Creating tuples

```
tuple1 = () # Empty tuple
print("\nEmpty Tuple: ", tuple1)
```

```
tuple1 = ('Hello', 'Sam') # Using strings
print("Tuple with strings: ", tuple1)
```

```
list1 = ['Hello', 'Sam'] # From list
print("Tuple from list: ", tuple(list1))
```

```
tuple1 = tuple('Sam') # Using built-in function
print("Tuple from string using tuple(): ", tuple1)
```

```
tuple1 = (5, 'aiojdio', 7, 'JFidsof') # Mixed datatypes
print("Tuple with mixed datatypes: ", tuple1)
```

2. Nested tuples

```
t1 = (1, 2, 3)
t2 = ('a', 'b', 'c')
t3 = (t1, t2)
print("Nested Tuple: ", t3)
```

3. Tuple Creation Using Loop

```
t1 = ('Sam',)
```

```
n = 5
print("\nTuple Creation Using Loop:")
for _ in range(n):
    t1 = (t1,)
    print(t1)
```

4. Swapping two values using tuple assignment

```
t1 = (2, 3)
a, b = t1
print("\nBefore Swap: a =", a, ", b =", b)
a, b = b, a
print("After Swap: a =", a, ", b =", b)
```

5. Area and circumference of a circle

```
import math

def func1(r):
    area = math.pi * r * r
    circum = 2 * math.pi * r
    return area, circum

rad = float(input("\nEnter radius of the circle: "))
area, circum = func1(rad)
print("Area: ", area)
print("Circumference: ", circum)
```

6. Extract username and domain from email

```
email = input("\nEnter the email address: ")
email_tuple = tuple(email.split("@"))
print("Extracted Tuple: Username =", email_tuple[0], ", Domain =", email_tuple[1])
```

OUTPUT:

```
Empty Tuple: ()
Tuple with strings: ('Hello', 'Sam')
Tuple from list: ('Hello', 'Sam')
Tuple from string using tuple(): ('S', 'a', 'm')
Tuple with mixed datatypes: (5, 'aiojdio', 7, 'JFidsof')
Nested Tuple: ((1, 2, 3), ('a', 'b', 'c'))

Tuple Creation Using Loop:
(('Sam',),)
(((('Sam',),),),)
((((('Sam',),),),),)
((((('Sam',),),),),)
((((('Sam',),),),),)

Before Swap: a = 2 , b = 3
After Swap: a = 3 , b = 2

Enter radius of the circle: 5
Area: 78.53981633974483
Circumference: 31.41592653589793

Enter the email address: sahilsandal@gmail.com
Extracted Tuple: Username = sahilsandal , Domain = gmail.com
```

EXPERIMENT 6

OBJECTIVE: Demonstrate functions and modules and develop code for given problem statements:

1. Create a function to return the square of the number
2. Demonstrate Pass by Reference and Pass by value
3. WAP that subtracts two numbers using a function
4. WAP using functions and return statements to check whether a number is even or odd
5. WAP to calculate simple interest. Suppose the customer is a Senior citizen and is being offered 12% ROI. For all other customers, ROI is 10%.
6. Program to find certain power of a number using recursion

THEORY:

Functions in Python

Functions are reusable blocks of code designed to perform a specific task. Functions improve modularity and reduce code redundancy.

CODE:

Defining the function

```
def square(num):  
    return num**2
```

```
obj = square(6)  
print(obj)
```

OUTPUT:

```
In [25]: def square(num):  
         return num**2  
  
         obj = square(6)  
         print(obj)
```

36

Pass by Reference and Pass by value

```
def square(item_list):  
    squares = []  
    for i in item_list:  
        squares.append(i**2)  
    return squares
```

```
num = [1, 2, 3, 4, 5]  
obj = square(num)  
print(obj)
```

OUTPUT:

```
In [34]: def square(item_list):
          squares = []
          for i in item_list:
              squares.append(i**2)
          return squares

          num = [1, 2, 3, 4, 5]
          obj = square(num)
          print(obj)

          [1, 4, 9, 16, 25]
```

Pass by value

```
obj = square([1, 2, 3, 4, 5])
```

```
print(obj)
```

OUTPUT:

```
In [ ]:
```

```
In [ ]:
```

```
In [27]: obj = square([1, 2, 3, 4, 5])
          print(obj)
```

```
[1, 4, 9, 16, 25]
```

WAP that subtracts two numbers using a function

```
def func(a, b):
```

```
    return a - b
```

```
a = int(input("Enter num1: "))
```

```
b = int(input("Enter num2: "))
```

```
print("num1 - num2 = ", func(a, b))
```

OUTPUT:

```
In [32]: def func(a, b):
          return a - b

          a = int(input("Enter num1: "))
          b = int(input("Enter num2: "))
          print("num1 - num2 = ", func(a, b))

          Enter num1: 3
          Enter num2: 3
          num1 - num2 = 0
```

```
# WAP using functions and return statements to check whether a number is even or odd
def func(a):
    if a % 2 == 0:
        return "Even"
    else:
        return "Odd"

a = int(input("Enter num1: "))
print("Number is", func(a))
```

