CONTENTS

S. NO.	EXPERIMENT	DATE OF SUBMISSION	PAGE NO.	REMARKS
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	6	
3.	Demonstrate conditional and loop statements and develop code for given problem statements	02 SEPT 2024	10	
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

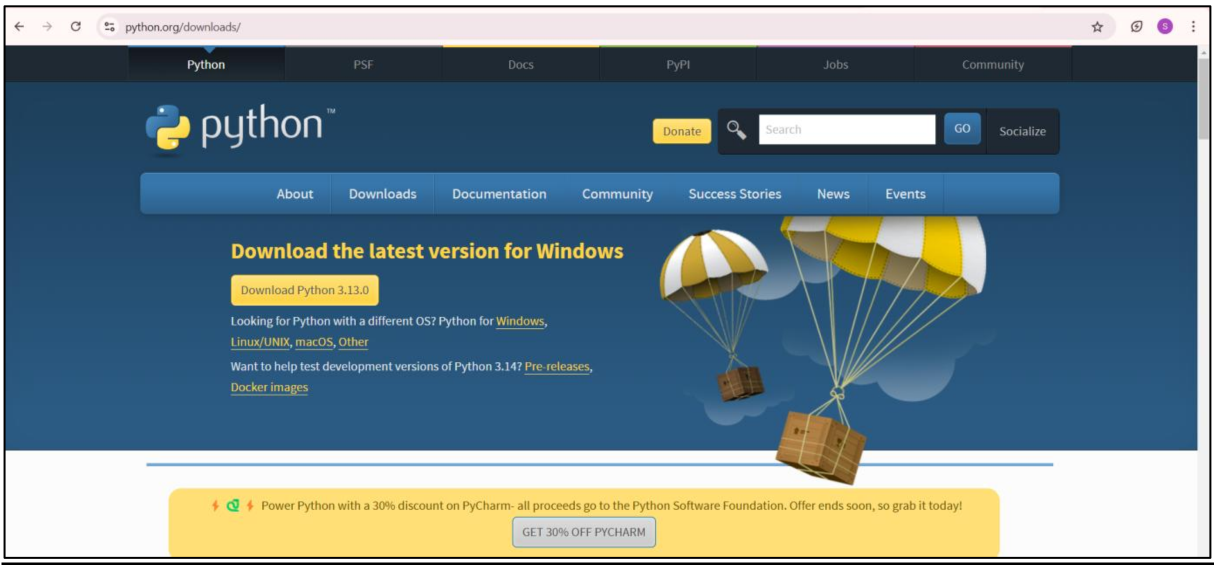
OBJECTIVE: Install Python and write basic programs to explore its syntax and functionality.

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).



Download the latest version for Windows

Download Python 3.13.0

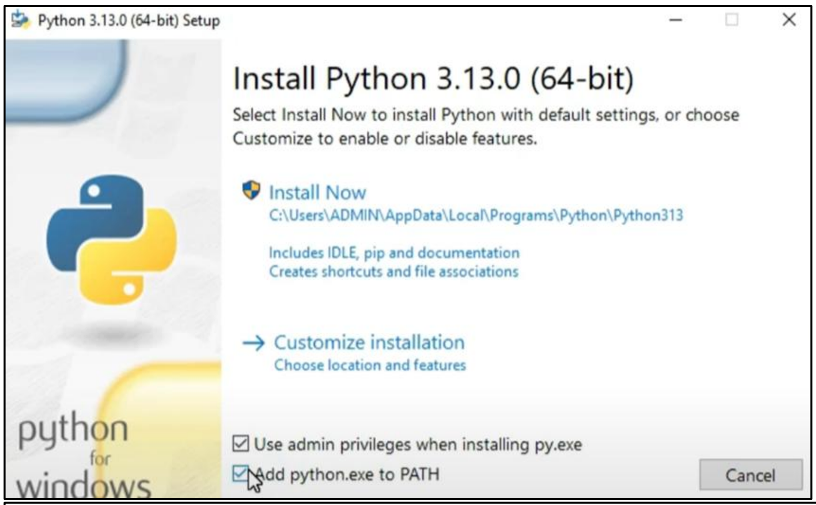
Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.14? Pre-releases, Docker images

Power Python with a 30% discount on PyCharm- all proceeds go to the Python Software Foundation. Offer ends soon, so grab it today!

GET 30% OFF PYCHARM

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.


Python 3.13.0 (64-bit) Setup

Install Python 3.13.0 (64-bit)

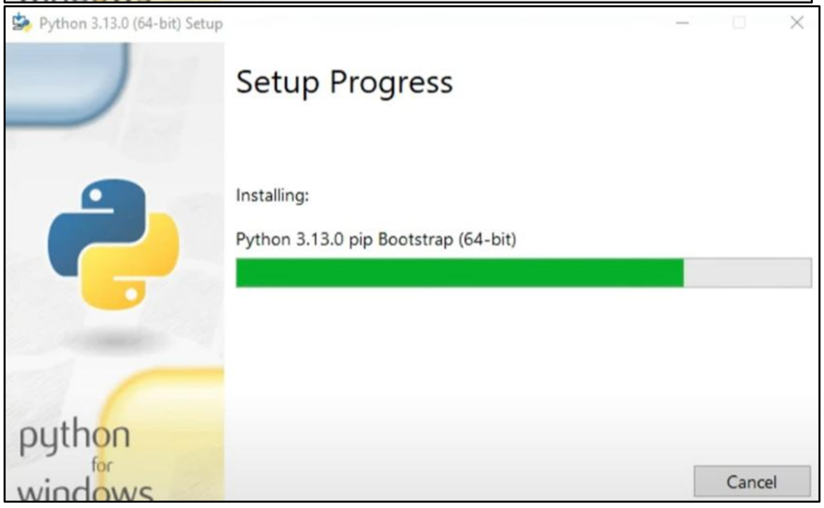
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\ADMIN\AppData\Local\Programs\Python\Python313

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

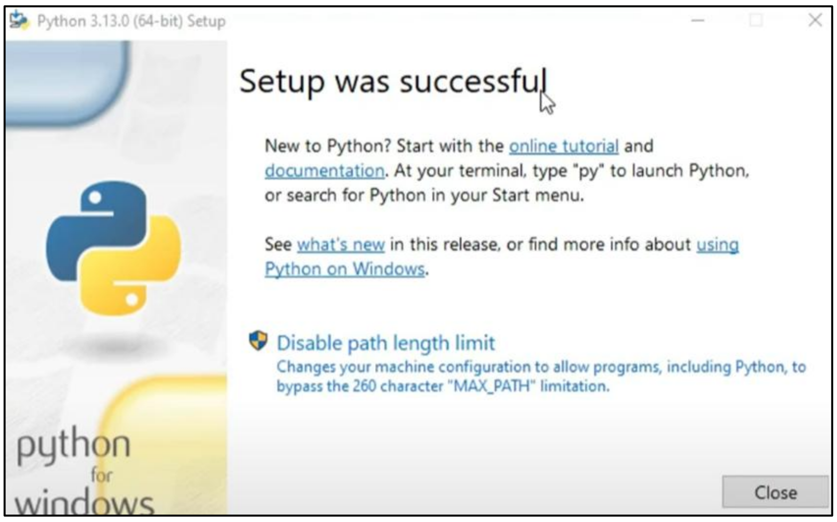

Python 3.13.0 (64-bit) Setup

Setup Progress

Installing:

Python 3.13.0 pip Bootstrap (64-bit)

3. Verify Installation:


Python 3.13.0 (64-bit) Setup

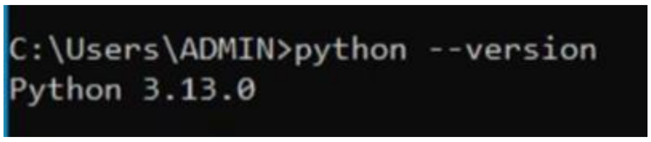
Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

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

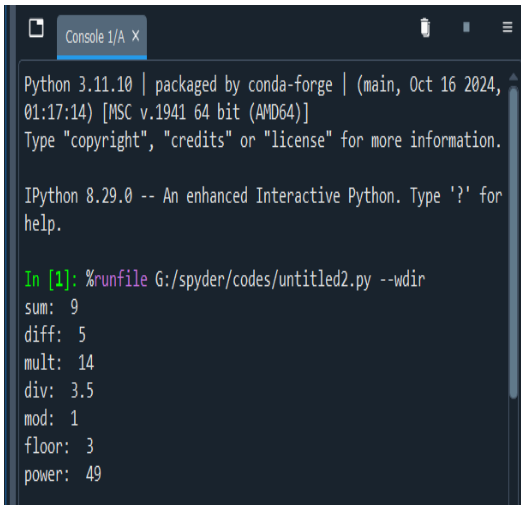


```
C:\Users\ADMIN>python --version
Python 3.13.0
```

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:



```
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(b)
print(type(b))

# Find hex value of given int
b = hex(56)
print(b)
print(type(b))

# Convert int to octal
b = oct(56)
print(b)
print(type(b))
```

2. Convert string to tuple, set and list

```
x = 'javaTpoint'
y=tuple(x)
print("after converting the string to a tuple: ", end="")
print(y)

y = set(x)
print("after converting the string to a set: ", end="")
print(y)

y = list(x)
print("after converting the string to a list: ", end="")
print(y)
```

3. Perform arithmetic operations on 2 numbers

```
# Arithmetic operators in python
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)
```

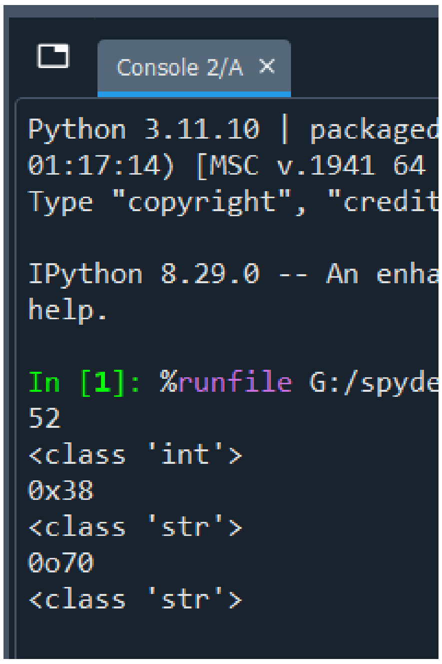
4. Demonstrate use of comparison, logical, identity, membership operators

```
# Comparison Operators
a=5
b=2
print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a<=b)
print(a>=b)

# Logical Operators
a=5
b=6
print((a>2) and (b>=6))
print((a>2) or (b>=6))

# Identity operators
```

```
x1=5
y1=5
x2='Hello'
y2='Hello'
x3=[1,2,3]
y3=[1,2,3]
print(x1 is not y1)
print(x2 is y2)
print(x3 is y3)
```

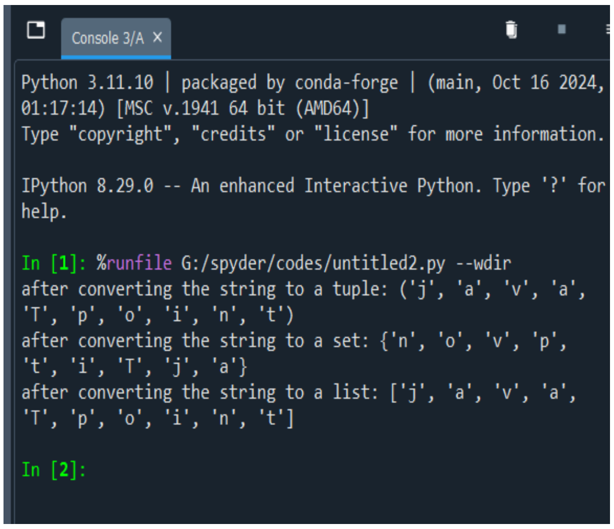
<u>RESULTS</u>:

CODE 1



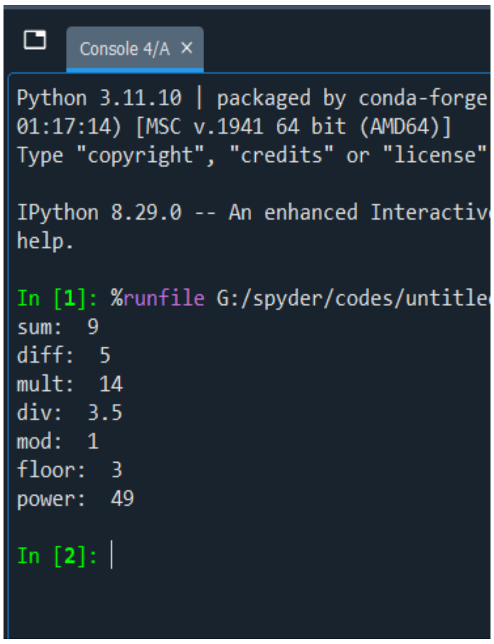
```
Python 3.11.10 | packaged
01:17:14) [MSC v.1941 64
Type "copyright", "credit

IPython 8.29.0 -- An enha
help.

In [1]: %runfile G:/spyde
52
<class 'int'>
0x38
<class 'str'>
0o70
<class 'str'>
```

CODE 2



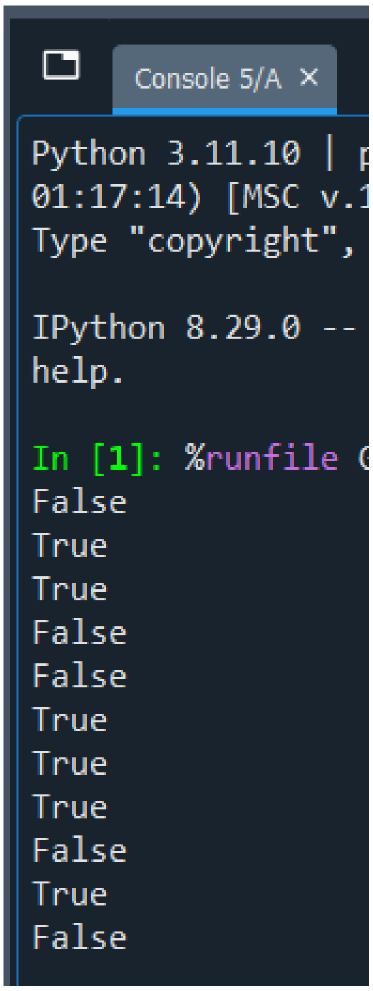
```
Python 3.11.10 | packaged by conda-forge | (main, Oct 16 2024,
01:17:14) [MSC v.1941 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

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

In [1]: %runfile G:/spyder/codes/untitled2.py --wdir
after converting the string to a tuple: ('j', 'a', 'v', 'a',
'T', 'p', 'o', 'i', 'n', 't')
after converting the string to a set: {'n', 'o', 'v', 'p',
't', 'i', 'T', 'j', 'a'}
after converting the string to a list: ['j', 'a', 'v', 'a',
'T', 'p', 'o', 'i', 'n', 't']

In [2]:
```

CODE 3



```
Python 3.11.10 | packaged by conda-forge
01:17:14) [MSC v.1941 64 bit (AMD64)]
Type "copyright", "credits" or "license"

IPython 8.29.0 -- An enhanced Interactiv
help.

In [1]: %runfile G:/spyder/codes/untitle
sum:  9
diff:  5
mult:  14
div:  3.5
mod:  1
floor:  3
power:  49

In [2]: 
```

CODE 4



```
Python 3.11.10 | p
01:17:14) [MSC v.1
Type "copyright",

IPython 8.29.0 --
help.

In [1]: %runfile G
False
True
True
False
False
True
True
True
False
True
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

```
1
2   2
3   3   3
4   4   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 + ... + 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:

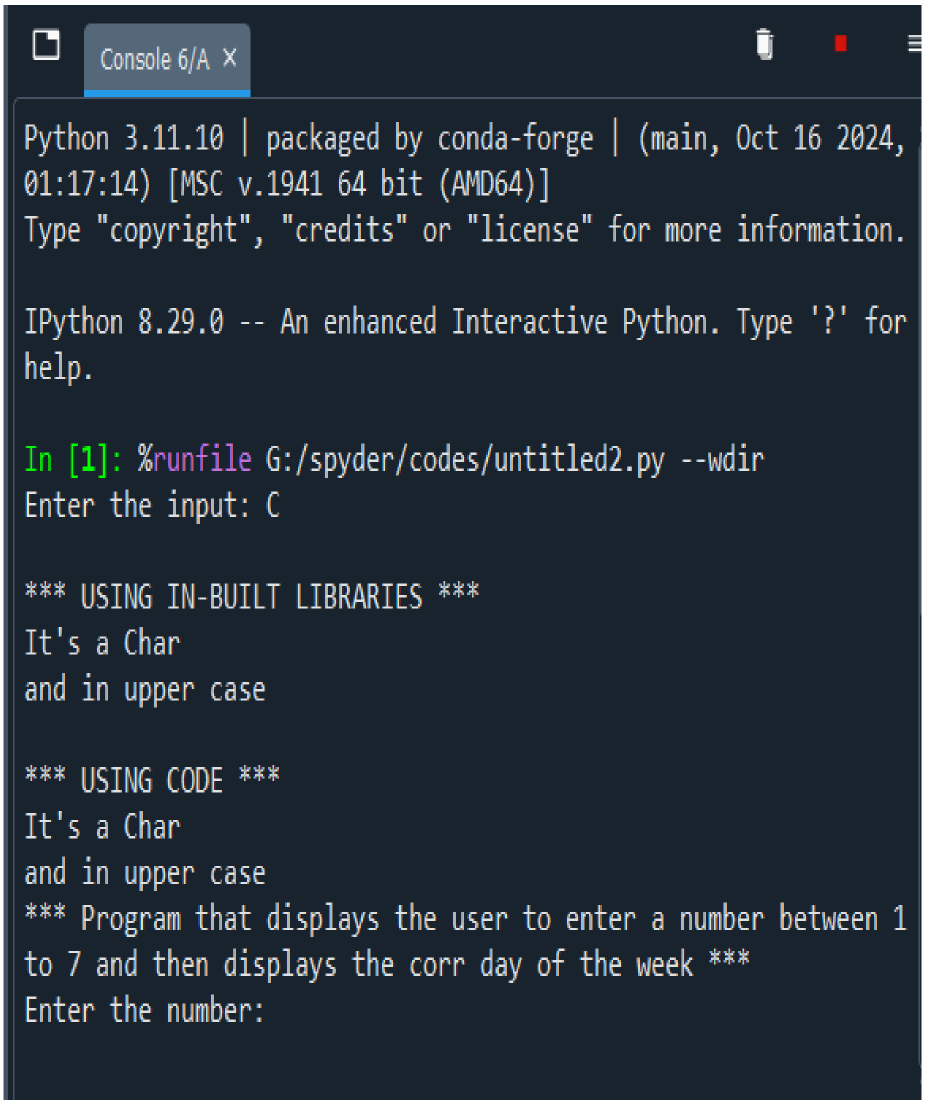
```
# WAP to take input from a user and then check whether it is a number or a character.
# If it is a char, determine whether it is Upper case or lower case

inp = input("Enter the input: ")
''' USING IN-BUILT LIBRARIES '''
print()
print("*** USING IN-BUILT LIBRARIES ***")
if (inp.isalpha()):
    print("It's a Char")
    if inp.isupper():
        print("and in upper case")
    elif inp.islower():
        print("and in lower case")
    else:
        print("and has both cases")
elif(inp.isnumeric()):
    print("It's a number")
else:
    print("Invalid Input")


''' ALTERNATE APPROACH '''
print()
print("*** USING CODE ***")
l1 = [0,0,0] #It will have 3 elements. First is No. of upper case char, second is no. of lower case chars, third is no. of integers
len1 = len(inp)
flag = 0
for i in inp:
    in_ascii = ord(i)
    if in_ascii in range(65,91) or in_ascii in range(97, 123):
        flag = 1
        if in_ascii in range(65,91):
            l1[0] +=1
        else:
            l1[1] +=1
    elif in_ascii in range(48, 58):
        flag = 2
        l1[2] +=1
if flag == 1:
    if l1[0] == len1:
        print("It's a Char")
        print("and in upper case")
    elif l1[1] == len1:
        print("It's a Char")
        print("and in lower case")
    elif l1[0]+l1[1] == len1:
        print("It's a Char")
        print("and has both cases")
    else:
        print("Invalid Input")
elif flag == 2:
    if l1[2] == len1:
        print("It's a number")
    else:
        print("Invalid Input")
else:
        print("Invalid Input")
```

```
# 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 ***")
num = int(input("Enter the number: "))
if num >= 1 and num <= 7:
    if num == 1:
        print ("Monday")
    if num == 2:
        print ("Tuesday")
    if num == 3:
        print ("Wednesday")
    if num == 4:
        print ("Thursday")
    if num == 5:
        print ("Friday")
    if num == 6:
        print ("Saturday")
    if num == 7:
        print ("Sunday")
else:
    print("Incorrect number")
```

```
# Nested loop to print pattern
for i in range(1,6):
    for j in range(1, i+1):
        print("*", end = " ")
    print()


# While loop inside for loop
names = ["Kelly", "Jessa", "Emma"]
for name in names:
    count = 0
    while(count<5):
        print(name, end=' ')
        count+=1
    print()



# WAP to print the pattern
for i in range(1, 6):
    for k in range(1, 6-i):
        print(" ", end=" ")
    for j in range(1,i+1):
        print(i, " ", end=" ")

    print()

# Alternate approach
n=5
for i in range(1, n+1):
    for k in range(n, i, -1):
        print(" ", end=" ")
    for j in range(1,i+1):
        print(i, " ", end=" ")
    print()

# Calculating factorial
fact = 1
for i in range(2,n+1):
    fact *= i
print("Factorial is: ", fact)


# WAP that displays all leap years from 1900 to 2101
year = int(input("Enter the year (1900-2101) to check whether leap year: "))
if year%100 == 0:
    if year%400 == 0:
        print("Leap year")
    else:
        print("Not leap year")
else:
    if year%4 == 0:
        print("Leap year")
    else:
        print("Not leap year")
```

```
# WAP to sum the series numbers - 1 + 1/2 + ... + 1/n using for loop
n = int(input("Enter the number: "))
s = 0
for i in range(1, n+1):
    s += (1/i)
print("Sum of series is: ", s)
```

RESULTS:



Console 6/A

```
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
Enter the input: C

*** USING IN-BUILT LIBRARIES ***
It's a Char
and in upper case

*** USING CODE ***
It's a Char
and in upper case
*** Program that displays the user to enter a number between 1
to 7 and then displays the corr day of the week ***
Enter the number:
```

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.