OUTPUT:

```
In [33]: def func(a):
         if a % 2 == 0:
             return "Even"
         else:
             return "Odd"

         a = int(input("Enter num1: "))
         print("Number is", func(a))

Enter num1: 3
Number is Odd
```

```
# WAP to calculate simple interest.
age = int(input("Enter age of person: "))
principal = float(input("Enter principal amount: "))
time = int(input("Enter time in years: "))

if age >= 60:
    r = 12
else:
    r = 10

si = principal * r * time / 100
print("Simple Interest is: ", si)
```

OUTPUT:

```
In [30]: age = int(input("Enter age of person: "))
         principal = float(input("Enter principal amount: "))
         time = int(input("Enter time in years: "))

         if age >= 60:
             r = 12
         else:
             r = 10

         si = principal * r * time / 100
         print("Simple Interest is: ", si)

Enter age of person: 40
Enter principal amount: 1000000
Enter time in years: 5
Simple Interest is: 500000.0
```


Program to find certain power of a number using recursion
def func1(n, i):

 if i == 0:

 return 1

 else:

 return n * func1(n, i - 1)

func1(2, 6)

OUTPUT:

```
In [31]: def func1(n, i):  
         if i == 0:  
             return 1  
         else:  
             return n * func1(n, i - 1)  
  
         func1(2, 6)
```

Out[31]: 64

EXPERIMENT 7

OBJECTIVE: Demonstrate Set operations and develop code for given problem statements:

1. Set Operations - Create set, Add items in set, Add items from another set into this set, Add elements of a list to the set, Remove item, Remove item using discard()
2. WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update, pop, remove and clear function
3. WAP that creates two sets one of even numbers in the range 1 to 10 and the other as all composite numbers in range 1 to 20. Demonstrate the use of all(), issuperset(), len() and sum() on the sets.

THEORY:

Sets in Python

A **set** is an unordered, mutable collection of unique elements. Python sets are used for membership tests, removing duplicates, and performing mathematical set operations like union, intersection, and difference.

CODE:

```
# SETS
```

```
# SETS: Basic Operations
```

```
# WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update, pop, remove and clear function
```

```
# Creating a set
```

```
thisset = {"apple", "banana", "cherry"}  
print("Initial set:", thisset)  
print("Type of the set:", type(thisset))  
print("Is 'banana' in the set?", "banana" in thisset)
```

```
# Add items in set
```

```
thisset.add("orange")  
print("\nAfter adding 'orange':", thisset)
```

```
# Add items from another set into this set
```

```
tropical = {"mango", "papaya"}  
thisset.update(tropical)  
print("\nAfter updating with another set (tropical):", thisset)
```

```
# Add elements of a list to the set
```

```
l1 = ["mango2", "papaya2"]  
thisset.update(l1)  
print("\nAfter updating with a list:", thisset)
```

```
# Remove item
```

```
thisset.remove("mango2")  
print("\nAfter removing 'mango2':", thisset)
```

```

# Remove item using discard()
thisset.discard("banana")
print("\nAfter discarding 'banana':", thisset)

# WAP: Sets of squares and cubes in range 1 to 10
squares = {i ** 2 for i in range(1, 11)}
cubes = {i ** 3 for i in range(1, 11)}

print("\nSet of squares (1 to 10):", squares)
print("Set of cubes (1 to 10):", cubes)

# Demonstrating update, pop, remove, and clear functions
print("\nUsing update() to add 'example' to squares:")
squares.update({"example"})
print("Squares after update:", squares)

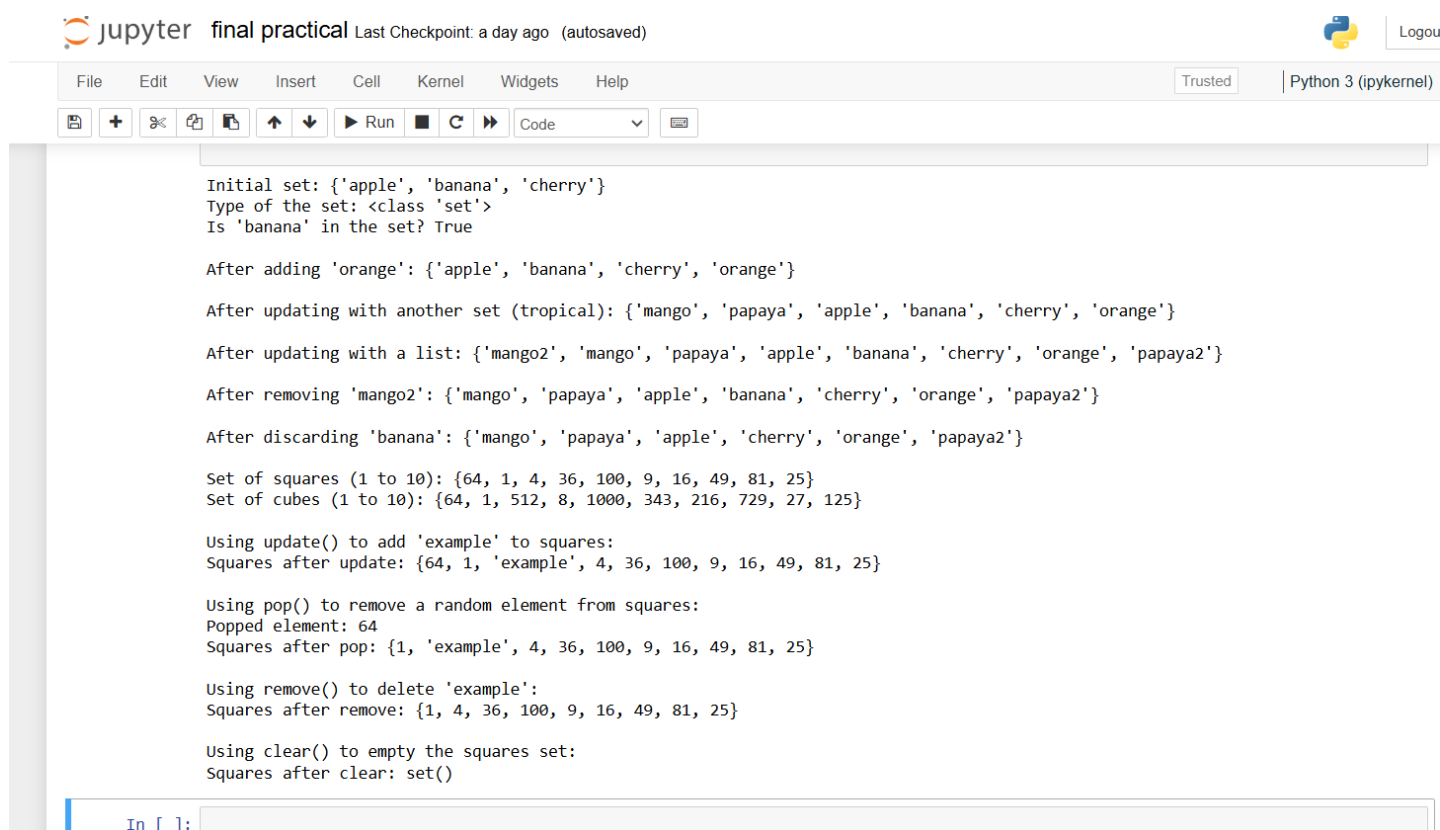
print("\nUsing pop() to remove a random element from squares:")
print("Popped element:", squares.pop())
print("Squares after pop:", squares)

print("\nUsing remove() to delete 'example':")
squares.remove("example")
print("Squares after remove:", squares)

print("\nUsing clear() to empty the squares set:")
squares.clear()
print("Squares after clear:", squares)