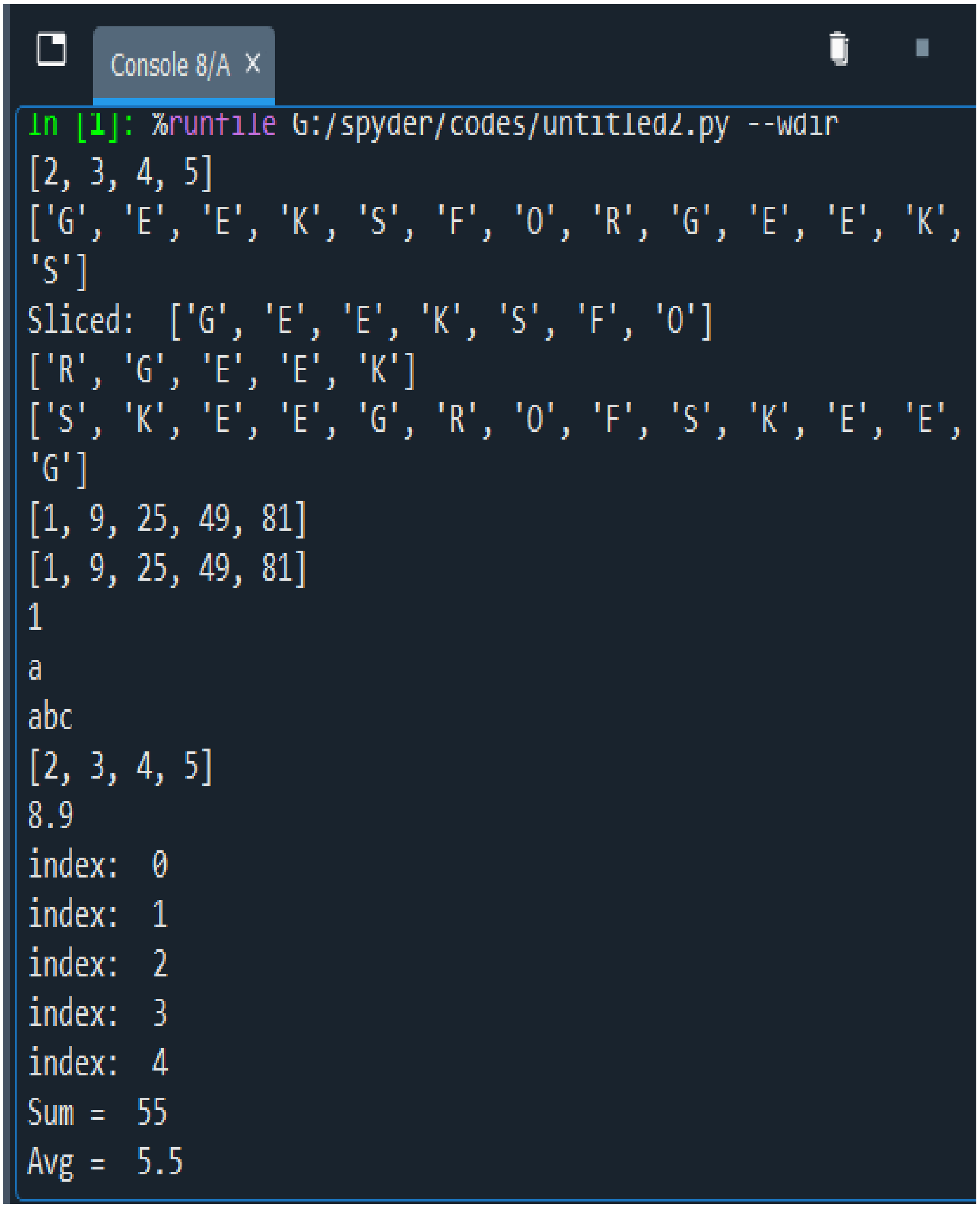
CODE:

```
# List slicing

list1 = ['physics', 'chem', 1997, 2000]

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

print(list2[1:5])


# List methods- insert, append, extend, reverse, reversed, remove, pop, slicing,

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

```
print(List)
Sliced_list = List[:-6]
print("Sliced: ", Sliced_list)
l2 = List[-6:-1]
print(l2)
l3 = List[::-1]
print(l3)

# List Comprehension
# Syntax - [expression(element) for element in oddList if condition]
l1 = [x**2 for x in range(1,11) if x%2 == 1]
print(l1)

# List Comprehension
# Syntax - [expression(element) for element in oddList if condition]
l1 = [x**2 for x in range(1,11) if x%2 == 1]
print(l1)

# Looping in lists
ls = [1,'a',"abc",[2,3,4,5],8.9]
i = 0
while i < (len(ls)):
    print(ls[i])
    i+=1
# Program to print index of values in a list
l1 = [1,2,3,4,5]
for i in range(len(l1)):
    print("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("Sum = ", s)
print("Avg = ", s/len(l1))
```

RESULTS:



```
In [1]: %runfile G:/spyder/codes/untitled2.py --wdir
[2, 3, 4, 5]
['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K',
'S']
Sliced:  ['G', 'E', 'E', 'K', 'S', 'F', 'O']
['R', 'G', 'E', 'E', 'K']
['S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E',
'G']
[1, 9, 25, 49, 81]
[1, 9, 25, 49, 81]
1
a
abc
[2, 3, 4, 5]
8.9
index:  0
index:  1
index:  2
index:  3
index:  4
Sum =  55
Avg =  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:

```
# Creating array in python
import array as arr
a = arr.array('i', [1,2,3])
print(a)
for i in range(0,3):
   print(a[i], end=" ")
```

```
# Demonstrate the functions in arrays like insert(), append()
a = arr.array('i', [1,2,3])
print("Array of integers (Before): ", a)
a.insert(1,4)
print("Array of integers (After Inserting): ",a)
b = arr.array('d', [1,2,3])
print("Array of floats (Before): ", b)
b.append(4.4)
print("Array of floats (After appending): ", b)

# Slicing
import array as arr
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
a = arr.array('i', l)
print("Initial Array: ")
for i in (a):
   print(i, end = " ")
sliced_array = a[3:8]
print("\nSlicing elements in a range 3-8: ")
print(sliced_array)
sliced_array = a[5:]
print("\nElements sliced from 5th element till the end: ")
print(sliced_array)
sliced_array=a[:]
print("\nPrinting all elements using slice operartion: ")
print(sliced_array)

# Array Updation
import array
arr = array.array('i', [1,2,3,1,2,5])
for i in range(0,6):
   print(arr[i], end = " ")
print("\nAfter updation")
arr[2]=6
for i in range(0,6):
   print(arr[i], end=" ")

# Create empty tuple:
tuple1 = ()
print(tuple1)

# Create tuple using string:
tuple1 = ('Hello', 'Sam')
print(tuple1)

# Create tuple using list:
```

```
list1 = ['Hello', 'Sam']
print(tuple(list1))

# Create a tuple using built-in function:
tuple1 = tuple('Sam')
print(tuple1)

# Creating a tuple with mixed datatypes
tuple1 = (5, 'aiojdio', 7, 'JFidsof')
print(tuple1)

# Nested tuples

t1 = (1,2,3)
t2 = ('a', 'b', 'c')
t3 = (t1, t2)
print(t3)

# Program to demonstrate use of nested tuples

Toppers = (("arav", 97, "B.Sc."), ("raghav", 87, "BCA"))
for i in Toppers:
    print(i)

# WAP that has a nested list to store toppers details. Edit the details and reprint the details.
# Eg - l1 = ["Arav", "MSC", 92]

l1 = [["Arav", "MSC", 92], ["Student2", "MBA", 99], ["Student3", "MTech", 94],
["Student4", "BSC", 95]]

print("The original list of toppers is: ", l1)
print("Enter the metadata you wish to edit: ")
print("\nChoose the name of the student you wish to edit the details for. Press")
for i in range(len(l1)):
    print(f'{i}. To edit the details of student {l1[i][0]}')
ch1 = int(input("Enter your choice: "))

print("Press\n1. To edit the name\n2. To edit the branch\n3. To edit the marks")
ch2 = int(input("Enter your choice (1/2/3): "))

if ch1 not in range(len(l1)):
    print("Wrong Student index chosen!")
else:

    if ch2 == 1:
        new_name = input("Enter the new name: ")
        l1[ch1][0] = new_name
    elif ch2 == 1:
        new_name = input("Enter the new branch: ")
        l1[ch1][1] = new_name
    elif ch2 == 1:
        new_name = input("Enter the new marks: ")
        l1[ch1][2] = new_name
    else:
        print("Wrong choice entered!")

    print("New list is: ", l1)
```

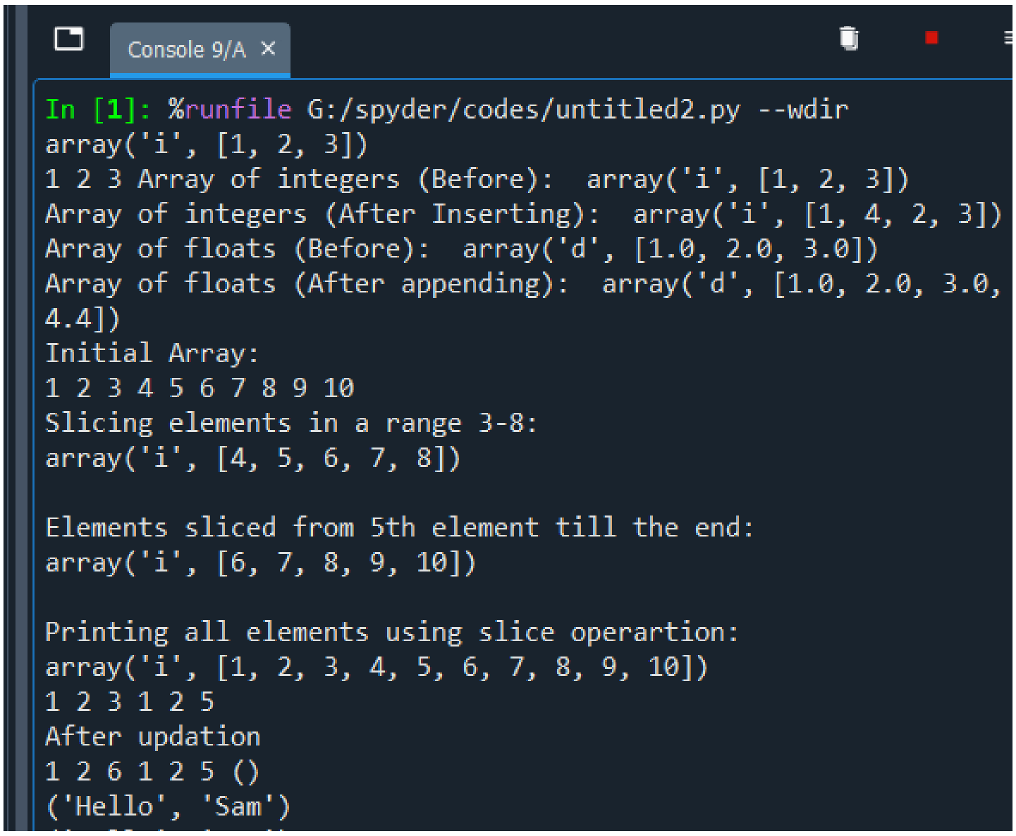
```
# Creating a tuple using Loop
t1 = ('Sam')
n = 5
for i in range(int(n)):
    t1 = (t1,)
    print(t1)
```

```
# WAP to swap two values using tuple assignment
t1 = (2,3)
print("Tuple is: ", t1)
print("Before swap: ")
a, b = t1
print(f'Value of a is {a} and value of b is {b}')
print("After swap: ")
(a, b) = (b, a)
print(f'Value of a is {a} and value of b is {b}')
```