```

OUTPUT:



```

Initial set: {'apple', 'banana', 'cherry'}
Type of the set: <class 'set'>
Is 'banana' in the set? True

After adding 'orange': {'apple', 'banana', 'cherry', 'orange'}

After updating with another set (tropical): {'mango', 'papaya', 'apple', 'banana', 'cherry', 'orange'}

After updating with a list: {'mango2', 'mango', 'papaya', 'apple', 'banana', 'cherry', 'orange', 'papaya2'}

After removing 'mango2': {'mango', 'papaya', 'apple', 'banana', 'cherry', 'orange', 'papaya2'}

After discarding 'banana': {'mango', 'papaya', 'apple', 'cherry', 'orange', 'papaya2'}

Set of squares (1 to 10): {64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
Set of cubes (1 to 10): {64, 1, 512, 8, 1000, 343, 216, 729, 27, 125}

Using update() to add 'example' to squares:
Squares after update: {64, 1, 'example', 4, 36, 100, 9, 16, 49, 81, 25}

Using pop() to remove a random element from squares:
Popped element: 64
Squares after pop: {1, 'example', 4, 36, 100, 9, 16, 49, 81, 25}

Using remove() to delete 'example':
Squares after remove: {1, 4, 36, 100, 9, 16, 49, 81, 25}

Using clear() to empty the squares set:
Squares after clear: set()

```

```

# WAP that creates two sets one of even numbers in the range 1 to 10 and the other as all composite numbers
in range 1 to 20
# Demonstrate the use of all(), issuperset(), len() and sum() on the sets.

# WAP: Even and Composite Sets

# Set of even numbers in range 1 to 10
even_numbers = {i for i in range(1, 11) if i % 2 == 0}
print("Set of even numbers (1 to 10):", even_numbers)

# Set of composite numbers in range 1 to 20
composite_numbers = set()
for i in range(2, 21):
    for j in range(2, i):
        if i % j == 0:
            composite_numbers.add(i)
            break
print("Set of composite numbers (1 to 20):", composite_numbers)

# Demonstrating set functions

# all() function: Check if all elements in the set are truthy
print("\nUsing all() on even_numbers:", all(even_numbers))

# Removing an element
even_numbers.remove(2)
print("\nSet after removing '2' from even_numbers:", even_numbers)

# issuperset() function
print("\nChecking if composite_numbers is a superset of even_numbers:",
      composite_numbers.issuperset(even_numbers))

# len() function
print("\nLength of composite_numbers:", len(composite_numbers))

# sum() function
print("\nSum of elements in even_numbers:", sum(even_numbers))

```

OUTPUT:

```

Set of even numbers (1 to 10): {2, 4, 6, 8, 10}
Set of composite numbers (1 to 20): {4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20}

Using all() on even_numbers: True

Set after removing '2' from even_numbers: {4, 6, 8, 10}

Checking if composite_numbers is a superset of even_numbers: True

Length of composite_numbers: 11

Sum of elements in even_numbers: 28

```

EXPERIMENT 8

OBJECTIVE: Demonstrate dictionary operations and develop code for given problem statements:

1. Dictionary Operations –
 - a. Accessing values in a Dictionary, Updating a dict, adding new values, Delete particular entries, Clear whole dict, Delete whole dict
 - b. Dictionary methods – len(), copy(), dictionary to string, Fromkeys(), get(), items(), setdefault(), Update(), values()
2. WAP to merge two dictionaries with a third one
3. Iterating through a dictionary
4. WAP to Sort dictionary by values

THEORY:

A **dictionary** in Python is a collection of key-value pairs, where each key must be unique. Dictionaries are mutable, unordered, and allow for fast retrieval and modification of data. They are often used to store and manage related data in a structured way.

Dictionary Operations

1. **Accessing Values in a Dictionary:**
You can access a value in a dictionary by referring to its key, e.g., dict[key]. You can also use the get() method to avoid errors when a key is missing.
2. **Updating a Dictionary:**
You can add or modify an entry by assigning a value to a key, e.g., dict[key] = value.
3. **Adding New Values:**
You can add new key-value pairs using the same syntax used for updating, or use the update() method to merge two dictionaries.
4. **Deleting Particular Entries:**
To remove a specific entry, use del dict[key]. Alternatively, you can use pop() to remove a key-value pair and return the value.
5. **Clearing the Whole Dictionary:**
To remove all entries from a dictionary, use dict.clear().
6. **Deleting the Whole Dictionary:**
To delete the dictionary entirely, use del dict.

CODE:

```
# Accessing values in a dictionary
dict1 = {'Name': 'Sahil', 'Age': 18, 'Class': '10'}
print("Value of 'Name':", dict1['Name'])
print("Value of 'Age':", dict1['Age'])

# Updating a dictionary
dict1['Age'] = 19 # Updating age
print("Updated dictionary (Age changed):", dict1)
```

```
# Adding a new entry
```

```
dict1['School'] = 'Greenwood High'
```

```
print("Dictionary after adding 'School':", dict1)
```

```
# Deleting an entry
```

```
del dict1['Name']
```

```
print("Dictionary after deleting 'Name':", dict1)
```

```
# Clearing the entire dictionary
```

```
dict1.clear()
```

```
print("Dictionary after clearing all entries:", dict1)
```

```
# Deleting the dictionary entirely
```

```
del dict1
```

```
try:
```

```
    print(dict1)
```

```
except NameError:
```

```
    print("Error: 'dict1' no longer exists as it was deleted.")
```

```
# Merging two dictionaries into a third one
```

```
a = {'Name': 'Sahil', 'Age': 18}
```

```
b = {'Gender': 'Male'}
```

```
c = {'Senior_Citizen': 'No'}
```

```
c.update(b) # Merging 'b' into 'c'
```

```
c.update(a) # Merging 'a' into 'c'
```

```
print("Merged dictionary:", c)
```

```
# Iterating through a dictionary
```

```
dict1 = {"x": "book", "y": "pen", "z": "bag"}
```

```
print("Key-Value pairs in dictionary:")
```

```
for key, value in dict1.items():
```

```
    print(key, ":", value)
```

```
print("\nKeys in dictionary:")
```

```
for key in dict1.keys():
```

```
    print(key)
```

```

print("\nValues in dictionary:")

for value in dict1.values():

    print(value)


# Sorting a dictionary by values
dict1 = {"p": 10, "q": 50, "r": 5, "s": 25, "t": 100}
print("\nOriginal dictionary:", dict1)


sorted_values = sorted(dict1.values())
print("Sorted values:", sorted_values)


# Creating a new dictionary sorted by values
sorted_dict = {}
for value in sorted_values:
    for key in dict1.keys():
        if dict1[key] == value:
            sorted_dict[key] = value


print("Dictionary sorted by values:", sorted_dict)

```

OUTPUT:

```

Value of 'Name': Sahil
Value of 'Age': 18
Updated dictionary (Age changed): {'Name': 'Sahil', 'Age': 19, 'Class': '10'}
Dictionary after adding 'School': {'Name': 'Sahil', 'Age': 19, 'Class': '10', 'School': 'Greenwood High'}
Dictionary after deleting 'Name': {'Age': 19, 'Class': '10', 'School': 'Greenwood High'}
Dictionary after clearing all entries: {}
Error: 'dict1' no longer exists as it was deleted.
Merged dictionary: {'Senior_Citizen': 'No', 'Gender': 'Male', 'Name': 'Sahil', 'Age': 18}
Key-Value pairs in dictionary:
x : book
y : pen
z : bag

Keys in dictionary:
x
y
z

Values in dictionary:
book
pen
bag

Original dictionary: {'p': 10, 'q': 50, 'r': 5, 's': 25, 't': 100}
Sorted values: [5, 10, 25, 50, 100]
Dictionary sorted by values: {'r': 5, 'p': 10, 's': 25, 'q': 50, 't': 100}

```

EXPERIMENT 9

OBJECTIVE: Demonstrate strings and its related operations and develop code for given problem statements:

- 1) Slicing – WAP to Get the characters from o in “World” to but not included d in "World"
- 2) WAP to display powers of number without using formatting characters
- 3) String methods and functions –
 - i. capitalize(), center(), count(), endswith(), startswith(), find(), index(), rfind(), rindex(), isalnum(), isalpha(), isdigit(), islower(), isupper(), len(), etc.
 - ii. WAP to print following pattern

A
AB
ABC
ABCD
ABCDE
ABCDEF
 - iii. WAP using while loop to iterate a given string
 - iv. WAP that encrypts a message by adding a key value to every character
 - v. WAP that uses split function to split a multi-line string
 - vi. WAP that accepts a string from user and re-displays the same string after removing vowels
- 4) Regular Expressions
 - i. WAP to find patterns that begin with one or more characters followed by space and followed by one or more digits
 - ii. WAP that uses a regex to match strings which start with sequence of digits (atleast 1) followed by a blank and after this add arbitrary characters

THEORY:

1) Slicing:

- Slicing in Python extracts a portion of a string using string[start:end]. The start index is inclusive, and the end index is exclusive. For example, "World"[1:4] gives "orl".