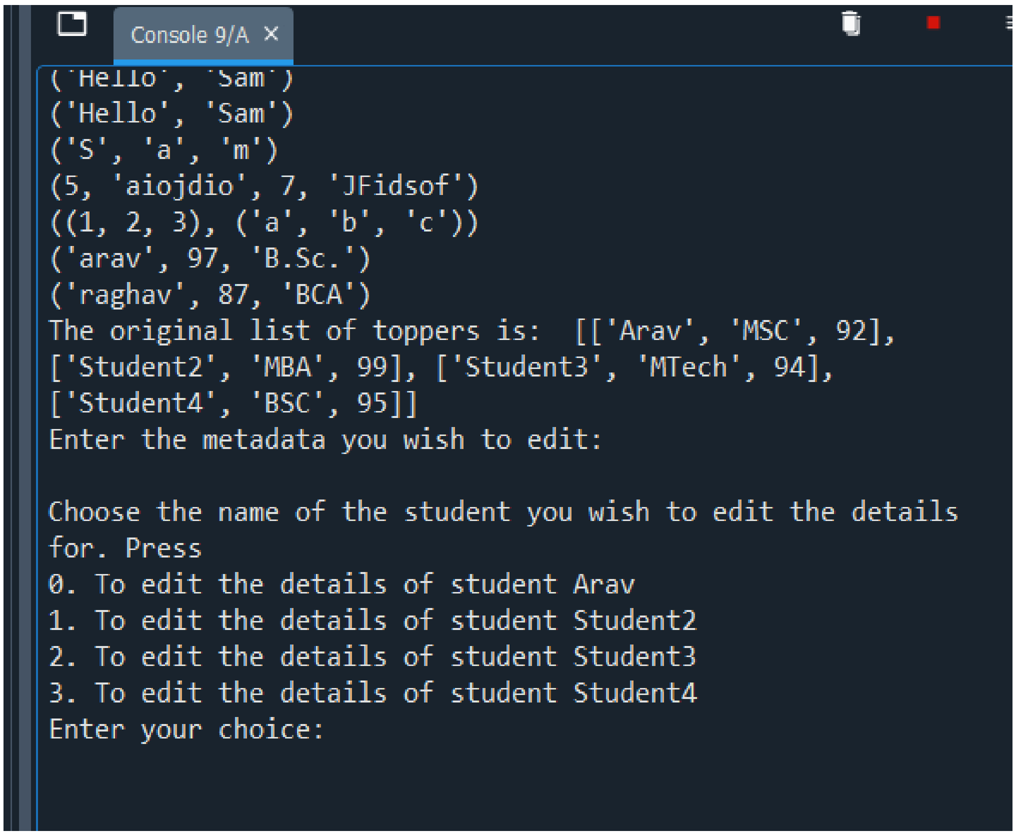
```
# WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument
import math
def func1(r):
    area = math.pi * r * r
    circum = 2 *math.pi *r
    return (area, circum)
rad = int(input("Enter radius: "))
(ar, circum) = func1(rad)
print("Area is: ", ar)
print("Circumference is: ", circum)
```

```
# WAP that scans an email address and forms a tuple of username and domain
email = input("Enter the email address: ")
email = email.split("@")
email_tuple = tuple(email)
print(email_tuple)
```

RESULTS:



```
In [1]: %runfile G:/spyder/codes/untitled2.py --wdir
array('i', [1, 2, 3])
1 2 3 Array of integers (Before):  array('i', [1, 2, 3])
Array of integers (After Inserting):  array('i', [1, 4, 2, 3])
Array of floats (Before):  array('d', [1.0, 2.0, 3.0])
Array of floats (After appending):  array('d', [1.0, 2.0, 3.0,
4.4])
Initial Array:
1 2 3 4 5 6 7 8 9 10
Slicing elements in a range 3-8:
array('i', [4, 5, 6, 7, 8])

Elements sliced from 5th element till the end:
array('i', [6, 7, 8, 9, 10])

Printing all elements using slice operartion:
array('i', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
1 2 3 1 2 5
After updation
1 2 6 1 2 5 ()
('Hello', 'Sam')
```



```
('Hello', 'Sam')
('Hello', 'Sam')
('S', 'a', 'm')
(5, 'aiojdio', 7, 'JFidsof')
((1, 2, 3), ('a', 'b', 'c'))
('arav', 97, 'B.Sc.')
('raghav', 87, 'BCA')
The original list of toppers is:  [['Arav', 'MSC', 92],
['Student2', 'MBA', 99], ['Student3', 'MTech', 94],
['Student4', 'BSC', 95]]
Enter the metadata you wish to edit:

Choose the name of the student you wish to edit the details
for. Press
0. To edit the details of student Arav
1. To edit the details of student Student2
2. To edit the details of student Student3
3. To edit the details of student Student4
Enter your choice:
```

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 and modules are essential concepts in Python programming, promoting code reusability, modularity, and readability. Below is an explanation with examples to demonstrate their use.

- A function is a block of reusable code that performs a specific task. It is defined using the def keyword.
- A module is a file containing Python code (functions, classes, or variables) that can be imported and reused in other programs. Python comes with a rich standard library of modules, and you can also create your own.

Benefits of Functions and Modules

1. Code Reusability: Write once, use many times.
2. Modularity: Divide large programs into smaller, manageable parts.
3. Readability: Easier to read and maintain.
4. Collaboration: Multiple developers can work on different modules.

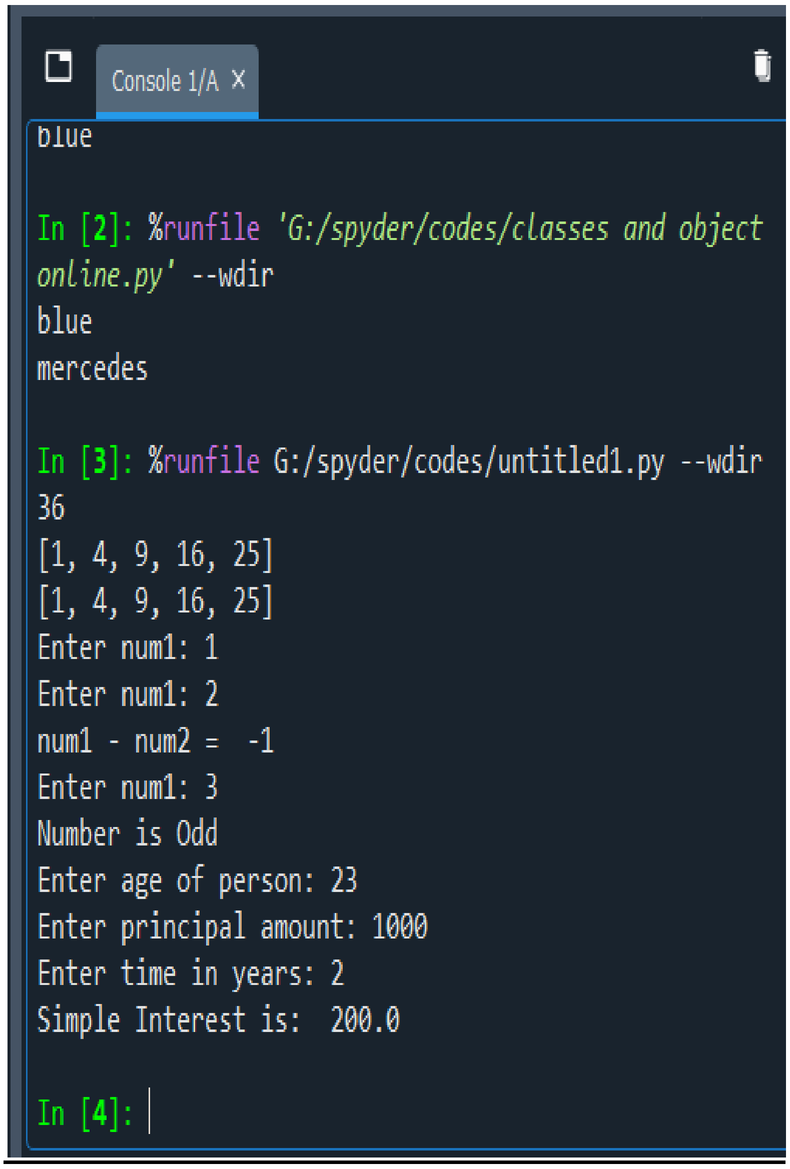
By combining functions and modules, Python programs can be structured effectively, enabling efficient development.

CODE:

```
# Defining the function
def square(num):
#     Returns the square of the number
   return num**2

obj = square(6)
print(obj)

# Pass by Reference and Pass by value

def square(item_list):
#     Returns the square of the number
```

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

# Pass by reference
num = [1,2,3,4,5]
obj = square(num)
print(obj)

# Pass by value
obj = square([1,2,3,4,5])
print(obj)

# WAP that subtracts two numbers using a function
def func(a,b):
    return a - b
a = int(input("Enter num1: "))
b =  int(input("Enter num1: "))
print("num1 - num2 = ", func(a,b))

# 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))

# 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%.
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)

# 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)
```

RESULTS:



```
blue

In [2]: %runfile 'G:/spyder/codes/classes and object
online.py' --wdir
blue
mercedes

In [3]: %runfile G:/spyder/codes/untitled1.py --wdir
36
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]
Enter num1: 1
Enter num1: 2
num1 - num2 =  -1
Enter num1: 3
Number is Odd
Enter age of person: 23
Enter principal amount: 1000
Enter time in years: 2
Simple Interest is:  200.0