2) Power Calculation Without Formatting:

- To display powers of a number, use a loop to raise the number to different powers. This avoids using formatting characters like % or f-string.

3) String Methods:

- Python strings come with built-in methods for manipulation:
 - capitalize(), center(), count(), endswith(), startswith(), find(), index(), isalnum(), isalpha(), isdigit(), islower(), isupper(), and len() are commonly used methods.

4) While Loop to Iterate Through a String:

- A while loop can iterate over a string, accessing each character using its index.

5) Encrypting a Message:

- Message encryption can be done by shifting the ASCII value of each character by a specified key, effectively "encrypting" the message.

CODE:

```
# Displaying powers of numbers without formatting characters
```

```
i = 1
print("Powers of numbers (using manual formatting):")
while i <= 5:
    print(i**1, "\t", i**2, "\t", i**3, "\t", i**4)
    i += 1
print("\n")
```

```
i = 1
print("Powers of numbers (using %d formatting):")
while i <= 5:
    print("%d\t%d\t%d\t%d" % (i**1, i**2, i**3, i**4))
    i += 1
print("\n")
```

```
# Creating a table-like structure for powers
print("Table of i, i^2, and i^3:")
print("%-4s%-5s%-6s" % ('i', 'i**2', 'i**3'))
i = 1
while i <= 5:
    print("%-4d%-5d%-6d" % (i, i**2, i**3))
    i += 1
```

```
# Printing a pattern with letters
```

```
print("\nPattern of letters:")
for i in range(1, 7):
    ch = 'A'
    for j in range(1, i + 1):
        print(ch, end="")
        ch = chr(ord(ch) + 1)
    print()
```

OUTPUT:

```
Powers of numbers (using manual formatting):
```

```
1      1      1      1
2      4      8      16
3      9     27     81
4     16     64    256
5     25    125   625
```

```
Powers of numbers (using %d formatting):
```

```
1      1      1      1
2      4      8      16
3      9     27     81
4     16     64    256
5     25    125   625
```

```
Table of i, i^2, and i^3:
```

```
i      i**2  i**3
1      1      1
2      4      8
3      9     27
4     16     64
5     25    125
```

```
Pattern of letters:
```

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
```

```
# Built-in string methods and functions
```

```

# Demonstrating various string methods
s = "hello"
print("\nString Methods Demonstrations:")
print("Capitalize:", s.capitalize())
print("Center with '*':", s.center(10, '*'))

msg = 'he'
str1 = "hellohello"
print("Count occurrences of 'he':", str1.count(msg))

msg = "she is my best friend"
print("Ends with 'end':", msg.endswith("end"))

str1 = "the world is beautiful"
print("Starts with 'th':", str1.startswith("th"))

msg = "she is my best my friend"
print("Find 'my':", msg.find("my"))
print("Find 'mine':", msg.find("mine"))

try:
    print("Index of 'mine':", msg.index("mine"))
except ValueError:
    print("Substring 'mine' not found!")

msg = "is this your bag?"
print("Reverse find 'is':", msg.rfind("is"))

msg = "jamesbond007"
print("Is Alphanumeric:", msg.isalnum())
print("Is Alphabetic:", msg.isalpha())
msg = "007"
print("Is Digit:", msg.isdigit())

# Regex for identifying patterns
import re
pattern = r"[a-zA-Z]+\s+\d+"
matches = re.finditer(pattern, "LXI 2013, VXI 2015, VDI 20104, Maruti Suzuki Cars available with us")
print("\nRegex Matches (Words followed by numbers):")
for match in matches:
    print(f'Match: {match.group()} at {match.span()}')

```

OUTPUT:

```

String Methods Demonstrations:
Capitalize: Hello
Center with '*': **hello***
Count occurrences of 'he': 2
Ends with 'end': True
Starts with 'th': True
Find 'my': 7
Find 'mine': -1
Substring 'mine' not found!
Reverse find 'is': 5
Is Alphanumeric: True
Is Alphabetic: False
Is Digit: True

Regex Matches (Words followed by numbers):
Match: LXI 2013 at (0, 8)
Match: VXI 2015 at (10, 18)
Match: VDI 20104 at (20, 29)

```

WAP that encrypts a message by adding a key value to every character

Encrypting a message by shifting characters using a key

```
s = input("Enter the string to encrypt: ")
```

```
key = int(input("Enter the encryption key (integer): "))
```

```
new_s = ""
```

```
for i in s:
```

```
    new_s += chr(ord(i) + key)
```

```
print("Encrypted string:", new_s)
```

Splitting a multi-line string

```
s = "Dear Students, I am pleased to inform you that, there is a workshop on Python in college tomorrow.
```