In [4]:
```

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:

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.

1. Creating Sets
   Sets are created using curly braces {} or the set() constructor.
2. Union
   The union of two sets contains all unique elements from both sets
3. Intersection
   The intersection of two sets contains only the elements that are common to both.
4. Difference
   The difference of two sets contains the elements of the first set that are not in the second set.
5. Symmetric Difference
   The symmetric difference of two sets contains elements that are in either of the sets, but not in both.
6. Subset and Superset
    - A set is a subset of another if all its elements are contained in the other.
    - A set is a superset of another if it contains all elements of the other.

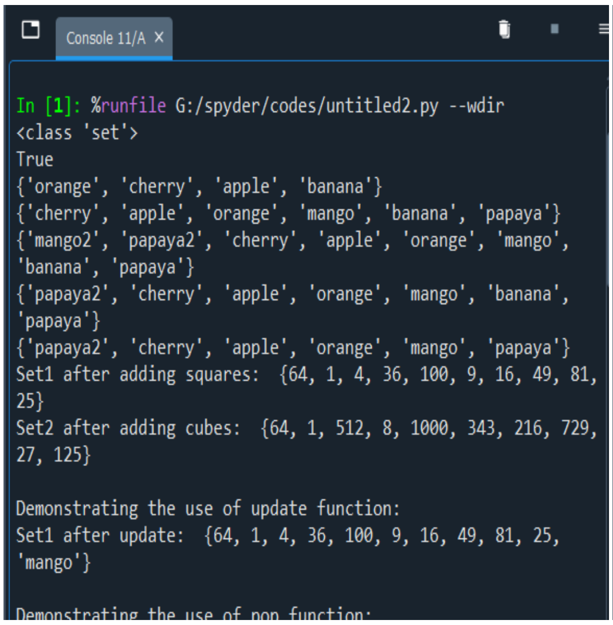
CODE:

```
# SETS
thisset = {"apple", "banana", "cherry"}
print(type(thisset))
print("banana" in thisset)

# Add items in set
thisset.add("orange")
print(thisset)

# Add items from another set into this set
tropical = {"mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

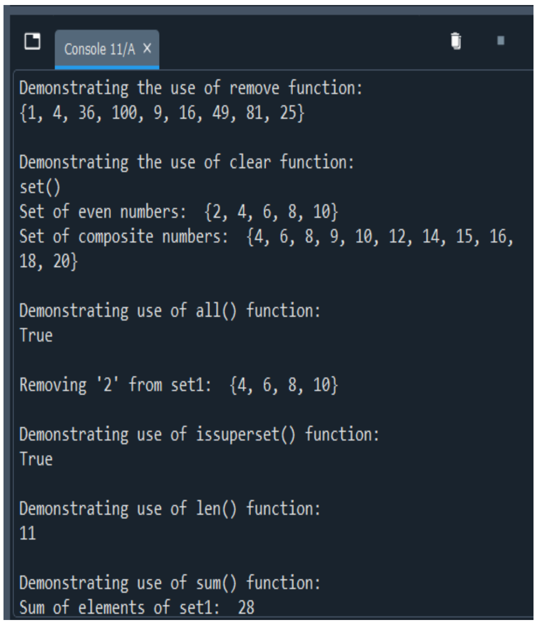
```
# Add elements of a list to the set
l1 = ["mango2", "papaya2"]
thisset.update(l1)
print(thisset)

# Remove item
thisset.remove("mango2")
print(thisset)

# Remove item using discard()
thisset.discard("banana")
print(thisset)

# WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update,
pop, remove and clear function
set1 = set()
set2 = set()
for i in range(1, 11):
   set1.add(i*i)
   set2.add(i*i*i)
print("Set1 after adding squares: ", set1)
print("Set2 after adding cubes: ", set2)

print("\nDemonstrating the use of update function: ")
set3 = {"mango"}
set1.update(set3)
print("Set1 after update: ", set1)

print("\nDemonstrating the use of pop function: ")
print(set1.pop())

print("\nDemonstrating the use of remove function: ")
set1.remove("mango")
print(set1)

print("\nDemonstrating the use of clear function: ")
set1.clear()
print(set1)

# 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.

set1 = {i for i in range(1, 11) if i % 2 == 0 }
print("Set of even numbers: ",set1)

set2 = set()
```

```
c = 0
for i in range(2, 21):
    for j in range(2, i):
        if i%j ==0:
            c+=1
    if c!=0:
        set2.add(i)
    c = 0
print("Set of composite numbers: ", set2)

# all() function returns True if all elements are True, else returns False
print("\nDemonstrating use of all() function: ")
print(all(set1))

set1.remove(2)
print("\nRemoving '2' from set1: ", set1)

print("\nDemonstrating use of issuperset() function: ")
print(set2.issuperset(set1))

print("\nDemonstrating use of len() function: ")
print(len(set2))

print("\nDemonstrating use of sum() function: ")
print("Sum of elements of set1: ", sum(set1))
```

RESULTS:



```
In [1]: %runfile G:/spyder/codes/untitled2.py --wdir
<class 'set'>
True
{'orange', 'cherry', 'apple', 'banana'}
{'cherry', 'apple', 'orange', 'mango', 'banana', 'papaya'}
{'mango2', 'papaya2', 'cherry', 'apple', 'orange', 'mango',
'banana', 'papaya'}
{'papaya2', 'cherry', 'apple', 'orange', 'mango', 'banana',
'papaya'}
{'papaya2', 'cherry', 'apple', 'orange', 'mango', 'papaya'}
Set1 after adding squares:  {64, 1, 4, 36, 100, 9, 16, 49, 81,
25}
Set2 after adding cubes:  {64, 1, 512, 8, 1000, 343, 216, 729,
27, 125}

Demonstrating the use of update function:
Set1 after update:  {64, 1, 4, 36, 100, 9, 16, 49, 81, 25,
'mango'}

Demonstrating the use of pop function:
```



```
Demonstrating the use of remove function:
{1, 4, 36, 100, 9, 16, 49, 81, 25}

Demonstrating the use of clear function:
set()
Set of even numbers:  {2, 4, 6, 8, 10}
Set of composite numbers:  {4, 6, 8, 9, 10, 12, 14, 15, 16,
18, 20}

Demonstrating use of all() function:
True

Removing '2' from set1:  {4, 6, 8, 10}

Demonstrating use of issuperset() function:
True

Demonstrating use of len() function:
11

Demonstrating use of sum() function:
Sum of elements of set1:  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

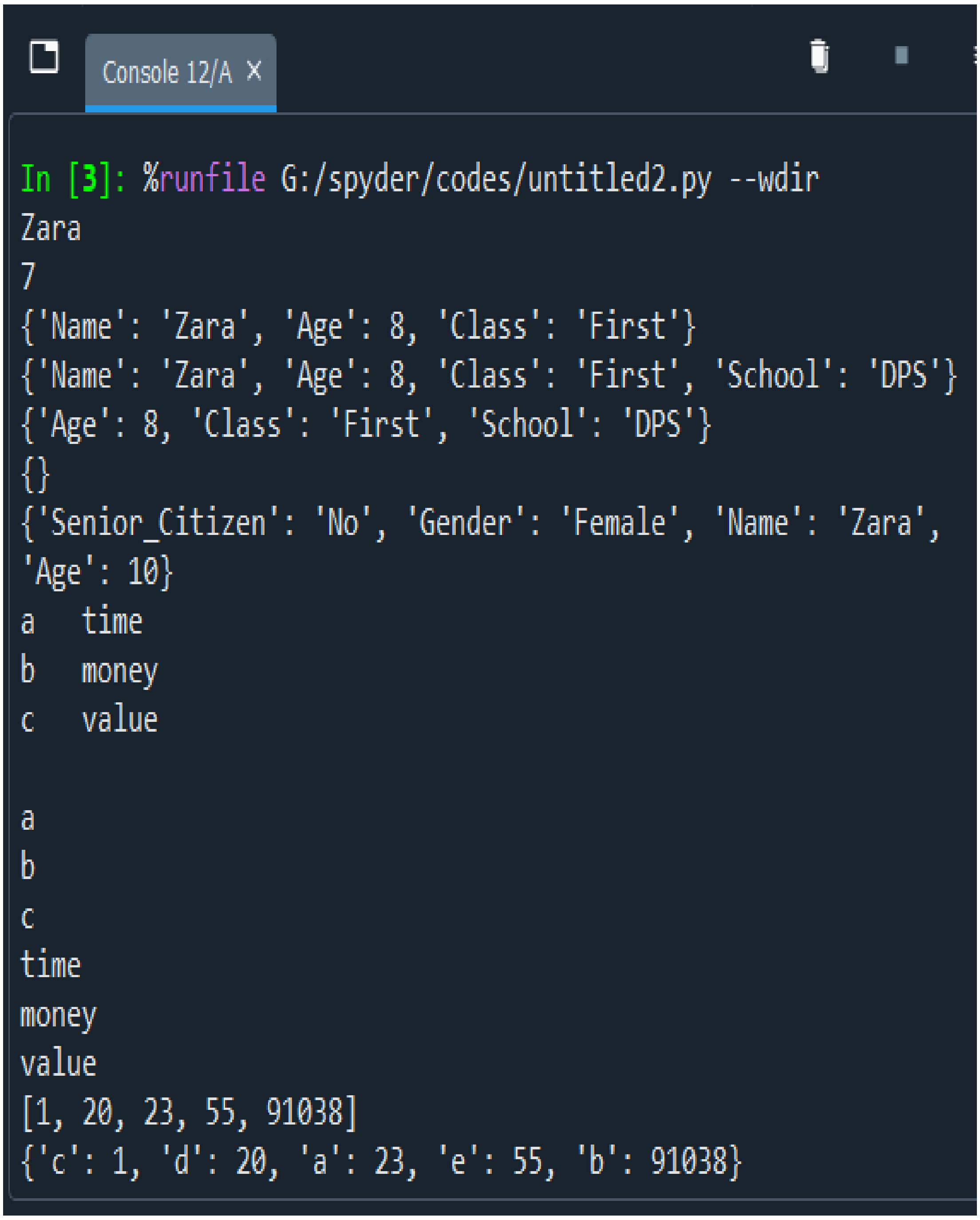
1. **Accessing Values in a Dictionary**:
   You can access a value in a dictionary by referring to its key, e.g., dict[key]. You can also use the get() method to avoid errors when a key is missing.
2. **Updating a Dictionary**:
   You can add or modify an entry by assigning a value to a key, e.g., dict[key] = 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': 'Zara', 'Age': 7, 'Class': 'First'}
print(dict1['Name'])
print(dict1['Age'])

# Updating a dict
dict1['Age'] = 8
print(dict1)
```

```
# Add a new entry
dict1['School'] = 'DPS'
print(dict1)

# Delete entries
del dict1['Name']
print(dict1)

# Clear whole dict
dict1.clear()
print(dict1)

# Delete whole dict
del dict1
print(dict1)

# WAP to merge two dictionaries with a third one
a = {'Name': 'Zara', 'Age': 10}
b = {'Gender': 'Female'}
c = {'Senior_Citizen': 'No'}
c.update(b)
c.update(a)
print(c)

# Iterating through a dictionary
dict1 = {"a": "time", "b": "money", "c": "value"}
for key, values in dict1.items():
    print(key, " ", values)
print()
for i in dict1.keys():
    print(i)
for i in dict1.values():
    print(i)

# Sort dictionary by values
dict1 = {"a": 23, "b": 91038, "c": 1, "d": 20, "e": 55}
# print(sorted(dict1, key = dict1.values))
print(dict1)
ls = sorted(dict1.values())
print(ls)
dict2 = {}
for i in ls:
    for j in dict1.keys():
        if dict1.get(j) == i:
            dict2[j] = i
print(dict2)
```

RESULTS:



```
In [3]: %runfile G:/spyder/codes/untitled2.py --wdir
Zara
7
{'Name': 'Zara', 'Age': 8, 'Class': 'First'}
{'Name': 'Zara', 'Age': 8, 'Class': 'First', 'School': 'DPS'}
{'Age': 8, 'Class': 'First', 'School': 'DPS'}
{}
{'Senior_Citizen': 'No', 'Gender': 'Female', 'Name': 'Zara',
'Age': 10}
a    time
b    money
c    value

a
b
c
time
money
value
[1, 20, 23, 55, 91038]
{'c': 1, 'd': 20, 'a': 23, 'e': 55, 'b': 91038}
```