```
Everyone should come and there will also be a quiz in Python, whosoever wins will win a gold medal."
```

```
print("\nSplitting a multi-line string into lines:")
```

```
print(s.split('\n'))
```

Removing vowels from a string

```
vowels = "aeiouAEIOU"
```

```
s = input("\nEnter a string to remove vowels: ")
```

```
result = ""
```

```
for char in s:
```

```
    if char not in vowels:
```

```
        result += char
```

```
print("String after removing vowels:", result)
```

OUTPUT:

```
# Encrypting a message by shifting characters using a key
s = input("Enter the string to encrypt: ")
key = int(input("Enter the encryption key (integer): "))
new_s = ""
for i in s:
    new_s += chr(ord(i) + key)
print("Encrypted string:", new_s)

# Splitting a multi-line string
s = '''Dear Students, I am pleased to inform you that, there is a workshop on Python in college tomorrow.
Everyone should come and there will also be a quiz in Python, whosoever wins will win a gold medal.'''
print("\nSplitting a multi-line string into lines:")
print(s.split('\n'))

# Removing vowels from a string
vowels = "aeiouAEIOU"
s = input("\nEnter a string to remove vowels: ")
result = ""
for char in s:
    if char not in vowels:
        result += char
print("String after removing vowels:", result)
```

Enter the string to encrypt: hey i am a ai student

Enter the encryption key (integer): 1324

Encrypted string: njnfnfnfn

Splitting a multi-line string into lines:

['Dear Students, I am pleased to inform you that, there is a workshop on Python in college tomorrow.', 'Everyone should come and there will also be a quiz in Python, whosoever wins will win a gold medal.']

Enter a string to remove vowels: i am a student

String after removing vowels: m stdnt

EXPERIMENT 10

OBJECTIVE: Demonstrate file handling and develop code for given problem statements:

- 1) WAP that copies first 10 bytes of a binary file into another
- 2) WAP that accepts a file name as an input from the user. Open the file and count the number of times a character appears in the file
- 3) WAP to create a new directory in the current directory, WAP that changes current directory to newly created directory new_dir, WAP to delete new_dir
- 4) WAP to print the absolute path of a file using os.path.join

THEORY:

File Handling in Python: File handling in Python refers to the process of reading from, writing to, and performing other operations on files. Python provides built-in functions to handle files, such as open(), read(), write(), and others. Files can be opened in various modes like 'r' (read), 'w' (write), 'a' (append), 'b' (binary), etc. You can also manipulate file paths using the os and os.path modules.

CODE:

Copying first 10 bytes of a binary file into another

```
import os
with open("file_handling.txt", "rb") as f:
    a = f.read(10)
print("First 10 bytes of file1:", a)

with open("file_handling2.txt", "wb+") as f2:
    print("File2 contents before copying:")
    f2.seek(0)
    print(f2.read())
    f2.seek(0)
    t = f2.write(a)
    f2.seek(0)
    print("File2 contents after copying:")
    print(f2.read())
```

Creating a new directory

```
os.mkdir("new_dir")
print("New directory 'new_dir' created.")
```

Changing the current working directory to the new directory

```
os.chdir("new_dir")
print("Current directory changed to 'new_dir'.")
```

Deleting the new directory

```
os.chdir("../") # Move back to the parent directory
os.rmdir("new_dir")
print("Directory 'new_dir' deleted.")
```

OUTPUT:

```
First 10 bytes of file1: b'hey i am a'
File2 contents before copying:
b''
File2 contents after copying:
b'hey i am a'
New directory 'new_dir' created.
Current directory changed to 'new_dir'.
Directory 'new_dir' deleted.
```

EXPERIMENT 11

Classes, objects and inheritance :

- 1) WAP with class Employee that keeps a track of the number of employees in an organisation and also stores their name, designation, and salary details.
- 2) WAP that has a class Circle. Use a class variable to define the value of constant pi. Use this class variable to calculate area and circumference of a circle with specified radius.
- 3) Inheritance
 - i. WAP that has a class Point. Define another class Location which has 2 objects - location and destination. Also define a function in location that prints the reflection of destination on the x-axis.
 - ii. WAP that has classes such as Student, Course, Department. Enroll a student in a course of a particular department. Classes are -
 1. Student details - name, roll no
 2. Course - name, code, year and semester
 3. Department – Name

THEORY:

Classes, Objects, and Inheritance

- Classes and Objects

Class: A class is a blueprint for creating objects. It defines attributes (variables) and methods (functions) to represent the behavior of the objects.

Object: An instance of a class containing its own data and behavior.

- **Inheritance**

Inheritance allows a class (child) to inherit the properties and methods of another class (parent). It supports code reuse and creates a hierarchical relationship between classes.

Types of Inheritance:

Single: One class inherits from another.

Multiple: A class inherits from multiple classes.

Multilevel: A class inherits from a child class, creating a chain.

CODE:

WAP with class Employee that keeps a track of the number of employees in an organisation and also stores their name, designation, and salary details.