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:

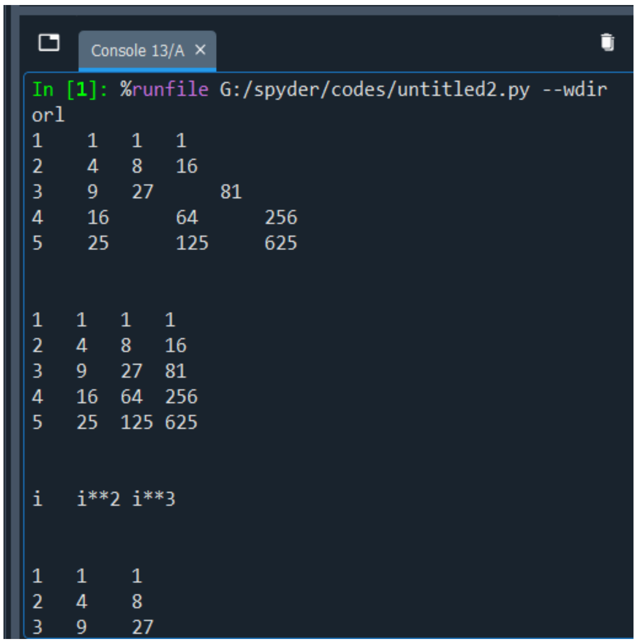
- Message encryption can be done by shifting the ASCII value of each character by a specified key, effectively "encrypting" the message.

CODE:

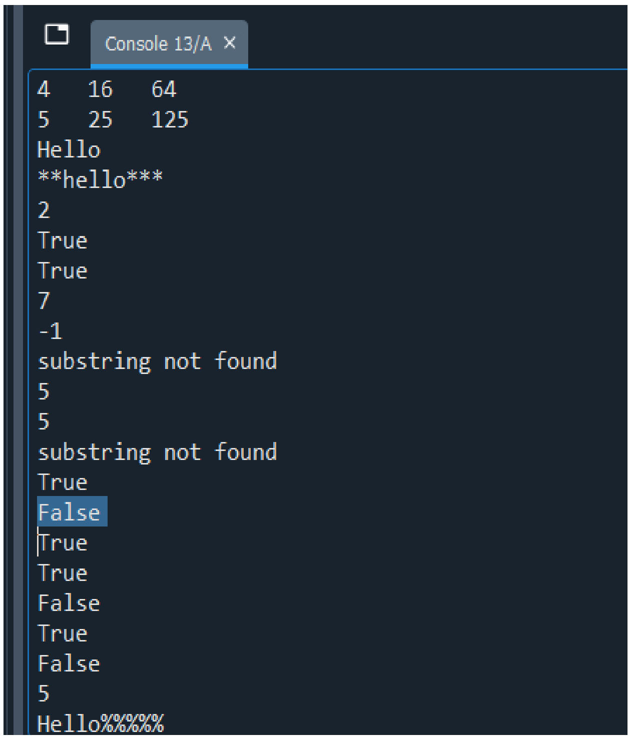
```
a = "HelloWorld"
# Get the characters from o in World to but not included d in "World"
print(a[-4:-1])

# WAP to display powers of number without using formatting characters
i=1
while i<=5:
   print(i**1, "\t", i**2, "\t",  i**3, "\t",  i**4)
   i+=1
print()
print()

i=1
while i<=5:
   print("%d\t%d\t%d\t%d"%(i**1, i**2, i**3,  i**4))
   i+=1
print()
print()

i = 1
print("%-4s%-5s%-6s"%('i', 'i**2', 'i**3'))
print()
print()

i = 1
while i<=5:
   print("%-4d%-5d%-6d"%(i, i**2, i**3))
   i+=1


# Built-in string methods and functions
s =  "hello"
print(s.capitalize())

s = "hello"
print(s.center(10, '*'))
```

```
msg = 'he'
str1 = "hellohello"
print(str1.count(msg, 0, len(str1)))

msg = "she is my best friend"
print(msg.endswith("end", 0, len(msg)))

str1 = "the world is beautiful"
print(str1.startswith("th", 0, len(str1)))

msg = "she is my best my friend"
print(msg.find("my", 0, len(msg)))
print(msg.find("mine", 0, len(msg)))

try:
    print(msg.index("mine", 0, len(msg)))
except:
    print("substring not found")

# rfind searches from end
msg = "is this your bag?"
print(msg.rfind("is", 0, len(msg)))

print(msg.rindex("is"))
try:
    print(msg.rindex("z"))
except:
    print("substring not found")

msg = "jamesbond007"
print(msg.isalnum())

print(msg.isalpha())
msg = "jamesbond"
print(msg.isalpha())

msg = "007"
print(msg.isdigit())

msg = "Hello"
print(msg.islower())

msg = "   "
print(msg.isspace())

msg = "Hello"
print(msg.isupper())
print(len(msg))

s = "Hello"
print(s.ljust(10,'%'))

print(s.rjust(10,'*'))
print(s.rjust(10))

s = "-1234"
print(s.zfill(10))

s = "  Hello  "
print('abc' + s.lstrip() + 'zyx')

print('abc' + s.rstrip() + 'zyx')

print('abc' + s.strip() + 'zyx')

s = "Hello friends"
print(max(s))

s = "Hello Hello Hello"
print(s.replace("He", "Fo"))
print(s.replace("He", "Fo", 2))

s = "The world is beautiful"
print(s.title())

s = "hEllO WorLD"
print(s.swapcase())

s = "abc, def, ghi, jkl"
print(s.split(','))

# WAP to print the pattern
for i in range(1, 7):
    ch = 'A'
    print()
    for j in range(1, i+1):
        print(ch, end="")
        ch = chr(ord(ch)+1)

# WAP using while loop to iterate a given string
s = "Welcome to Python"
i = 0
while i < len(s):
    print(s[i], end="")
    i+=1

# WAP that encrypts a message by adding a key value to every character
s = input("Enter the string: ")
key = int(input("Enter the encryption key: "))
new_s = ""
for i in s:
    new_s += chr(ord(i)+key)
print(new_s)

# WAP that uses split function to split 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(s.split('\n'))

# WAP that accepts a string from user and re-displays the same string after removing vowels
vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
s = input("Enter the string: ")
for i in s:
    if i not in vowels:
        print(i, end="")

pattern = r"[a-zA-Z]+\s+\d+"
# Patterns that begin with one or more characters followed by space and followed by one or
more digits
matches = re.finditer(pattern, "LXI 2013,VXI 2015,VDI 20104,Maruti Suzuki Cars available
with us")
for match in matches:
    print(match.start(), match.end(), match.span())

# 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

pat = r"^\d+\s*"
pat = r"^[0-9]+ .*"
if re.match(pat, "123 adij"):
    print("Good")
```

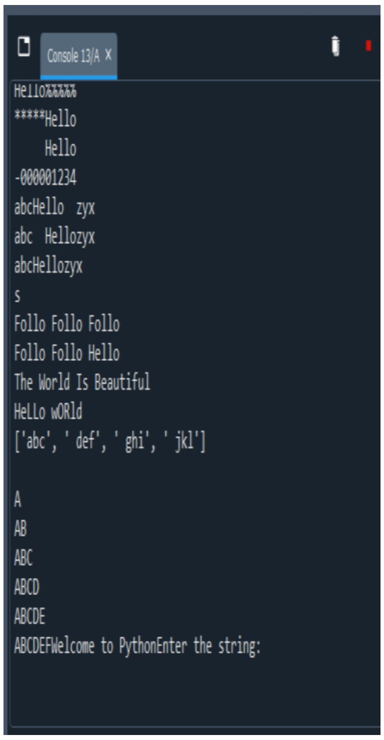
RESULTS:



```
In [1]: %runfile G:/spyder/codes/untitled2.py --wdir
orl
1    1    1    1
2    4    8    16
3    9    27        81
4    16        64        256
5    25        125       625


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


i   i**2 i**3


1   1    1
2   4    8
3   9    27
```



```
4   16   64
5   25   125
Hello
**hello***
2
True
True
7
-1
substring not found
5
5
substring not found
True
False
True
True
False
True
False
5
Hello%%%%%
```



```
Hello%%%%%
*****Hello
     Hello
-000001234
abcHello  zyx
abc  Hellozyx
abcHellozyx
s
Follo Follo Follo
Follo Follo Hello
The World Is Beautiful
HeLLo wORld
['abc', ' def', ' ghi', ' jkl']

A
AB
ABC
ABCD
ABCDE
ABCDEFWelcome to PythonEnter the string:
```

<u>EXPERIMENT 10</u>

<u>OBJECTIVE</u>: 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

<u>THEORY</u>:

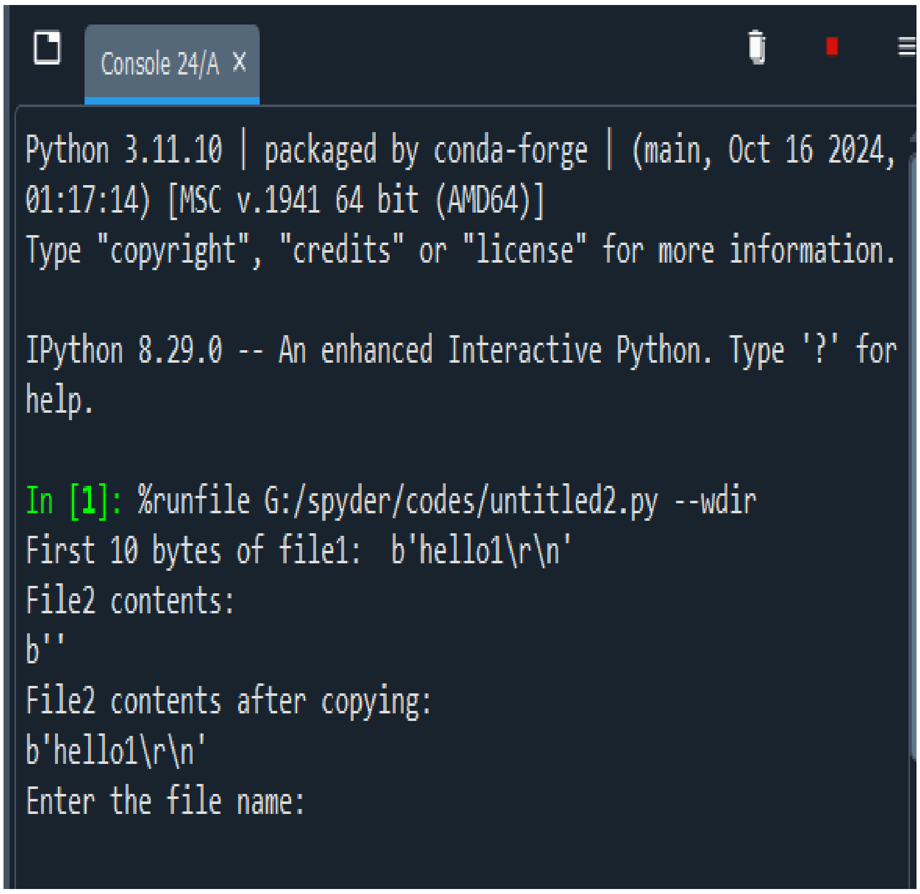
File handling in Python refers to the process of reading from, writing to, and performing other operations on files. Python provides built-in functions to handle files, such as open(), read(), write(), and others. Files can be opened in various modes like 'r' (read), 'w' (write), 'a' (append), 'b' (binary),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

<u>CODE</u>:

```
# WAP that copies first 10 bytes of a binary file into another
with open("file_handling_test/file1.txt", "rb") as f:
    a = f.read(10)
    print("First 10 bytes of file1: ", a)

with open("file2.txt", "wb+") as f2:
    print("File2 contents:")
    print(f2.read())
    f2.seek(0)
    t = f2.write(a)
    f2.seek(0)
    print("File2 contents after copying:")
    print(f2.read())

# 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

f = input("Enter the file name: ")
ch = input("Enter the character to be searched: ")
count = 0
with open("file_handling_test/"+f, "r") as f1:
    for line in f1:
        for c in line:
            if c == ch:
                count+=1
print("Count of given character in file: ", count)

# WAP to create a new directroy in the current directory
```

```
os.mkdir("new_dir")

# WAP that changes curr dur to newly created dir new_dir
os.chdir("new_dir")

# WAP to delete new_dir
os.rmdir("new_dir")
```

RESULTS:



```
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
First 10 bytes of file1:  b'hello1\r\n'
File2 contents:
b''
File2 contents after copying:
b'hello1\r\n'
Enter the file name:
```

EXPERIMENT 11

OBJECTIVE: 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 designation. Also define a function in location that prints the reflection of designation 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 are fundamental building blocks in object-oriented programming (OOP). They serve as blueprints for creating objects, defining the properties (attributes) and behaviors (methods) that the objects will have.

Objects are instances of a class. When a class is instantiated, an object is created with specific values for its attributes. For instance, an object of the Car class could be a red Toyota that knows how to start and drive. Objects encapsulate data and behaviors, providing a way to manage complexity in software development.

Inheritance allows a class (called a child or subclass) to derive properties and behaviors from another class (called a parent or superclass). This promotes code reuse and establishes a hierarchical relationship between classes. For instance, if Car is a parent class, a subclass like ElectricCar can inherit its attributes and methods while introducing specific features such as a battery_capacity attribute. This principle also supports polymorphism, where a subclass can override or extend the behavior of its parent class.

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:
    # Class variable to keep track of the number of employees
    total_employees = 0

    def __init__(self, name, designation, salary):
        # Instance variables for employee-specific data
        self.name = name
        self.designation = designation
        self.salary = salary
        # Increment the employee count
        Employee.total_employees += 1

    def display_details(self):
        # Display details of an employee
        print(f"Name: {self.name}, Designation: {self.designation}, Salary: {self.salary}")

e1 = Employee("Alice", "Manager", 75000)
e2 = Employee("Bob", "Developer", 50000)

e1.display_details()
e2.display_details()
print(f"Total Employees: {Employee.total_employees}")

# 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

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

    def area(self):
        # Calculate area of the circle
        return Circle.pi * (self.radius ** 2)

    def circumference(self):
        # Calculate circumference of the circle
        return 2 * Circle.pi * self.radius

c = Circle(5)
print(f"Area: {c.area()}")
print(f"Circumference: {c.circumference()}")
```

# 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.

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

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

    def reflect_on_x_axis(self):
        # Reflect designation along the x-axis
        return f"Reflection  on x-axis is ({self.x}, {-self.y})"

loc = Location(3, 4)
print(loc.reflect_on_x_axis())
```

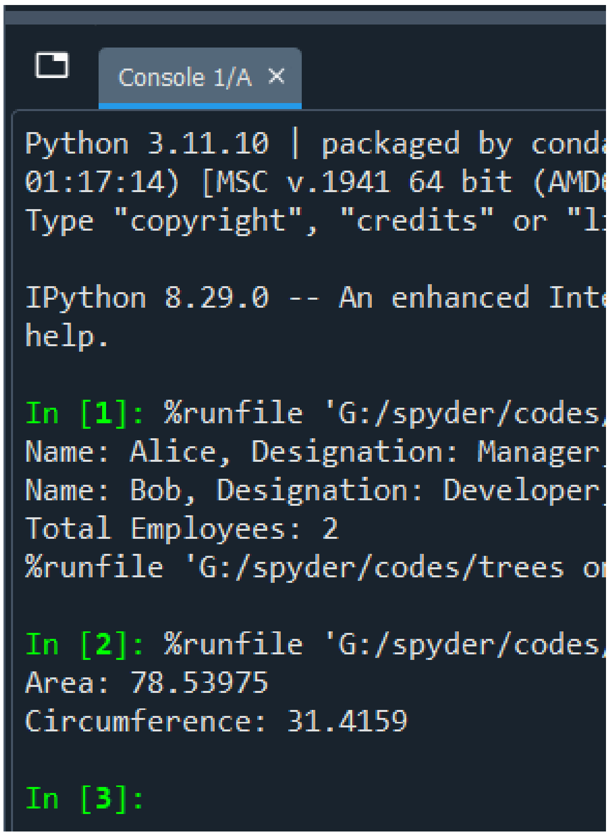
# WAP that has classes such as Student, Course, Department. Enroll a student in a course of a particular department. Classes are -

1. Student details - name, roll no
2. Course - name, code, year and semester
3. Department – Name

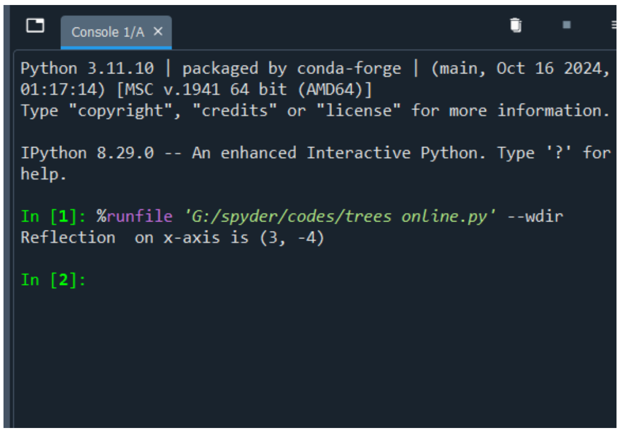
```
class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no

    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: {self.name}, Code: {self.code}, Year: {self.year}, Semester: {self.semester}")

class Department:
    def __init__(self, name):
        self.name = name

    def display(self):
        print(f"Department: {self.name}")

# Enroll student example
student = Student("John Doe", 101)
course = Course("Computer Science", "CS101", 2024, "Fall")
```

```
department = Department("Engineering")

student.display()
course.display()
department.display()
```

RESULTS:



```
Console 1/A
Python 3.11.10 | packaged by conda
01:17:14) [MSC v.1941 64 bit (AMD
Type "copyright", "credits" or "l

IPython 8.29.0 -- An enhanced Int
help.

In [1]: %runfile 'G:/spyder/codes
Name: Alice, Designation: Manager
Name: Bob, Designation: Developer
Total Employees: 2
%runfile 'G:/spyder/codes/trees o

In [2]: %runfile 'G:/spyder/codes
Area: 78.53975
Circumference: 31.4159

In [3]:
```



```
Console 1/A
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/trees online.py' --wdir
Reflection  on x-axis is (3, -4)

In [2]:
```

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 tp 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 means "many forms." In Python, it allows methods or functions to process objects of different types uniformly.

Errors and exceptions are runtime issues that can disrupt program execution. Python provides a mechanism to handle these gracefully using **try-except blocks**.

CODE:

```
# Demonstrate operator overloading
class ComplexNumber:
  def __init__(self, real, imag):
    self.real = real
    self.imag = imag

  # Overloading the '+' operator
  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)
print(f"Sum of complex numbers: {c1 + c2}")



#Demonstrate Method Overriding
class Animal:
    def speak(self):
        return "Animal speaks"

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

# Example usage
animal = Animal()
dog = Dog()
print(animal.speak())
print(dog.speak())

#3) WAP to handle the divide by zero exception
try:
    num = int(input("Enter numerator: "))
    denom = int(input("Enter denominator: "))
    result = num / denom
    print(f"Result: {result}")
except ZeroDivisionError:
    print("Error: Division by zero is not allowed!")

#4) Demonstrate Raise Exceptions, Instantiating Exceptions, assertion
```

```
# Raise and instantiate an exception
try:
    raise ValueError("This is a manually raised exception.")
except ValueError as e:
    print(e)

# Assertion example
x = 5
assert x > 0, "x must be positive"
print("Assertion passed, x is positive.")

# WAP that prompts the use to enter a number and prints the square of that number.
try:
    number = int(input("Enter a number: "))
    print(f"The square of {number} is {number ** 2}")
except ValueError:
    print("Error: Please enter a valid number.")
except KeyboardInterrupt:
    print("\nError: KeyboardInterrupt detected. Exiting.")
```

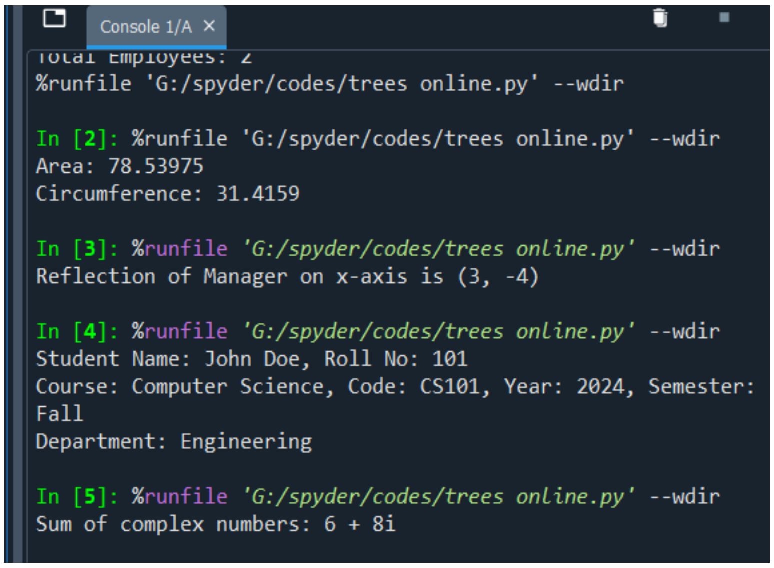
#6) WAP which infinitely prints natural numbers. Raise the stop Iteration Exception after displaying first 20 numbers tp exit from the program

```
class NaturalNumbers:
    def __init__(self, limit):
        self.current = 1
        self.limit = limit

    def __iter__(self):
        return self
    def __next__(self):
        if self.current > self.limit:
            raise StopIteration
        self.current += 1
        return self.current - 1

# Example usage
try:
    for num in NaturalNumbers(20):
        print(num)
except StopIteration:
    print("Stopped after 20 numbers.")
```