```
class Employee:
```

```
    count = 0 # Class variable to track the number of employees
```

```
    def __init__(self, name, designation, salary):
```

```
self.name = name
self.designation = designation
self.salary = salary
Employee.count += 1
```

```
@staticmethod
```

```
def display_count():
    print(f"Total Employees: {Employee.count}")
```

```
def display_details(self):
    print(f"Name: {self.name}, Designation: {self.designation}, Salary: {self.salary}")
```

```
# Example usage
```

```
emp1 = Employee("John", "Manager", 50000)
```

```
emp2 = Employee("Jane", "Developer", 40000)
```

```
emp1.display_details()
```

```
emp2.display_details()
```

```
Employee.display_count()
```

OUTPUT:

```
In [10]: class Employee:
          count = 0 # Class variable to track the number of employees

          def __init__(self, name, designation, salary):
              self.name = name
              self.designation = designation
              self.salary = salary
              Employee.count += 1

          @staticmethod
          def display_count():
              print(f"Total Employees: {Employee.count}")

          def display_details(self):
              print(f"Name: {self.name}, Designation: {self.designation}, Salary: {self.salary}")

          # Example usage
          emp1 = Employee("John", "Manager", 50000)
          emp2 = Employee("Jane", "Developer", 40000)

          emp1.display_details()
          emp2.display_details()
          Employee.display_count()

Name: John, Designation: Manager, Salary: 50000
Name: Jane, Designation: Developer, Salary: 40000
Total Employees: 2
```

WAP that has a class Circle. Use a class variable to define the value of constant pi. Use this class variable to calculate area and circumference of a circle with specified radius.

```
class Circle:
```

```
    pi = 3.14159 # Class variable for constant pi
```

```
    def __init__(self, radius):
        self.radius = radius
```

```
    def calculate_area(self):
        return Circle.pi * self.radius ** 2
```



```

def calculate_circumference(self):
    return 2 * Circle.pi * self.radius

def display(self):
    print(f"Radius: {self.radius}")
    print(f"Area: {self.calculate_area()}")
    print(f"Circumference: {self.calculate_circumference()}")

# Example usage
circle1 = Circle(7)
circle2 = Circle(10)

circle1.display()
circle2.display()

```

OUTPUT:

```

[11]: class Circle:
      pi = 3.14159 # class variable for constant pi

      def __init__(self, radius):
          self.radius = radius

      def calculate_area(self):
          return Circle.pi * self.radius ** 2

      def calculate_circumference(self):
          return 2 * Circle.pi * self.radius

      def display(self):
          print(f"Radius: {self.radius}")
          print(f"Area: {self.calculate_area()}")
          print(f"Circumference: {self.calculate_circumference()}")

# Example usage
circle1 = Circle(7)
circle2 = Circle(10)

circle1.display()
circle2.display()

Radius: 7
Area: 153.93791
Circumference: 43.98226
Radius: 10
Area: 314.159
Circumference: 62.8318

```

Inheritance

PART I : Point and Location Classes

```

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def display(self):
        print(f"Point Coordinates: ({self.x}, {self.y})")

class Location(Point):
    def __init__(self, x, y, dest_x, dest_y):

```

```
super().__init__(x, y)
self.destination = Point(dest_x, dest_y)
```

```
def reflect_x(self):
    reflected = Point(self.destination.x, -self.destination.y)
    return reflected
```

```
def display(self):
    super().display()
    print(f'Destination Coordinates: ({self.destination.x}, {self.destination.y})')
    reflected = self.reflect_x()
    print(f'Reflection on X-axis: ({reflected.x}, {reflected.y})')
```

Example usage

```
loc = Location(0, 0, 4, 5)
```

```
loc.display()
```

OUTPUT:

```
In [12]: class Point:
        def __init__(self, x, y):
            self.x = x
            self.y = y

        def display(self):
            print(f'Point Coordinates: ({self.x}, {self.y})')

class Location(Point):
    def __init__(self, x, y, dest_x, dest_y):
        super().__init__(x, y)
        self.destination = Point(dest_x, dest_y)

    def reflect_x(self):
        reflected = Point(self.destination.x, -self.destination.y)
        return reflected

    def display(self):
        super().display()
        print(f'Destination Coordinates: ({self.destination.x}, {self.destination.y})')
        reflected = self.reflect_x()
        print(f'Reflection on X-axis: ({reflected.x}, {reflected.y})')

# Example usage
loc = Location(0, 0, 4, 5)
loc.display()

Point Coordinates: (0, 0)
Destination Coordinates: (4, 5)
Reflection on X-axis: (4, -5)
```

In []:

Part ii: Student, Course, and Department

```
class Student:
```

```
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
```

```
    def display(self):
        print(f'Student Name: {self.name}, Roll No: {self.roll_no}')
```

```
class Course:
```

```
    def __init__(self, name, code, year, semester):
```

```
self.name = name
self.code = code
self.year = year
self.semester = semester
```

```
def display(self):
    print(f'Course Name: {self.name}, Code: {self.code}, Year: {self.year}, Semester: {self.semester}')
```

```
class Department:
```

```
    def __init__(self, name):
        self.name = name
        self.students = []
        self.courses = []
```

```
    def enroll_student(self, student):
        self.students.append(student)
```

```
    def add_course(self, course):
        self.courses.append(course)
```

```
    def display(self):
        print(f'Department: {self.name} ')
        print("Students:")
        for student in self.students:
            student.display()
        print("Courses:")
        for course in self.courses:
            course.display()
```

```
# Example usage
```

```
dept = Department("Computer Science")
student1 = Student("Alice", 101)
student2 = Student("Bob", 102)
```