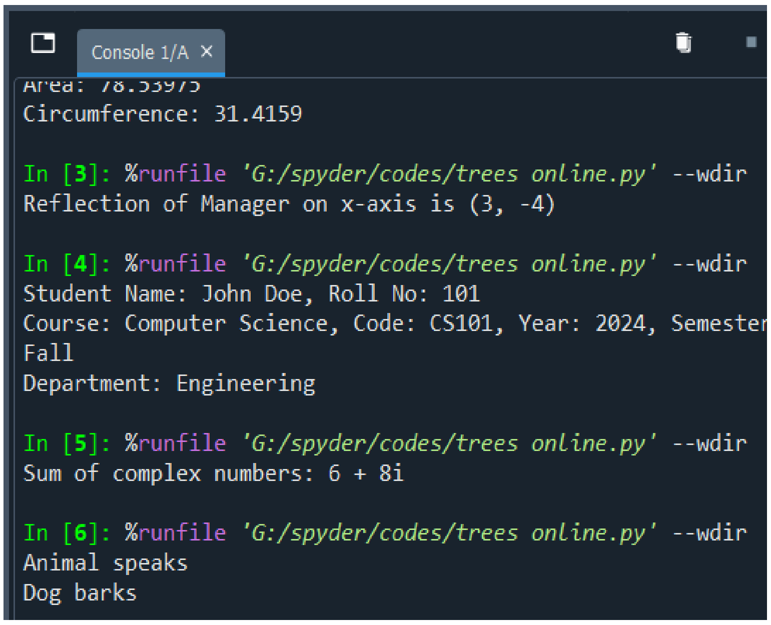
7) WAP that randomly generates a number. Raise a User Defined exception if the number is below 0.1

```
import random

class SmallNumberError(Exception):
    pass

try:
    number = random.random()
    print(f"Generated number: {number}")
    if number < 0.1:
        raise SmallNumberError("Number is below 0.1!")
except SmallNumberError as e:
    print(e)
```

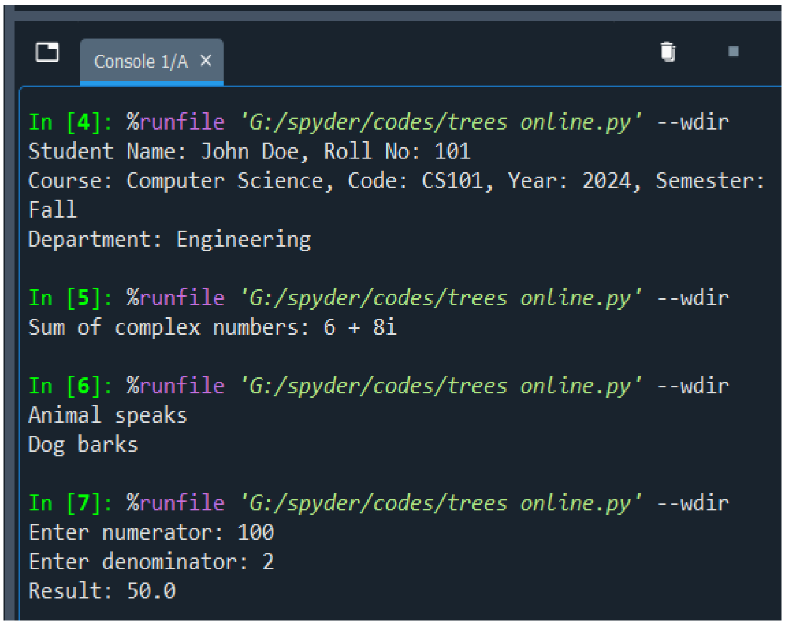
RESULTS:



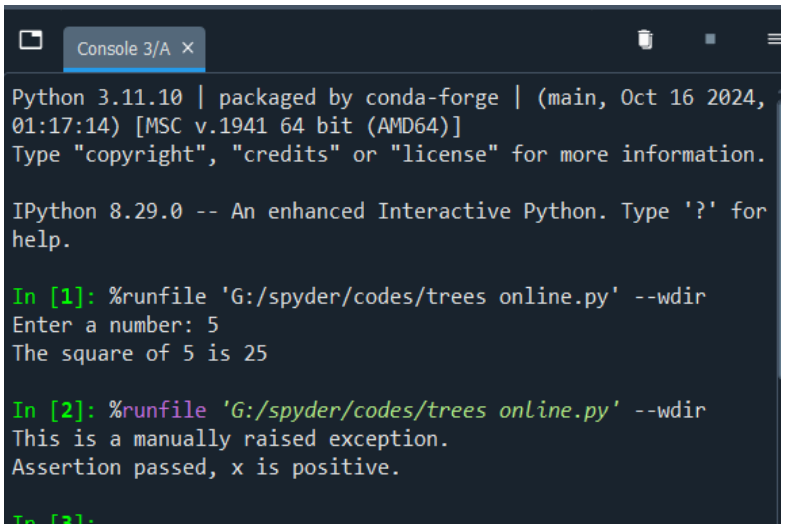
```
Console 1/A
Total Employees: 2
%runfile 'G:/spyder/codes/trees online.py' --wdir

In [2]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Area: 78.53975
Circumference: 31.4159

In [3]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Reflection of Manager on x-axis is (3, -4)

In [4]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Student Name: John Doe, Roll No: 101
Course: Computer Science, Code: CS101, Year: 2024, Semester:
Fall
Department: Engineering

In [5]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Sum of complex numbers: 6 + 8i
```



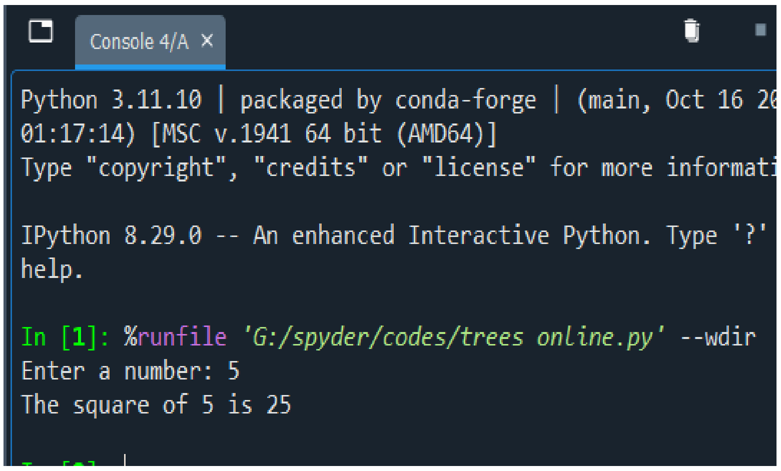
```
Console 1/A
Area: 78.53975
Circumference: 31.4159

In [3]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Reflection of Manager on x-axis is (3, -4)

In [4]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Student Name: John Doe, Roll No: 101
Course: Computer Science, Code: CS101, Year: 2024, Semester
Fall
Department: Engineering

In [5]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Sum of complex numbers: 6 + 8i

In [6]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Animal speaks
Dog barks
```



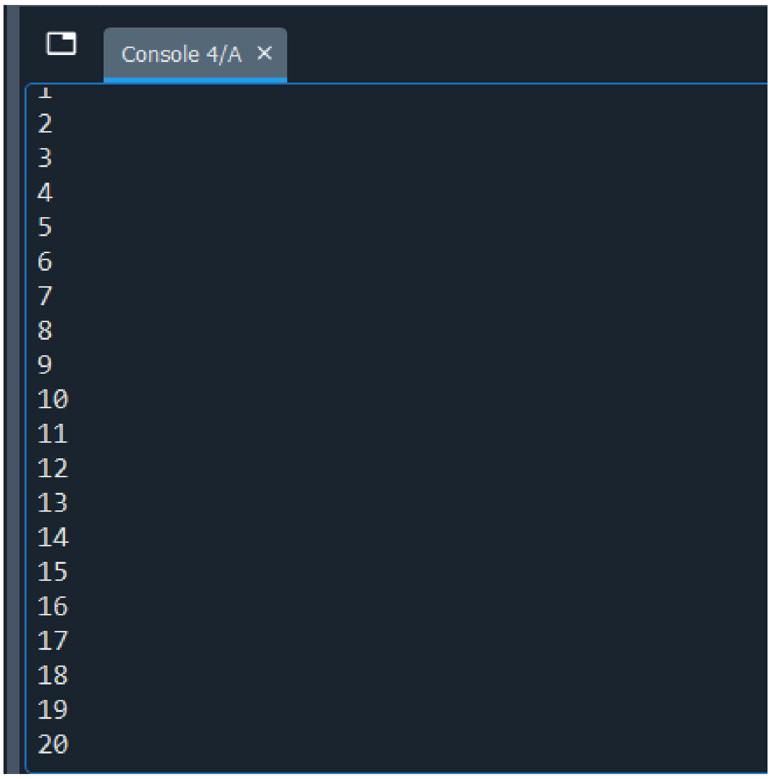
```
In [4]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Student Name: John Doe, Roll No: 101
Course: Computer Science, Code: CS101, Year: 2024, Semester:
Fall
Department: Engineering

In [5]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Sum of complex numbers: 6 + 8i

In [6]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Animal speaks
Dog barks

In [7]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Enter numerator: 100
Enter denominator: 2
Result: 50.0
```



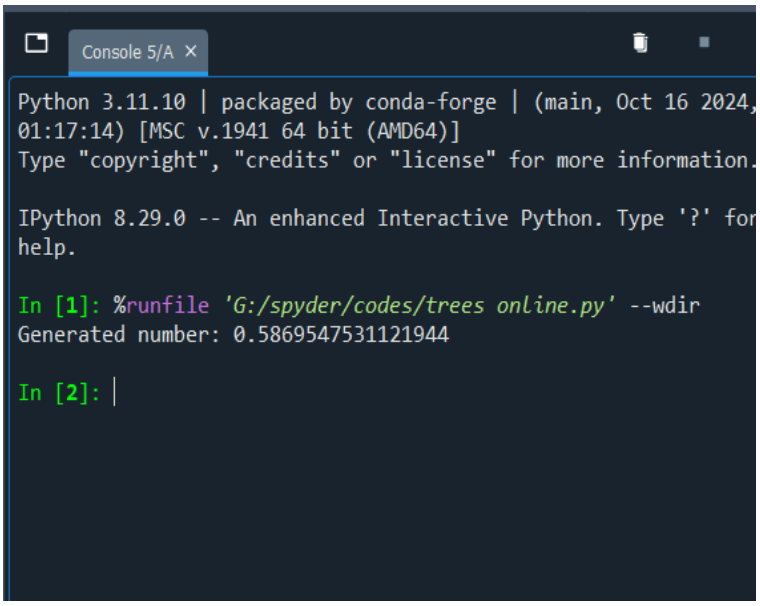
```
Python 3.11.10 | packaged by conda-forge | (main, Oct 16 2024,
01:17:14) [MSC v.1941 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

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

In [1]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Enter a number: 5
The square of 5 is 25

In [2]: %runfile 'G:/spyder/codes/trees online.py' --wdir
This is a manually raised exception.
Assertion passed, x is positive.
```



```
Python 3.11.10 | packaged by conda-forge | (main, Oct 16 20
01:17:14) [MSC v.1941 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more informati

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

In [1]: %runfile 'G:/spyder/codes/trees online.py' --wdir
Enter a number: 5
The square of 5 is 25
```



```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```



```
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/trees online.py' --wdir
Generated number: 0.5869547531121944

In [2]:
```