```
course1 = Course("Python Programming", "CS101", 2024, "Semester 1")
course2 = Course("Data Structures", "CS102", 2024, "Semester 1")
```

```
dept.enroll_student(student1)
dept.enroll_student(student2)
dept.add_course(course1)
dept.add_course(course2)
```

```
dept.display()
```

OUTPUT:

Department: Computer Science
Students:
Student Name: Alice, Roll No: 101
Student Name: Bob, Roll No: 102
Courses:
Course Name: Python Programming, Code: CS101, Year: 2024, Semester: Semester 1
Course Name: Data Structures, Code: CS102, Year: 2024, Semester: Semester 1

In []:

EXPERIMENT 12

OBJECTIVE : Polymorphism, Error and Exception handling

- 1) Demonstrate operator overloading
- 2) Demonstrate Method Overriding
- 3) WAP to handle the divide by zero exception
- 4) Demonstrate Raise Exceptions, Instantiating Exceptions, assertion
- 5) WAP that prompts the use to enter a number and prints the square of that number. If no number is entered, then a Key Board Interrupt is generated
- 6) WAP which infinitely prints natural numbers. Raise the stop Iteration Exception after displaying first 20 numbers to exit from the program
- 7) WAP that randomly generates a number. Raise a User Defined exception if the number is below 0.1

THEORY: Polymorphism, Error, and Exception Handling

Polymorphism

Polymorphism refers to the ability of different classes to respond to the same function call in their unique way. It is achieved through:

Method Overloading (not natively supported in Python but achieved using default arguments).

Operator Overloading (redefining operators like +, *, etc., for user-defined classes).

Method Overriding (redefining a parent class method in a child class).

Error and Exception Handling

Errors disrupt the program's normal flow. Python provides a robust mechanism for handling exceptions:

Try-Except: Handles exceptions to prevent program crashes.

Raise: Allows the user to trigger exceptions manually.

Assertions: Ensure a condition is met; otherwise, an AssertionError is raised.

Custom Exceptions: User-defined exceptions for specific scenarios.

CODE:

#Demonstrate operator overloading

#Demonstrate Method Overriding

1. Operator Overloading

class ComplexNumber:

def __init__(self, real, imag):

self.real = real

```
self.imag = imag
```

```
def __add__(self, other):  
    return ComplexNumber(self.real + other.real, self.imag + other.imag)  
  
def __str__(self):  
    return f"{self.real} + {self.imag}i"
```

```
# Example usage
```

```
c1 = ComplexNumber(2, 3)  
c2 = ComplexNumber(4, 5)  
c3 = c1 + c2  
print(c3)
```

```
# 2. Method Overriding
```

```
class Animal:  
    def speak(self):  
        print("Animal speaks")
```

```
class Dog(Animal):  
    def speak(self):  
        print("Dog barks")
```

```
# Example usage
```

```
a = Animal()  
d = Dog()  
a.speak()  
d.speak()
```

OUTPUT:

```
In [14]: # 1. Operator Overloading
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return ComplexNumber(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

# Example usage
c1 = ComplexNumber(2, 3)
c2 = ComplexNumber(4, 5)
c3 = c1 + c2
print(c3)

# 2. Method Overriding
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

# Example usage
a = Animal()
d = Dog()
a.speak()
d.speak()

6 + 8i
Animal speaks
Dog barks
```

#WAP to handle the divide by zero exception

#Demonstrate Raise Exceptions, Instantiating Exceptions, assertion

#WAP that prompts the use to enter a number and prints the square of that number. If no number is entered, then a Key Board Interrupt is generated

#WAP which infinitely prints natural numbers. Raise the stop Iteration Exception after displaying first 20 numbers to exit from the program

#WAP that randomly generates a number. Raise a User Defined exception if the number is below 0.1

3. Handle Divide by Zero Exception

try:

```
a = int(input("Enter numerator: "))
```

```
b = int(input("Enter denominator: "))
```

```
result = a / b
```

```
print(f"Result: {result}")
```

except ZeroDivisionError:

```
print("Error: Division by zero is not allowed!")
```

4. Raise, Instantiate Exceptions, and Assertion

```
try:
    x = -1
    if x < 0:
        raise ValueError("Value cannot be negative!")
except ValueError as e:
    print(e)
```

Assertion

```
assert 5 > 0, "Number must be greater than zero!"
```

5. Handle KeyboardInterrupt

```
try:
    num = int(input("Enter a number: "))
    print(f"Square of the number: {num ** 2}")
except KeyboardInterrupt:
    print("\nKeyboardInterrupt: No input received.")
```

6. StopIteration Exception

```
class NaturalNumbers:
    def __init__(self):
        self.num = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.num < 20:
            self.num += 1
            return self.num
        else:
            raise StopIteration
```

Example usage

```
for n in NaturalNumbers():
    print(n)
```

7. UserDefined Exception


```
import random
```

```
class NumberTooSmallError(Exception):
```

```
    pass
```

```
try:
```

```
    num = random.random()
```

```
    print(f'Generated number: {num}')
```

```
    if num < 0.1:
```

```
        raise NumberTooSmallError("The number is too small!")
```

```
except NumberTooSmallError as e:
```

```
    print(e)
```

OUTPUT:

```
Enter numerator: 30
Enter denominator: 10
Result: 3.0
Value cannot be negative!
Enter a number: 34
Square of the number: 1156
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Generated number: 0.705926667156578
```

In []: