PRACTICAL NO : 01

AIM : Program based on Data Types

Numerical Data Type

# Integer examples
a = 10
b = -5
print(type(a))
print(type(b))

```
# Integer examples
a = 10
b = -5

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

```
[1]
<class 'int'>
<class 'int'>
```

# Float examples
x = 10.5
y = -3.14
print(type(x))
print(type(y))

```
# Float examples
x = 10.5
y = -3.14

print(type(x))
print(type(y))
```

```
[4] 0.0s
<class 'float'>
<class 'float'>
```

# Complex number
z1 = 3 + 5j
z2 = -2 - 4j
print(type(z1))
print(type(z2))
print(z1.real)
print(z1.imag)

```
# Complex number
z1 = 3 + 5j
z2 = -2 - 4j

print(type(z1))
print(type(z2))
print(z1.real)
print(z1.imag)
```

```
[6] 0.0s
<class 'complex'>
<class 'complex'>
3.0
5.0
```

```
# Addition
result = 7 + 4.5
print(result)

# Subtraction
result = 10 - 6
print(result)

# Multiplication
result = 8 * 2
print(result)

# Division
result = 32 / 2
print(result)

# Integer Division
result = 8 // 4
print(result)

# Modulus
result = 6 % 2
print(result)

# Exponentiation
result = 2 ** 3
print(result)
```

```
# Integer Division
result = 8 // 4
print(result)

# Modulus
result = 6 % 2
print(result)

# Exponentiation
result = 2 ** 3
print(result)
```

✓ 0.0s

```
11.5
4
16
16.0
2
0
8
```

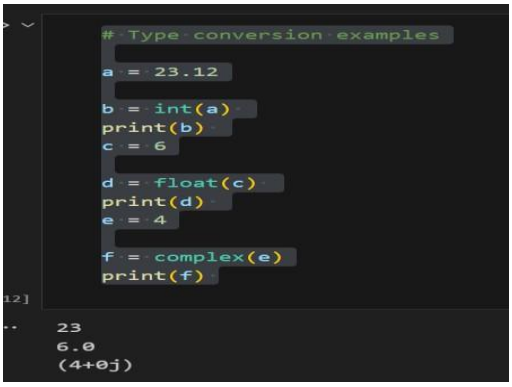
Type Conversion

Python allows conversion between numeric types using built-in functions:

- int(): Converts a float or string to an integer.
- float(): Converts an integer or string to a float.
- complex(): Converts numbers to complex numbers.

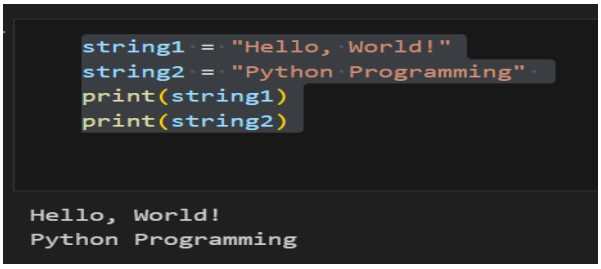
Type conversion examples

a = 23.12
b = int(a)
print(b)
c = 6
d = float(c)
print(d)
e = 4
f = complex(e)
print(f)



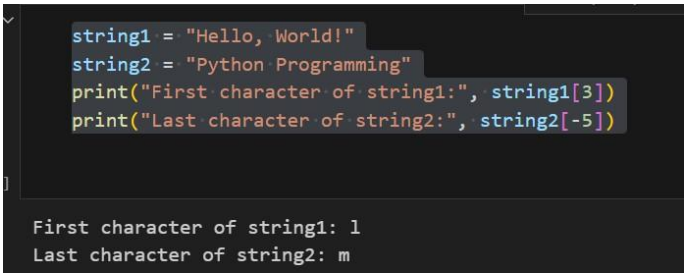
- **String Data Type**

string1 = "Hello, World!"
string2 = "Python Programming"
print(string1)
print(string2)



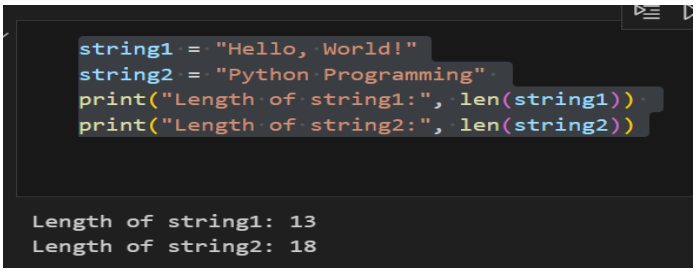
Accessing characters in a string

```
string1 = "Hello, World!"
string2 = "Python Programming"
print("First character of string1:", string1[3])
print("Last character of string2:", string2[-5])
```



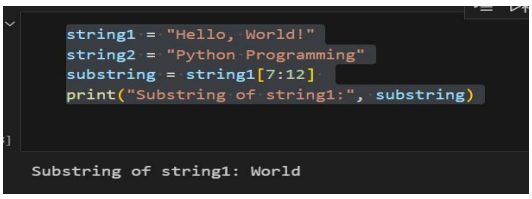
String length

```
string1 = "Hello, World!"
string2 = "Python Programming"
print("Length of string1:", len(string1))
print("Length of string2:", len(string2))
```



String slicing

```
string1 = "Hello, World!"
string2 = "Python Programming"
substring = string1[7:12]
print("Substring of string1:", substring)
```



String concatenation

```
string1 = "Hello, World!"
string2 = "Python Programming"
combined_string = string1 + " " + string2
print("Combined string:", combined_string)
```

```
string1 = "Hello, World!"
string2 = "Python Programming"
combined_string = string1 + " " + string2
print("Combined string:", combined_string)
```

```
Combined string: Hello, World! Python Programming
```

- **String Methods: upper(), lower(), count(), replace(), and split().**

string1 = "Hello, World!"
string2 = "Python Programming"
print("Uppercase string1:", string1.upper())
print("Lowercase string2:", string2.lower())
print("Count of 'o' in string1:", string1.count('o'))

```
string1 = "Hello, World!"
string2 = "Python Programming"
print("Uppercase string1:", string1.upper())
print("Lowercase string2:", string2.lower())
print("Count of 'o' in string1:", string1.count('o'))
```

```
Uppercase string1: HELLO, WORLD!
Lowercase string2: python programming
Count of 'o' in string1: 2
```

Replacing parts of a string

string1 = "Hello, World!"
string2 = "Python Programming"
new_string = string1.replace("World", "Python")
print("After replacement:", new_string)

```
string1 = "Hello, World!"
string2 = "Python Programming"
new_string = string1.replace("World", "Python")
print("After replacement:", new_string)
```

```
After replacement: Hello, Python!
```

Splitting a string

string1 = "Hello, World!"
string2 = "Python Programming"
words = string2.split()
print("Words in string2:", words)

```
string1 = "Hello, World!"
string2 = "Python Programming"
words = string2.split()
print("Words in string2:", words)
```

```
Words in string2: ['Python', 'Programming']
```

Checking if a substring exists:

```
string1 = "Hello, World!"
string2 = "Python Programming"
if "Python" in string2:
    print("Substring 'Python' found in string2!")
```

```
string1 = "Hello, World!"
string2 = "Python Programming"
if "Python" in string2:
    print("Substring 'Python' found in string2!")
```

```
Substring 'Python' found in string2!
```

Formatting strings

```
name = "Lilly"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print("Formatted string:", formatted_string)
```

```
Formatted string: My name is Lilly and I am 30 years old.
```

Iterating through a string:

```
string1 = "Hello, World!"
string2 = "Python Programming"
print("Characters in string1:")
for char in string1:
    print(char, end=' ')
    print()
```

```
string1 = "Hello, World!"
string2 = "Python Programming"
print("Characters in string1:")
for char in string1:
    print(char, end=' ')
    print()
```

```
Characters in string1:
H
e
l
l
o
,

W
o
r
l
d
!
```

Reversing a string

reversed_string = string1[::-5]
print("Reversed string1:", reversed_string)

```
reversed_string = string1[::-5]
print("Reversed string1:", reversed_string)
```

```
Reversed string1: !Wl
```

List Data type

List of Integers:

integers_list = [10,11,12,13,14,15]
print(integers_list)

A. List of Strings:

string_list = ["Lichi", "Grapes", "cherry"]
print(string_list)

List of Mixed Data Types:

mixed_list = [1, "Banana", 3.14, True]
print(mixed_list)

```
integers_list = [10,11,12,13,14,15]
print(integers_list)
```

```
[10, 11, 12, 13, 14, 15]
```

```
string_list = ["Lichi", "Grapes", "cherry"]
print(string_list)
```

```
['Lichi', 'Grapes', 'cherry']
```

```
mixed_list = [1, "Banana", 3.14, True]
print(mixed_list)
```

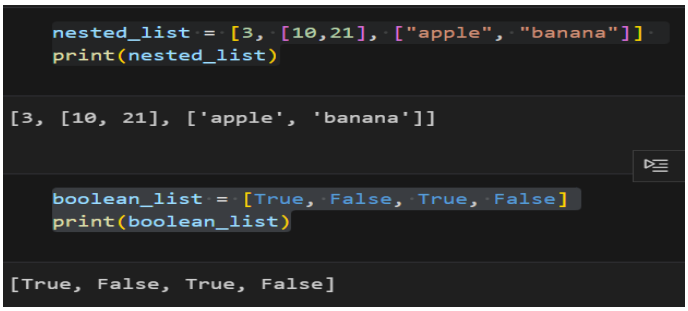
```
[1, 'Banana', 3.14, True]
```

Nested List (List inside a list):

nested_list = [3, [10,21], ["apple", "banana"]]
print(nested_list)

List of Boolean Values:

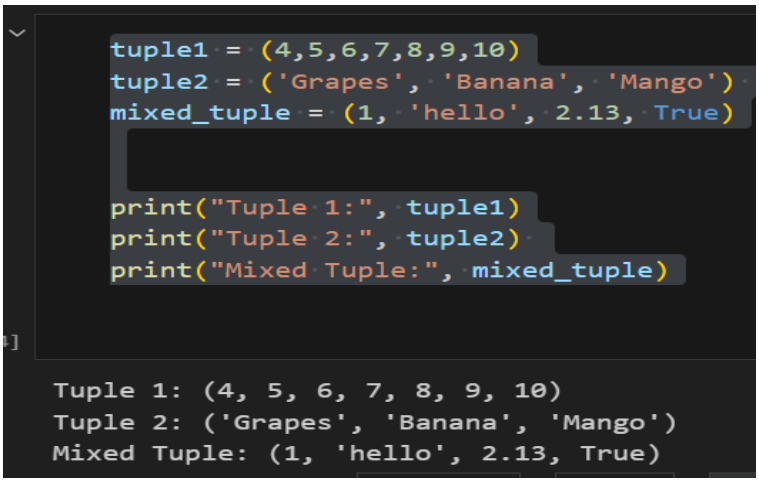
boolean_list = [True, False, True, False]
print(boolean_list)



Tuple Data Type

tuple1 = (4,5,6,7,8,9,10)
tuple2 = ('Grapes', 'Banana', 'Mango')
mixed_tuple = (1, 'hello', 2.13, True)

print("Tuple 1:", tuple1)
print("Tuple 2:", tuple2)
print("Mixed Tuple:", mixed_tuple)



Accessing elements in a tuple

tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("First element of tuple1:", tuple1[0])
print("Last element of tuple2:", tuple2[-1])

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("First element of tuple1:", tuple1[0])
print("Last element of tuple2:", tuple2[-1])
```

```
First element of tuple1: 1
Last element of tuple2: cherry
```

Tuple length

tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("Length of tuple1:", len(tuple1))
print("Length of tuple2:", len(tuple2))

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("Length of tuple1:", len(tuple1))
print("Length of tuple2:", len(tuple2))
```

```
Length of tuple1: 5
Length of tuple2: 3
```

Slicing a tuple:

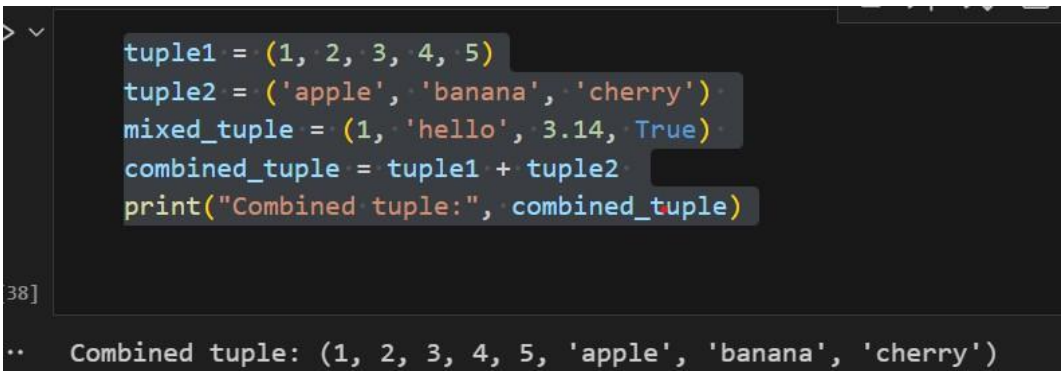
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
sub_tuple = tuple1[1:4]
print("Sliced tuple from tuple1:", sub_tuple)

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
sub_tuple = tuple1[1:4]
print("Sliced tuple from tuple1:", sub_tuple)
```

```
Sliced tuple from tuple1: (2, 3, 4)
```

Concatenating tuples

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
combined_tuple = tuple1 + tuple2
print("Combined tuple:", combined_tuple)
```



Repeating tuples

```
tuple1 = (3,4,5,6,7,8,9)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
repeated_tuple = tuple2 * 2
print("Repeated tuple2:", repeated_tuple)
```

Iterating through a tuple

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("Elements in mixed_tuple:")
for item in mixed_tuple:
    print(item, end=' ')
    print()
```

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("Elements in mixed_tuple:")
for item in mixed_tuple:
    print(item, end=' ')
    print()
```

```
Elements in mixed_tuple:
1
hello
3.14
True
```

Tuple unpacking

tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
a, b, c, d = mixed_tuple
print("Unpacked values:", a, b, c, d)

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
a, b, c, d = mixed_tuple
print("Unpacked values:", a, b, c, d)
```

```
Unpacked values: 1 hello 3.14 True
```

Nested tuples

tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('Grapes', 'Banana', 'Mango')
mixed_tuple = (1, 'hello', 3.14, True)
nested_tuple = (tuple1, tuple2)
print("Nested Tuple:", nested_tuple)

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('Grapes', 'Banana', 'Mango')
mixed_tuple = (1, 'hello', 3.14, True)
nested_tuple = (tuple1, tuple2)
print("Nested Tuple:", nested_tuple)
```

```
Nested Tuple: ((1, 2, 3, 4, 5), ('Grapes', 'Banana', 'Mango'))
```

Converting a tuple to a list

tuple1 = (4,5,7,8,9)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
tuple_to_list = list(tuple1)
print("Converted tuple1 to list:", tuple_to_list)

```
tuple1 = (4,5,7,8,9)
tuple2 = ('apple', 'banana', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
tuple_to_list = list(tuple1)
print("Converted tuple1 to list:", tuple_to_list)
```

```
Converted tuple1 to list: [4, 5, 7, 8, 9]
```

Creating a tuple with a single element

tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('Grapes', 'Banana', 'Mango')
mixed_tuple = (1, 'hello', 3.14, True)
single_element_tuple = (50,)
print("Single element tuple:", single_element_tuple)

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('Grapes', 'Banana', 'Mango')
mixed_tuple = (1, 'hello', 3.14, True)
single_element_tuple = (50,)
print("Single element tuple:", single_element_tuple)
```

```
[46]
Single element tuple: (50,)
```

Counting occurrences

tuple1 = (4,5,6,7,8)
tuple2 = ('mango', 'apple', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("Count of 2 in tuple1:", tuple1.count(2))

```
tuple1 = (4,5,6,7,8)
tuple2 = ('mango', 'apple', 'cherry')
mixed_tuple = (1, 'hello', 3.14, True)
print("Count of 2 in tuple1:", tuple1.count(2))
```

```
[47]
Count of 2 in tuple1: 0
```

Finding index of an element

tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'mango', 'banana')
mixed_tuple = (1, 'hello', 6.12, True)
print("Index of 3 in tuple1:", tuple1.index(5))

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ('apple', 'mango', 'banana')
mixed_tuple = (1, 'hello', 6.12, True)
print("Index of 3 in tuple1:", tuple1.index(5))
```

[48]

```
Index of 3 in tuple1: 4
```

# Creating a dictionary
person = {
"name": "Lilly", "age": 22,
"city": "England"
}
person2 = dict(name="Jack", age=30, city="France")

# Accessing the value by key
print(person["name"])

# Using the .get() method
print(person.get("age"))

```
# Creating a dictionary
person = {
"name": "Lilly", "age": 22,
"city": "England"
}
person2 = dict(name="Jack", age=30, city="France")

# Accessing the value by key
print(person["name"])

# Using the .get() method
print(person.get("age"))
```

```
Lilly
22
```

```
# Modifying an existing key
person["age"] = 23
# Adding a new key-value pair
person["job"] = "data analyst"
print(person)
```


```
del person["city"]
job = person.pop("job")
print(job)
print(person)
```


```
# Creating a set
fruits = {"apple", "banana", "mango"}

numbers = set([10,11,12,13,14,15])

# Creating an empty set
empty_set = set()
# Duplicate elements are removed
fruits = {"apple", "banana", "mango", "apple"}
print(fruits)
```

```
# Creating a set
fruits = {"apple", "banana", "mango"}

numbers = set([10,11,12,13,14,15])

# Creating an empty set
empty_set = set()


# Duplicate elements are removed
fruits = {"apple", "banana", "mango", "apple"}
print(fruits)
```

```
{'apple', 'banana', 'mango'}
```

# Adding a single element
fruits.add("lichi")
print(fruits)

# Adding multiple elements
fruits.update(["mango", "grapes"])
print(fruits)

```
# Adding a single element
fruits.add("lichi")
print(fruits)

# Adding multiple elements
fruits.update(["mango", "grapes"])
print(fruits)
```

```
{'orange', 'mango', 'apple', 'banana', 'lichi', 'grape', 'grapes'}
{'orange', 'mango', 'apple', 'banana', 'lichi', 'grape', 'grapes'}
```

fruits.remove("banana")
fruits.discard("cherry")
random_fruit = fruits.pop()
print(random_fruit)

```
fruits.remove("banana")

fruits.discard("cherry")
random_fruit = fruits.pop()

print(random_fruit)
```

```
orange
```

A = {4,5,6}
B = {10,11,12}
union_set = A.union(B)
print(union_set)

```
A = {4,5,6}

B = {10,11,12}

union_set = A.union(B)
print(union_set)
```

```
[66]
{4, 5, 6, 10, 11, 12}
```

intersection_set = A.intersection(B)
print(intersection_set)
difference_set = A.difference(B)
print(difference_set)
sym_diff_set = A.symmetric_difference(B)
print(sym_diff_set)

```
intersection_set = A.intersection(B)
print(intersection_set)
```

```
set()
```

```
difference_set = A.difference(B)
print(difference_set)
```

```
{4, 5, 6}
```

Generate + Code + Mark
Add Code Cell

```
sym_diff_set = A.symmetric_difference(B)
print(sym_diff_set)
```

```
{4, 5, 6, 10, 11, 12}
```

```
# Define sets
A = {1, 2, 3}
B = {3, 4, 5}

# Union
print("A | B =", A | B)

# Intersection
print("A & B =", A & B)

# Difference
print("A - B =", A - B)

# Symmetric Difference
print("A ^ B =", A ^ B)
```

```
A | B = {1, 2, 3, 4, 5}
A & B = {3}
A - B = {1, 2}
A ^ B = {1, 2, 4, 5}
```

PRACTICAL NO :02

AIM : Program based on Functions

To define a function in Python, you use the def keyword followed by the function name and parentheses.

Defining and Calling Functions

```
#Define a function
def my_function():
  print("Hello from a function")
my_function()
```

```
Python313/python.exe
Hello from a function
```

Parameters and Arguments

```
def greet(name):
  print(f"Hello, {name}")

greet("Jack")
greet("Sam")
```

```
Python/Python313/python.exe
Hello, Jack
Hello, Sam
PS C:\Users\swati> & C:/Use
```

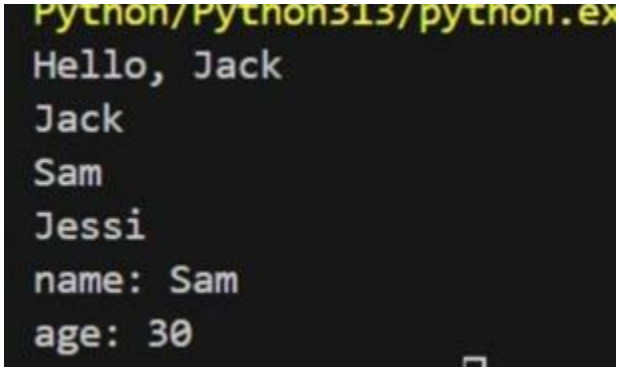
Return Values

```
def add(a, b):
  return a + b

result = add(5,10)
print(result)
```

```
Python/Python313/p
15
```

Default Parameter Values and Arbitrary Arguments

```
# Function with a default parameter value
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}") # Arbitrary arguments

def print_names(*names):
    for name in names:
        print(name)

# Arbitrary keyword arguments
def print_key_values(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

# Using the functions
greet("Jack") # Uses default greeting
print_names("Jack", "Sam", "Jessi")
print_key_values(name="Sam", age=30)
```



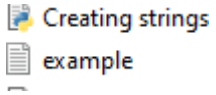
PRACTICAL NO : 03

AIM: Program based on File Handling

Creating a File

```
import os
def create_file(filename): try:
        with open(filename,'w') as f:
            f.write("Hello World!\n")
        print("File" + filename +" created successfully.") except
    error:
        print("Error:could not create file" + filename)

if __name___== "__main___":
    filename="example.txt"
    create_file(filename)
```

```
= RESTART: C:/Users/ADMIN/AppData/Local/Prog
py
Fileexample.txt created successfully.
>
```



- **Read a file**

```
import os
def read_file(filename): try:
        with open(filename, 'r') as f: contents
            = f.read() print(contents)
    except IOError:
        print("Error: could not read file " + filename)
if __name___== "__main___":
filename="example.txt"
    read_file(filename)
```

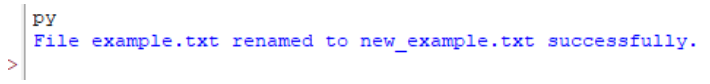
```
= RESTART: C:/Users/
py
Hello World!
```

Append a file

```
import os
def append_file(filename,text): try:
        with open(filename,'a') as f:
            f.write(text)
        print("Text appended to file " + filename + "Successfully.")

    except IOError:
       print("Error: could not append to file " + filename)

if __name___== "___main___":
    filename="example.txt"
    append_file(filename,"I am Lilly.")
```

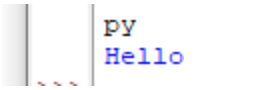
- **Rename a file**

```
import os
def rename_file(filename, new_filename): try:
        os.rename(filename, new_filename)
        print("File " + filename + " renamed to " + new_filename + " successfully.") except
    IOError:
        print("Error: could not rename file " + filename) if
____name___== "___main___":
    filename="example.txt" new_filename =
    "new_example.txt"
    rename_file(filename,new_filename)
```



- **Read Only Parts Of File**

```
f=open("new_example.txt","r")
print(f.read(5))
```



- **Read Lines**

```
f=open("new_example.txt","r")
```

print(f.readline())

```
= RESTART: C:/Users
py
Hello World!
```

- **Writing a File**

```
import os
def write_file(filename, content): try:
        with open(filename, 'w') as f:
            f.write(content)
        print(f"Successfully wrote to file: {filename}") except
    IOError:
        print("Error: could not write to file " + filename)

if __name___== "__main___":
    filename = "example.txt"
    content = "Hello, World!\nThis is a line written to the file."
    write_file(filename, content)
```

```
py
Successfully wrote to file: example.txt
```

```
File  Edit  Format  View  Help
Hello, World!
This is a line written to the file.
```

- **Closing a File**

```
f=open("example.txt","r")
print(f.readline())  f.close()
```

```
= RESTART: C:/Users/ADM
py
Hello, World!
>>>
```

- **Delete a file:**

```
import os
def delete_file(new_filename): try:
        os.remove(new_filename)
        print("File " + new_filename + " deleted successfully.") except
    IOError:
        print("Error: could not delete file " + new_filename)

        if  __name__==  "__main__":
    new_filename = "new_example.txt"
    delete_file(new_filename)
```

```
= RESTART: C:/Users/ADMIN/AppData/Local/Program
py
File new_example.txt deleted successfully.
>>
```

PRACTICAL NO : 04

AIM : **Program based on Packages**

Creating and Using a Simple Package Step

1:Directory Structure

```
my_package/
    __init__.py
    math_operations.py
    string_operations.py
```

Step 2: Package Files

math_operations.py (Module inside the package):

```
# math_operations.py

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

Step 3:string_operations.py:

```
def to_uppercase(s): return
    s.upper()

def to_lowercase(s): return
    s.lower()
```

Step 4:___init___.py (Package initializer):

```
from .math_operations import add, subtract
from .string_operations import to_uppercase, to_lowercase
```

Step 5:Using the Package

Now, you can use the my_package package in another script:

```
# main.py

import my_package

# Math operations
```

```
print(my_package.subtract(5, 3))

# String operations
print(my_package.to_uppercase('hello'))
print(my_package.to_lowercase('WORLD'))
```

Output:

```
8
2
HELLO
world
```

Using External Packages:

Step 1: Install numpy:We can install numpy packages by running “pip install numpy” on command prompt.

Step 2: Program using numpy

```
# numpy_example.py

import numpy as np

# Create an array
arr = np.array([1, 2, 3, 4, 5])

# Perform operations
print("Original Array:", arr)
print("Array multiplied by 2:", arr * 2)
print("Sum of array:", np.sum(arr))
print("Mean of array:", np.mean(arr))
```

Output:

```
Python/Python313/python.exe c:/Users/swa
Original Array: [ 6  7  8  9 10]
Array multiplied by 2: [12 14 16 18 20]
Sum of array: 40
```

Building a Custom Package for Data Handling

We will now create a package that helps handle basic CSV operations.

Step 1: Directory Structure

```
csv_handler/
    __init__.py
    read_csv.py
    write_csv.py
```

Step 2: Package Files

1. read_csv.py (Module for reading CSV files):

```
# read_csv.py

import csv

def read_csv_file(Salary_Sheet):
    data = []
    with open(Salary_Sheet, mode='r') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            data.append(row)
    return data
```

2. write_csv.py (Module for writing to CSV files):

```
# write_csv.py

import csv

def write_csv_file(Salary_Sheet, data):
    with open(Salary_Sheet, mode='w', newline='') as file:
        csv_writer = csv.writer(file)
        csv_writer.writerows(data)
```

3. __init__py: Package Initializer

```
# __init__.py
from .math_operations import add, subtract
from .string_operations import to_uppercase, to_lowercase
```

Step 3:Using the package

```
import csv_handler

# Writing data to a CSV file
data_to_write = [['Name', 'Basic Pay'], ['Sam', 57000], ['Jack', 40000]]
csv_handler.write_csv_file('Salary_Sheet.csv', data_to_write)

# Reading data from a CSV file
data = csv_handler.read_csv_file('Salary_Sheet.csv')

print(data)
```

Output:

```
[['Name', 'Basic Pay'], ['Sam', 57000], ['Jack', 40000]]
```

PRACTICAL NO :05

AIM : **Program based on Controlled Structures**

If,else and elif:

if Statement: Checks if the number is greater than zero and prints that it is positive. elif Statement: Checks if the number is less than zero and prints that it is negative.

```
def check_number():
    number = float(input("Enter a number: "))
    if number > 0:
        print(f"{number} is a positive number.")
    elif number < 0:
        print(f"{number} is a negative number.")
    else:
        print("The number is zero.")
if __name___== "__main__":
```

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.

========== RESTART: C:/Users/nehar/AppData/Local/Programs/Python/Python313/pyfilesss/check_number.py ========
Enter a number: 2
2.0 is a positive number.

========== RESTART: C:/Users/nehar/AppData/Local/Programs/Python/Python313/pyfilesss/check_number.py ========
Enter a number: -9
-9.0 is a negative number.

========== RESTART: C:/Users/nehar/AppData/Local/Programs/Python/Python313/pyfilesss/check_number.py ========
Enter a number: 0
The number is zero.
```

```
    check_number()
```

For Loop:

It Iterates over a sequence (like a list, tuple, or string) or a range of numbers.

```
def print_squares():
    print("Squares of numbers from 1 to 10:")

    # Using a for loop to iterate over a range of numbers for
    number in range(1, 11):
        square = number ** 2 # Calculate the square of the number
        print(f"The square of {number} is {square}")

        if _____name_== "__main_":
        print_squares()
```

```
Squares of numbers from 1 to 10:
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```

While Loop:

A variable count is initialized to 1, which will be used to keep track of the current number.

```
def count_to_ten():
    count = 1

# While loop to count from 1 to 10
    while count <= 10:
        print(count)
        count += 1

if __name___== "__main__":
    count_to_ten()
```

```
====== RESTAI
1
2
3
4
5
6
7
8
9
10
```

Break Statement:

1. A list numbers contains some predefined integer values.
2. The for loop iterates through the indices of the list numbers.
3. Inside the loop, an if statement checks if the current number matches the target.

```
def find_number(target):
    numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
    # Using a for loop to iterate through the list
    for index in range(len(numbers)):
        if numbers[index] == target:
            print(f"Number {target} found at index {index}.")
            break  # Exit the loop if the number is found
   else:
        # This executes only if the loop completes without 'break'
        print(f"Number {target} not found in the list.")
if __name___== "__main__":
    target_number = int(input("Enter a number to find: "))
```

```
find_number(target_number)
===== RESTART: C:/Users/swat
Enter a number to find: 30
Number 30 found at index 2.
```

Continue Statement:

The for loop iterates over the numbers from 1 to 10 using range(1, 11).

```
def print_even_numbers():  print("Even
    numbers from 1 to 10:")

    # Using a for loop to iterate through numbers from 1 to 10 for number
    in range(1, 11):
        if number % 2 != 0:
            continue  # Skip odd numbers
        print(number)  # Print even numbers

if __name___== "__main__":
    print_even_numbers()
```

```
Even numbers from 1 to 10:
2
4
6
8
10
```

Pass Statement:

1. The for loop iterates through each number in the provided list.
2. An if statement checks if the current number is even using number % 2 == 0.
3. If the number is even, the pass statement is executed, which means no action is taken for even numbers.
4. If the number is odd, it is printed to the console.

```
def skip_even_numbers(numbers):
    for number in numbers:
        if number % 2 == 0:
            pass  # Do nothing for even numbers
        else:
            print(f"Odd number: {number}")
def main():
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print("Processing numbers:")
    skip_even_numbers(numbers)

if __name___== "__main__":
    main()
```

```
========== RESTART: C:/Users/nehar/AppData/Local/Programs/Python/Python313/pyfilesss/skip_even_numbers.py ==
Processing numbers:
Odd number: 1
Odd number: 3
Odd number: 5
Odd number: 7
Odd number: 9
```

PRACTICAL NO : 06

AIM : Program based on Exception Handling

In Python, exception handling is done using try, except, else, and finallyblocks to manage errors during program execution.

- tryblock: Contains the code that might raise an exception.
- exceptblock: Handles specific exceptions if they occur.
- elseblock (optional): Runs if no exceptions are raised in the tryblock.
- finallyblock (optional): Always runs, regardless of whether an exception occurred or not.
- If an error occurs, it's caught by the corresponding exceptblock, preventing the program from crashing.

Basic Exception Handling:

- try block contains the code that may potentially throw exceptions.
- except ValueError handles the case where the user input is not a valid integer.
- except ZeroDivisionError handles cases where division by zero is attempted.

```
try:
    # Prompt user for input and convert to integer
    number = int(input("Enter a number: "))
    # Attempt division
    result = 100 / number
    print("The result is:", result)
except ValueError:
    # Handle invalid input (non-numeric)
    print("Invalid input! Please enter a valid integer.")
except ZeroDivisionError:
    # Handle division by zero
    print("Error! Cannot divide by zero.")
```

```
========== RESTART
Enter a number: 8
The result is: 12.5

========== RESTART
Enter a number: K
Invalid input! Please enter a valid integer.
```

Catching Multiple Exceptions:

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("The result is:", result)
except ValueError:
    print("That's not a valid number.")
except ZeroDivisionError:
```

```
    print("Cannot divide by zero.")
```

```
========================================
Enter a number: 7
The result is: 1.4285714285714286

========================================
Enter a number: GHJ
That's not a valid number.

========================================
Enter a number: 0
Cannot divide by zero.
```

Using else block: The else block will run only if no exceptions were raised in the try block.

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Cannot divide by zero.")
else:
    print("Result is:", result)
```

```
313/python.exe c:/Users/swati/Downlo
ads/ifelse.py
Enter a number: 9
Result is: 1.1111111111111112
PS C:\Users\swati> & C:/Users/swati/
AppData/Local/Programs/Python/Python
313/python.exe c:/Users/swati/Downlo
ads/ifelse.py
Enter a number: 0
Cannot divide by zero.
PS C:\Users\swati> \
```

Using finally Block: The finally block will execute no matter what happens in the try and except blocks.

```
try:
    age = int(input("Enter your age: "))
    if age < 0:
        raise ValueError("Age cannot be negative.")
except ValueError as ve:
    print(ve)
```

```
Enter your age: -58
Age cannot be negative.
```

Raising Exceptions: You can use raise to manually trigger an exception.

```
try:
    file = open("example.txt", "r") content
    = file.read()
except FileNotFoundError:
    print("File not found.")
finally:
    print("Execution complete.")
```

```
File not found.
Execution complete.
```

Custom Exceptions: You can define your own exceptions by subclassing the built-in Exception class.

```
class NegativeAgeError(Exception):
  pass
def check_age(age):
  if age < 0:
    raise NegativeAgeError("Age cannot be negative.")
try:
  age = int(input("Enter your age: "))
  check_age(age)
except NegativeAgeError as e:
  print(e)
```

```
313/python.exe c:/Users/sw
ads/ifelse.py
Enter your age: -10
Age cannot be negative.
PS C:\Users\swati>
```

PRACTICAL NO : 07

AIM: Program based on Inheritance

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (child class) to inherit attributes and methods from another class (parent class). This promotes code reuse and allows for hierarchical classifications.

Example of Inheritance in Python

```
# Parent class
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return f"{self.name} makes a sound."

# Child class inheriting from Animal
class Dog(Animal):
    def speak(self):
        return f"{self.name} barks."

# Another child class inheriting from Animal
class Cat(Animal):
    def speak(self):
        return f"{self.name} meows."

# Creating objects
dog = Dog("Jannu")
cat = Cat("Anbu")

# Calling the methods
print(dog.speak())
print(cat.speak())
```

```
.py
Jannu barks.
Anbu meows.
```

Example of Using super() in Inheritance

The super() function is used to call methods from the parent class inside the child class. Here's an example where we extend the parent class method using super().

```
# Parent class
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def speak(self):
        return f"{self.name} makes a sound."

# Child class Dog with super()
class Dog(Animal):
    def __init__(self, name, species, breed):
        super().__init__(name, species)
        self.breed = breed

    def speak(self):
        return f"{self.name}, the {self.breed}, barks."

# Creating an object of Dog
dog = Dog("Jannu", "Dog", "Golden Retriever")

# Accessing attributes and methods
print(dog.name)
print(dog.species)
print(dog.breed)
print(dog.speak())
```

```
.py
Jannu
Dog
Golden Retriever
Jannu, the Golden Retriever, barks.
```

A) Types of Inheritance:

1. Single Inheritance – A child class inherits from one parent class.
2. Multiple Inheritance – A child class inherits from more than one parent class.
3. Multilevel Inheritance – A child class inherits from a parent class, which in turn inherits from another parent class.

```
# Parent class 1
class Animal:
```

```
    def speak(self):
        return "Animal sound"

# Parent class 2
class Vehicle:
    def move(self):
        return "Moves on the road"

# Child class inheriting from both Animal and Vehicle
class RobotDog(Animal, Vehicle):
    def speak(self):
        return "RobotDog barks electronically"

# Creating an object of RobotDog
robot_dog = RobotDog()

# Accessing methods
print(robot_dog.speak())
print(robot_dog.move())
```

```
.py
RobotDog barks electronically
Moves on the road
```

PRACTICAL NO : 08

AIM: Program based on Overloading

In Python, method overloading (as seen in other languages like Java) is not directly supported because Python does not support function signature-based overloading. However, we can achieve similar behavior by using default parameters, variable-length arguments, or checking types inside the function to handle different input types.

EXAMPLE:

```
class Calculator:
    def add(self, a, b=0, c=0):
        """Adds two or three numbers depending on the arguments passed."""
        return a + b + c

# Create an object of Calculator class
calc = Calculator()

# Calling with two arguments
print(calc.add(7, 14))

# Calling with three arguments
print(calc.add(7, 14, 28))
```

```
.py
21
49
```

Example 2: Overloading with Variable-Length Arguments (*args)

You can use *args to accept a variable number of arguments.

```
class Calculator:
    def add(self, *args):
        """Adds any number of numbers."""
        return sum(args)

# Create an object of Calculator class
calc = Calculator()

# Calling with two arguments
print(calc.add(2, 4))

# Calling with three arguments
print(calc.add(2, 4, 6))
```

```
# Calling with more than three arguments
print(calc.add(2, 4, 6, 8, 10))
```

```
.py
6
12
30
```

Example 3: Overloading by Checking the Type of Arguments

You can simulate method overloading by checking the types of the arguments passed inside the function.

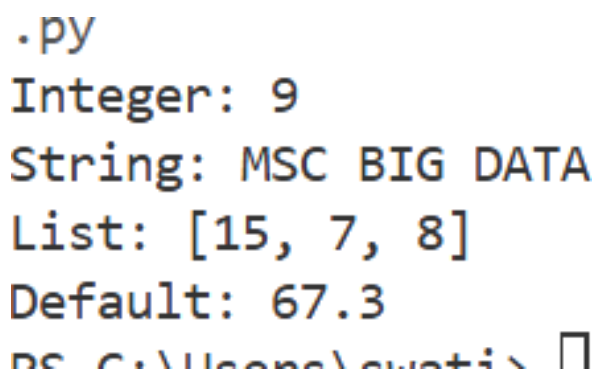
```
class Calculator:
  def multiply(self, a, b):
      """Multiplies either two numbers or concatenates strings."""
      if isinstance(a, int) and isinstance(b, int):
          return a * b
      elif isinstance(a, str) and isinstance(b, int):
          return a * b
      else:
          return "Invalid input types"
# Create an object of Calculator class
calc = Calculator()

# Multiply two integers
print(calc.multiply(6, 9))

# Repeat a string multiple times
print(calc.multiply("Hello", 5))

# Invalid input types
print(calc.multiply("Hello", "World"))
```

```
54
HelloHelloHelloHelloHello
Invalid input types
```

EXAMPLE 4: Overloading with @singledispatch (Type-Based Overloading)

Python's functools module provides a @singledispatch decorator to create type-based function overloading.

```
from functools import singledispatch
@singledispatch
def display(value):
    print("Default:", value)

@display.register(int)
def _(value):
    print("Integer:", value)

@display.register(str)
def _(value):
    print("String:", value)

@display.register(list)
def _(value):
    print("List:", value)

# Calling the overloaded function
display(9)
display("MSC BIG DATA")
display([15,7,8])
display(67.3)
```



PRACTICAL NO : 09

AIM : Working on Big Data Libraries: Numpy, Pandas, Matplotlib

Numpy:

NumPy (Numerical Python) is a powerful Python library widely used for numerical computing, data manipulation, and scientific computations. It provides support for arrays, matrices, and high-level mathematical functions to operate on these data structures efficiently.

Installation

To install NumPy, you can use pip:

-pip install numpy

Importing Numpy

To import NumPy, you can use:

-import numpy as np

A) NumPy Arrays

```
import numpy as np
arr=np.array([1,2,3,4,5])
print(arr)
arr2d=np.array([[1,2,3],[4,5,6]])
print(arr2d)
```

```
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
```

Array Operations

```
import numpy as np
a=np.array([1,2,3])
b=np.array([4,5,6])

# Element-wise addition
c=a+b
print("Addition:", c)

# Element-wise multiplication
```

```
d= a*b
print("Multiplication:",d)

# Broadcasting: scalar and array
e=a+10
print("Broadcasting (adding 10):",e)

# Summing all elements
print("Sum of elements:",np.sum(a))

 # Summing all elements
print("Mean of elements:",np.mean(a))
```

```
Addition: [5 7 9]
Multiplication: [ 4 10 18]
Broadcasting (adding 10): [11 12 13]
Sum of elements: 6
Mean of elements: 2.0
```

Indexing and Slicing

```
import numpy as np
arr = np.array([1,2,3,4,5])
print(arr[0])
print(arr[1:4])
arr[2:4]=[10,20]
print(arr)
```

```
1
[2 3 4]
[ 1  2 10 20  5]
```

Reshaping Arrays

```
import numpy as np
arr=np.arange(1,13)
print("Original array:",arr)

reshaped_arr=arr.reshape(3,4)
print("Reshaped array (3x4):\n", reshaped_arr)
flattened_arr= reshaped_arr.flatten()
print("Flattened array:", flattened_arr)
```

```
Original array: [ 1  2  3  4  5  6  7  8  9 10 11 12]
Reshaped array (3x4):
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
Flattened array: [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

Array Arithmetic

```
import numpy as np
arr1=np.array([1,2,3,4])
arr2=np.array([5,6,7,8])
print("Addition:", arr1+arr2)

print("Subtraction:", arr1-arr2)

print("Division:",arr1/arr2)
```

```
Addition: [ 6  8 10 12]
Subtraction: [-4 -4 -4 -4]
Division: [0.2        0.33333333 0.42857143 0.5       ]
```

Broadcasting

```
import numpy as np

arr1=np.array([1,2,3])
arr2=np.array([[4],[5],[6]])
result=arr2+arr1
print("Broadcasting result:\n",result)
```

```
Broadcasting result:
 [[5 6 7]
 [6 7 8]
 [7 8 9]]
```

Array Concatenation

```
import numpy as np
arr1= np.array([[1,2],[3,4]])
arr2=np.array([[5,6],[7,8]])
concat_rows=np.concatenate((arr1,arr2),axis=0)
```

```
print("Concatenated along rows:\n",concat_rows)

concat_cols=np.concatenate((arr1,arr2),axis=1)
print("Concatenated along columns:\n",concat_cols)
```

```
Concatenated along rows:
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
Concatenated along columns:
 [[1 2 5 6]
 [3 4 7 8]]
```

Array Transposition

```
import numpy as np
arr=np.array([[1,2],[3,4],[5,6]])
transposed=np.transpose(arr)
print("Original array:\n",arr)
print("Transposed array:\n", transposed)
```

```
Original array:
 [[1 2]
 [3 4]
 [5 6]]
Transposed array:
 [[1 3 5]
 [2 4 6]]
```

Stacking Arrays

```
import numpy as np
arr1=np.array([1,2,3])
arr2=np.array([4,5,6])
vert_stack=np.vstack((arr1,arr2))
print("Vertically stacked:\n",vert_stack)
horiz_stack=np.hstack((arr1,arr2))
print("Horizontally stacked:",horiz_stack)
```

```
Vertically stacked:
 [[1 2 3]
 [4 5 6]]
Horizontally stacked: [1 2 3 4 5 6]
```

Splitting Arrays

```
import numpy as np
arr=np.array([1,2,3,4,5,6])

split_arr=np.split(arr,3)
print("Splitted array:",split_arr)

arr2D = np.array([[1,2,3],[4,5,6],[7,8,9]])
split_arr2D=np.vsplit(arr2D,3)
print("Vertically split 2D array:\n",split_arr2D)
```

```
Splitted array: [array([1, 2]), array([3, 4]), array([5, 6])]
Vertically split 2D array:
 [array([[1, 2, 3]]), array([[4, 5, 6]]), array([[7, 8, 9]])]
```

Generate + Code + Markdown

Array Sorting

```
import numpy as np
arr=np.array([3,1,2,5,4])
sorted_arr=np.sort(arr)
print("Sorted array:",sorted_arr)
arr2D = np.array([[3,2,1],[5,4,6]])
sorted_arr2D= np.sort(arr2D,axis=1)
print("2D array sorted along rows :\n",sorted_arr2D)
```

```
Sorted array: [1 2 3 4 5]
2D array sorted along rows :
 [[1 2 3]
 [4 5 6]]
```

Matrix Operations

```
import numpy as np
```

```
matrix_a=np.array([[1,2],[3,4]])
matrix_b=np.array([[5,6],[7,8]])
matrix_product=np.dot(matrix_a,matrix_b)
print("Matrix multiplication:\n",matrix_product)

matrix_transpose=np.transpose(matrix_a)
print("Transpose of matrix A:\n",matrix_transpose)

matrix_inverse=np.linalg.inv(matrix_a)
print("inverse of matrix A:\n",matrix_inverse)
```

```
Matrix multiplication:
 [[19 22]
 [43 50]]
Transpose of matrix A:
 [[1 3]
 [2 4]]
inverse of matrix A:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

Random Array and Statistical Functions

```
import numpy as np
random_array=np.random.randn(5)
print("Random array:", random_array)
print("Standard Deviation:", random_array)
std_dev=np.std(random_array)
print("Standard Deviation:",std_dev)
random_ints=np.random.randint(1,10,(3,3))
print("Random 3x3 integer array:\n", random_ints)
max_value=np.max(random_ints)
min_value=np.min(random_ints)
print("Max value:", max_value)
print("Min value:",min_value)
```

```
Random array: [1.61514844 0.94748818 1.26234    0.84476172 0.22507378]
Standard Deviation: [1.61514844 0.94748818 1.26234    0.84476172 0.22507378]
Standard Deviation: 0.4631164096503744
Random 3x3 integer array:
 [[2 8 8]
 [2 3 3]
 [6 3 7]]
Max value: 8
Min value: 2
```

M.Using Numpy to solve linear equations

```
import numpy as np
```

```
# Coefficients of equation A =
np.array([[2, 1], [1,3]])

# Constants on the right-hand side B =
np.array([5, 7])

# Solving the system of linear equation
solution = np.linalg.solve(A, B) print("Solution
(x, y):", solution)
```

```
Solution (x, y): [1.6 1.8]
```

Pandas

Pandas is a powerful and widely-used Python library for data manipulation, analysis, and exploration. It is built on top of NumPy and provides data structures like DataFrame and Series that make it easier to work with structured data, particularly for tasks related to data cleaning, transformation, and analysis.

Creating a Pandas DataFrame

```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {'Name': ['Digvijay', 'Anuran', 'Chiranjiv', 'Ashutosh'],


        'Age': [21, 22, 23, 24],
        'City': ['Ahmedabad', 'Lucknow', 'Chennai', 'Mumbai']} df =
pd.DataFrame(data)
print("DataFrame from dictionary:\n", df)

# Creating a DataFrame from a list of lists data
= [['Digvijay', 21, 'Ahmedabad'],
        ['Anuran', 22, 'Lucknow'],
        ['Ashutosh', 24, 'Mumbai']]
DataFrame from dictionary:
         Name  Age       City=['Name', 'Age', 'City'])
0   Digvijay   21  Ahmedabad :\n", df2)
1     Anuran   22    Lucknow
2  Chiranjiv   23    Chennai
3   Ashutosh   24     Mumbai

DataFrame from list of lists:
        Name  Age       City
0  Digvijay   21  Ahmedabad
1    Anuran   22    Lucknow
2  Ashutosh   24     Mumbai
```

Reading data from a csv file.

Read data from a CSV file into a Pandas

```
First 5 rows of the DataFrame:
       Name  Basic Pay  TA(4%)  DA(6%)  Gross Salary  PF(3%)  Net Salary
0    Ankush      33000    1320    1980         36300    1089       35211
1    Yogesh      25000    1000    1500         27500     825       26675
2    Khushi      31000    1240    1860         34100    1023       33077
3    Ayushi      59000    2360    3540         64900    1947       62953
4   Anubhav      46000    1840    2760         50600    1518       49082

Column names: Index(['Name', 'Basic Pay', 'TA(4%)', 'DA(6%)', 'Gross Salary', 'PF(3%)',
       'Net Salary'],
      dtype='object')
```

```
# Display the column names print("\nColumn
names:", df.columns)
```

Selecting Data(Selecting and Indexing)

Access specific rows and columns in a DataFrame.

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Anuran', 'Prem', 'Mukesh', 'Sunny'], 'Age':
       [21, 22, 23, 24],
       'City': ['Mumbai', 'Delhi', 'Bengaluru', 'Gurugram']} df =
pd.DataFrame(data)

# Selecting a single column print("Age
column:\n", df['Age'])

# Selecting multiple columns
print("\nName and City columns:\n", df[['Name', 'City']])

# Selecting rows by index (iloc: integer location)
print("\nFirst two rows:\n", df.iloc[:2])

# Selecting rows by condition
print("\nRows where Age > 22:\n", df[df['Age'] > 22])
```

Adding and Removing Columns

Add a new column or remove an existing column in a DataFrame.

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Prem', 'Satyendra', 'Mahesh'],
        'Age': [25, 30, 35],
        'City': ['Bhubaneshwar', 'Pune', 'Lucknow']}
df = pd.DataFrame(data)

# Adding a new column
df['Salary'] = [50000, 60000, 70000]
print("DataFrame after adding a new column:\n", df)

# Removing a column
df = df.drop('Salary', axis=1)
print("\nDataFrame after removing the Salary column:\n", df)
```

```
DataFrame after adding a new column:
         Name  Age          City  Salary
0       Prem   25  Bhubaneshwar   50000
1  Satyendra   30          Pune   60000
2     Mahesh   35       Lucknow   70000

DataFrame after removing the Salary column:
         Name  Age          City
0       Prem   25  Bhubaneshwar
1  Satyendra   30          Pune
2     Mahesh   35       Lucknow
```

```
First two rows:
      Name  Age    City
0   Anuran   21  Mumbai
1     Prem   22   Delhi

Rows where Age > 22:
      Name  Age       City
2   Mukesh   23  Bengaluru
3    Sunny   24   Gurugram
```

Sorting Data

Sort the data in a DataFrame by a specific column.

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Prem', 'Mahesh', 'Bindu', 'Angel'],
        'Age': [40, 25, 35, 30],
        'City': ['Dubai', 'Capetown', 'Canberra', 'Wellington']}
df = pd.DataFrame(data)

# Sorting by age in ascending order
sorted_df = df.sort_values(by='Age')
print("DataFrame sorted by Age (ascending):\n", sorted_df)

# Sorting by name in descending order
sorted_df = df.sort_values(by='Name', ascending=False)
print("\nDataFrame sorted by Name (descending):\n", sorted_df)
```

Output:

```
DataFrame sorted by Age (ascending):
      Name  Age        City
1   Mahesh   25    Capetown
3    Angel   30  Wellington
2    Bindu   35    Canberra
0     Prem   40       Dubai

DataFrame sorted by Name (descending):
      Name  Age        City
0     Prem   40       Dubai
1   Mahesh   25    Capetown
2    Bindu   35    Canberra
3    Angel   30  Wellington
```

GroupBy and Aggregation

Group the data by a specific column and apply aggregate functions like sum, mean, etc.

```
import pandas as pd

# Sample DataFrame
data = {'Department': ['HR', 'IT', 'HR', 'Finance', 'IT','Finance'],
        'Employee': ['Sonal', 'Anita', 'Kiran', 'Rahul', 'Virat','Gulfisha'],
        'Salary': [50000, 60000, 55000, 65000, 62000,68000]}
df = pd.DataFrame(data)

# Grouping by department and calculating the mean salary
grouped_df = df.groupby('Department')['Salary'].mean()
print("Mean salary by department:\n", grouped_df)

# Grouping by department and calculating the sum of salaries
grouped_sum = df.groupby('Department')['Salary'].sum()
print("\nTotal salary by department:\n", grouped_sum)
```

Output:

```
Mean salary by department:
 Department
Finance    66500.0
HR         52500.0
IT         61000.0
Name: Salary, dtype: float64

Total salary by department:
 Department
Finance    133000
HR         105000
IT         122000
Name: Salary, dtype: int64
```

Handling Missing Data

Handle missing values by filling or dropping them.

```
import pandas as pd
import numpy as np

# Sample DataFrame with missing values
data = {'Name': ['Anuran', 'Balwant', 'Raju', np.nan],
        'Age': [25, np.nan, 35, 40],

'City': ['Pune', np.nan, 'Hydreabad', 'Bengaluru']} df =
pd.DataFrame(data)

# Dropping rows with missing values
df_dropped = df.dropna()
print("DataFrame after dropping rows with missing values:\n", df_dropped)

# Filling missing values with a default value
df_filled = df.fillna({'Name': 'Prem', 'Age': 30, 'City': 'Noida'})
print("\nDataFrame after filling missing values:\n", df_filled)
```

```
DataFrame after dropping rows with missing values:
      Name   Age       City
0  Anuran  25.0       Pune
2    Raju  35.0  Hydreabad

DataFrame after filling missing values:
       Name   Age       City
0   Anuran  25.0       Pune
1  Balwant  30.0      Noida
2     Raju  35.0  Hydreabad
3     Prem  40.0  Bengaluru
```

Merging and Joining Data-Frames

Merge or join two DataFrames.

```
import pandas as pd

 # Sample DataFrames
 df1 = pd.DataFrame({'Employee': ['Sonal', 'Kiran', 'Gulfisha'],
                     'Department': ['HR', 'IT', 'Finance']})
 df2 = pd.DataFrame({'Employee': ['Sonal', 'Rachana', 'Gulfisha'],
                     'Salary': [50000, 60000, 55000]})

 # Merging DataFrames on the 'Employee' column
 merged_df = pd.merge(df1, df2, on='Employee', how='inner')
 print("Merged DataFrame (inner join):\n", merged_df)

 # Left join
 left_join_df = pd.merge(df1, df2, on='Employee', how='left')
 print("\nLeft join DataFrame:\n", left_join_df)
```

```
Merged DataFrame (inner join):
    Employee Department  Salary
0      Sonal         HR   50000
1  Gulfisha    Finance   55000

Left join DataFrame:
    Employee Department   Salary
0      Sonal         HR  50000.0
1      Kiran         IT      NaN
2  Gulfisha    Finance  55000.0
```

Pivot Tables

Create pivot tables to summarize data.

```
import pandas as pd

# Sample DataFrame
data = {'Department': ['HR', 'HR', 'IT', 'Finance', 'IT'],
        'Employee': ['Sonal', 'Rohit', 'Saqlain','Disha', 'Arvind'],
        'Salary': [50000, 55000, 60000, 65000, 62000]}
df = pd.DataFrame(data)

# Creating a pivot table showing the sum of salaries by department
pivot_table = df.pivot_table(values='Salary', index='Department', aggfunc='sum')
print("Pivot table (sum of salaries by department):\n", pivot_table)
```

Output:

```
Pivot table (sum of salaries by department):
             Salary
Department
Finance       65000
HR           105000
IT           122000
```

Exporting Data to CSV

Save a DataFrame to a CSV file.

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Anuran', 'Aarohi', 'Shivam', 'Rahul'],
        'Basic Pay':[69000, 71000, 55000, 45000],}
df = pd.DataFrame(data)

# Exporting the DataFrame to a CSV file
df.to_csv('Salary Sheet.csv', index=False)
print("DataFrame exported to 'Salary Sheet.csv'")
```

Output:

```
====== RESTART: C:/Users/anura/OneDrive/P:
DataFrame exported to 'Salary Sheet.csv'
```

	A	B
1	Name	Basic Pay
2	Anuran	69000
3	Aarohi	71000
4	Shivam	55000
5	Rahul	45000

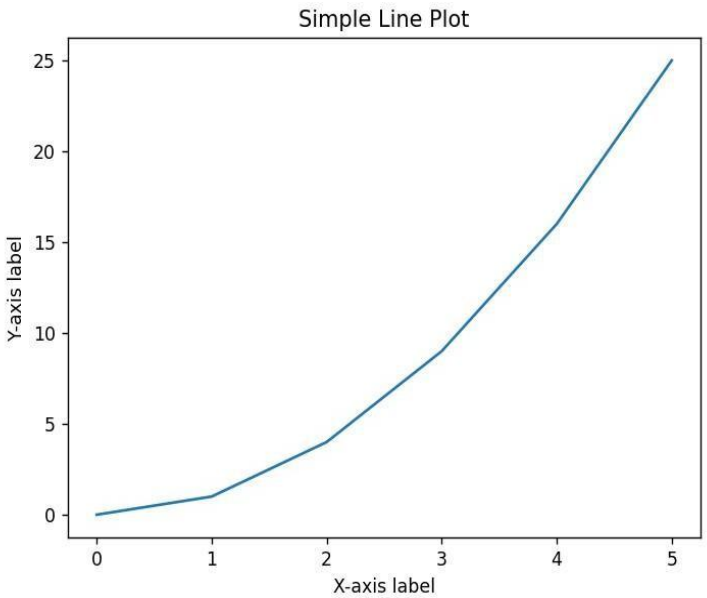
Matplotlib

Matplotlib is a popular Python library used for creating static, animated, and interactive visualizations. It provides a flexible way to generate a wide variety of plots and charts, making it essential for data analysis, exploration, and presentation. Matplotlib is commonly used in conjunction with other libraries like NumPy and Pandas for data science tasks

Basic Line Plot

This program shows how to create a simple line plot with labels and a title.

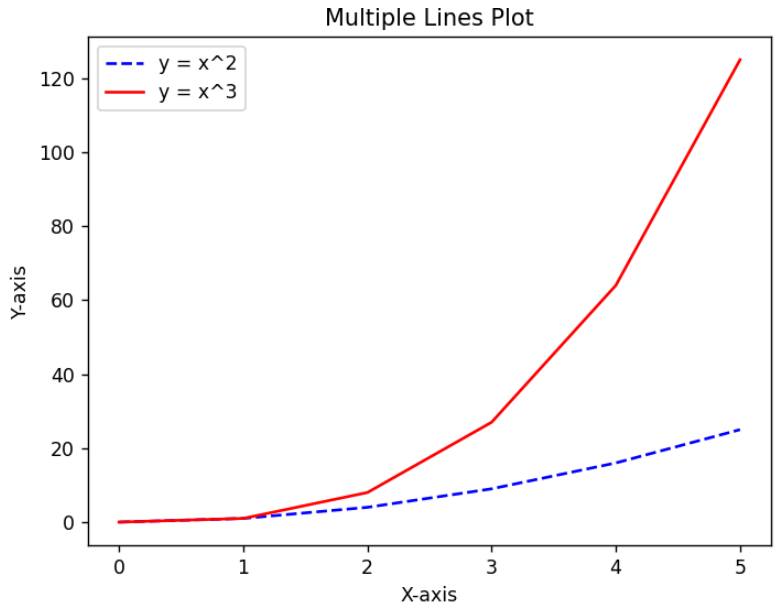
```
import matplotlib.pyplot as plt

# Data for plotting
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Creating the plot
plt.plot(x, y)

# Adding labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Simple Line Plot')
# Display the plot
plt.show()
```



Multiple Lines on the Same Plot

This example plots multiple lines on the same graph with different styles.

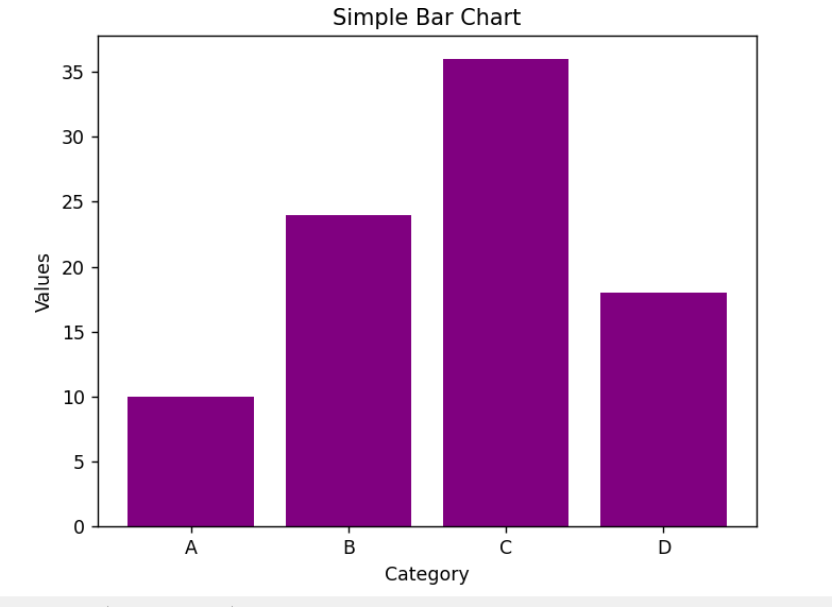
```
import matplotlib.pyplot as plt

# Data for plotting x =
[0, 1, 2, 3, 4, 5]
y1 = [0, 1, 4, 9, 16, 25]
y2 = [0, 1, 8, 27, 64, 125]

# Plotting multiple lines
plt.plot(x, y1, label='y = x^2', color='blue', linestyle='--')
plt.plot(x, y2, label='y = x^3', color='red', linestyle='-')

# Adding labels and title
plt.xlabel('X-axis') plt.ylabel('Y-
axis') plt.title('Multiple Lines
Plot')
plt.legend()

# Display the plot
plt.show()
```



Bar Chart

Create a bar chart to compare different categories

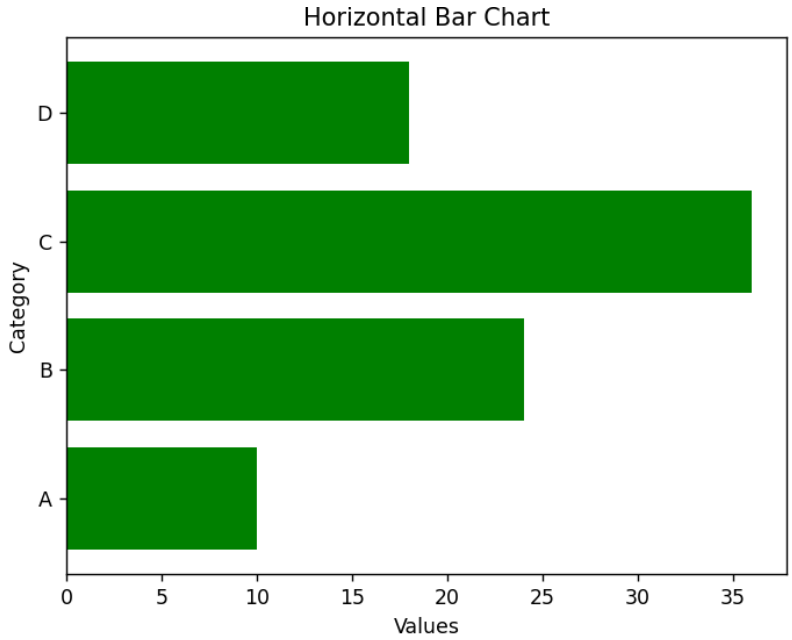
```
import matplotlib.pyplot as plt

# Data for plotting
categories = ['A', 'B', 'C', 'D']
values = [10, 24, 36, 18]

# Creating the bar chart
plt.bar(categories, values, color='purple')

# Adding labels and title
plt.xlabel('Category')
plt.ylabel('Values')
plt.title('Simple Bar Chart')

# Display the plot
plt.show()
```



Horizontal Bar Chart

This example demonstrates how to create a horizontal bar chart.

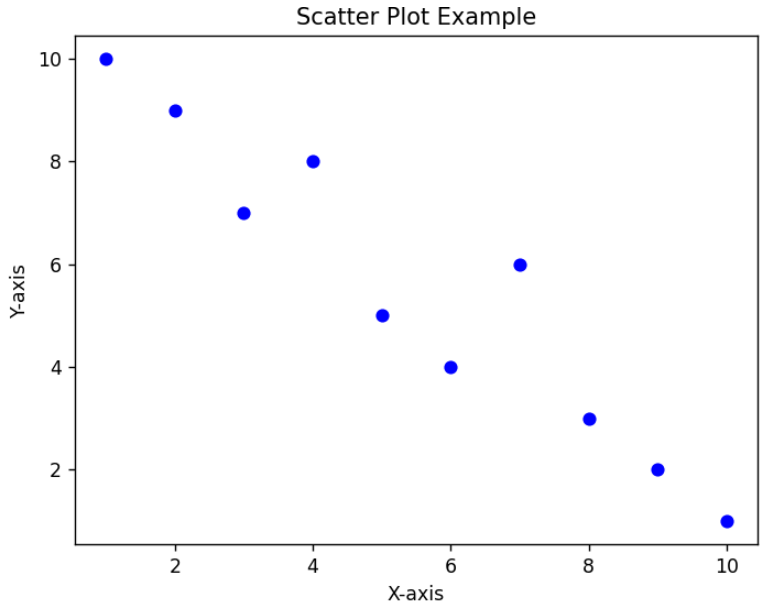
```
import matplotlib.pyplot as plt

# Data for plotting
categories = ['A', 'B', 'C', 'D']
values = [10, 24, 36, 18]

# Creating the horizontal bar chart
plt.barh(categories, values, color='green')

# Adding labels and title
plt.xlabel('Values')
plt.ylabel('Category')
plt.title('Horizontal Bar Chart')

# Display the plot
plt.show()
```



Scatter Plot

This program shows how to create a scatter plot to visualize relationships between two variables.

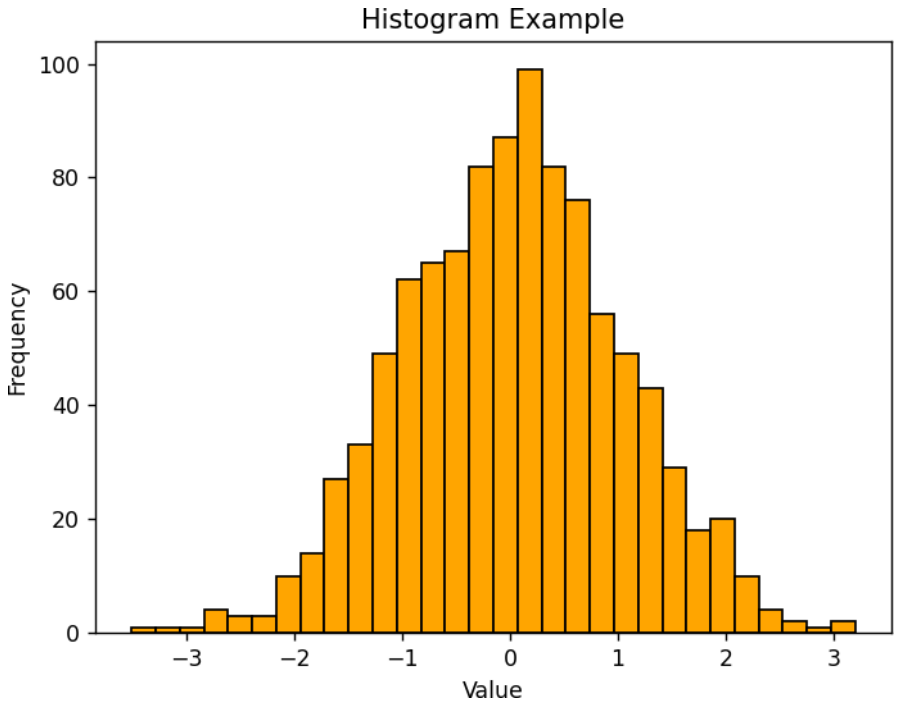
```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [10, 9, 7, 8, 5, 4, 6, 3, 2, 1]

# Creating the scatter plot
plt.scatter(x, y, color='blue', marker='o')

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot Example')

# Display the plot
plt.show()
```



Histogram

Create a histogram to visualize the distribution of data.

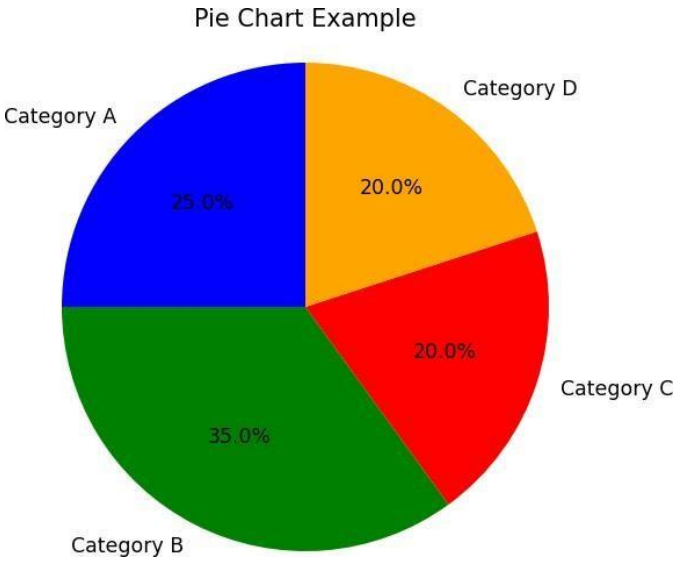
```
import matplotlib.pyplot as plt
import numpy as np

# Generating random data
data = np.random.randn(1000)

# Creating the histogram
plt.hist(data, bins=30, color='orange', edgecolor='black')

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')

# Display the plot
plt.show()
```



Pie Chart

This program demonstrates how to create a pie chart to represent the proportion of categories.

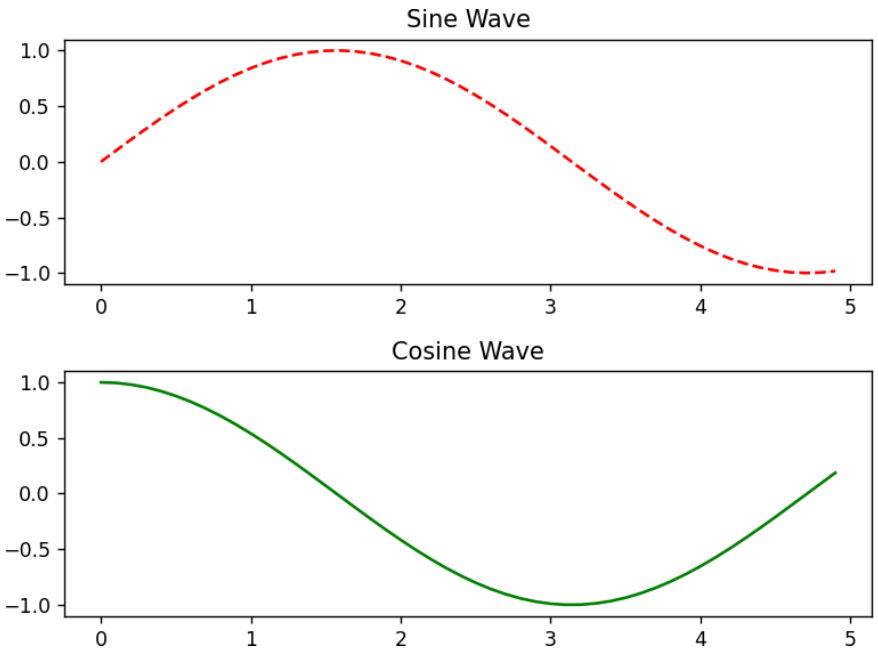
```
import matplotlib.pyplot as plt

# Data for plotting
labels = ['Category A', 'Category B', 'Category C', 'Category D']
sizes = [25, 35, 20, 20]
colors = ['blue', 'green', 'red', 'orange']

# Creating the pie chart
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)

# Adding title
plt.title('Pie Chart Example')

# Equal aspect ratio ensures that pie is drawn as a circle
plt.axis('equal')

# Display the plot
plt.show()
```



Subplots (Multiple Plots)

This example shows how to create multiple subplots in the same figure.

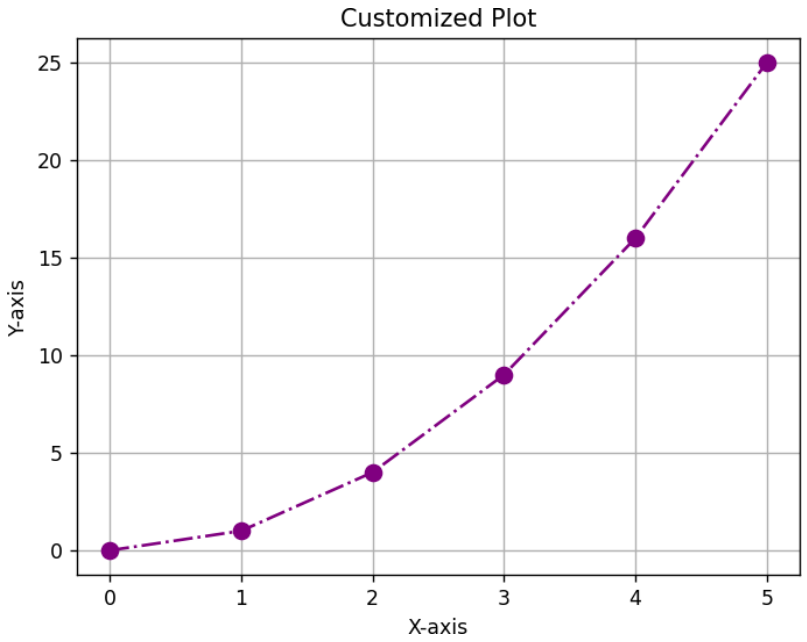
```
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
x = np.arange(0, 5, 0.1)
y1 = np.sin(x)
y2 = np.cos(x)

# Creating subplots
plt.subplot(2, 1, 1)
plt.plot(x, y1, 'r--')
plt.title('Sine Wave')

plt.subplot(2, 1, 2)
plt.plot(x, y2, 'g-')
plt.title('Cosine Wave')

# Adjust layout
plt.tight_layout()

# Display the plot
plt.show()
```



Customizing Plot Appearance

Customize the plot appearance, including grid lines, color, and style.

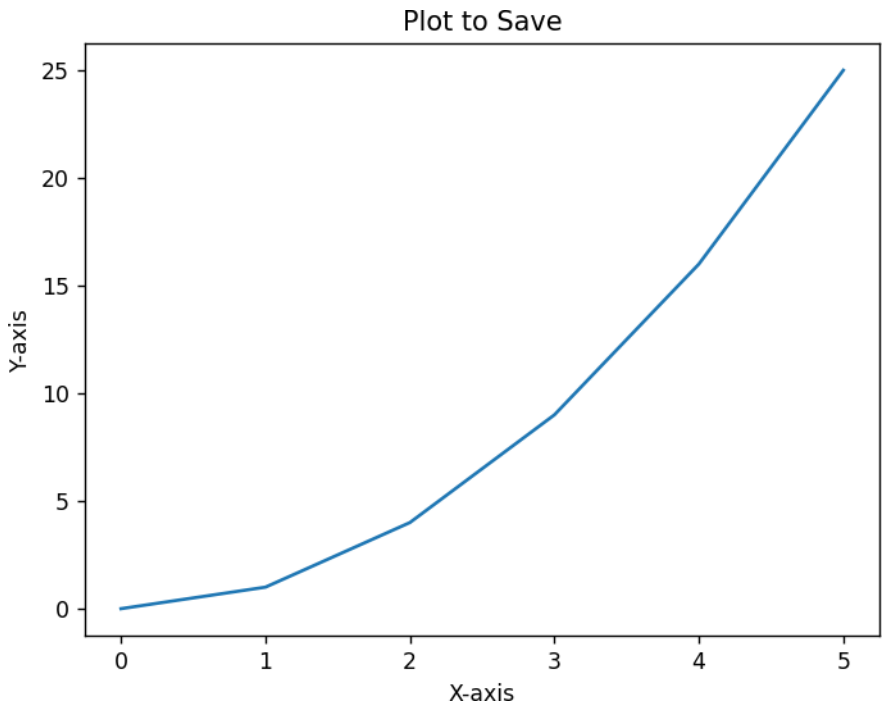
```
import matplotlib.pyplot as plt

# Data for plotting
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Creating the plot
plt.plot(x, y, color='purple', linestyle='-.', marker='o', markersize=8)

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Customized Plot')

# Adding grid lines
plt.grid(True)

# Display the plot
plt.show()
```



Saving a Plot to a File

This program demonstrates how to save a plot to a file (e.g., PNG, PDF).

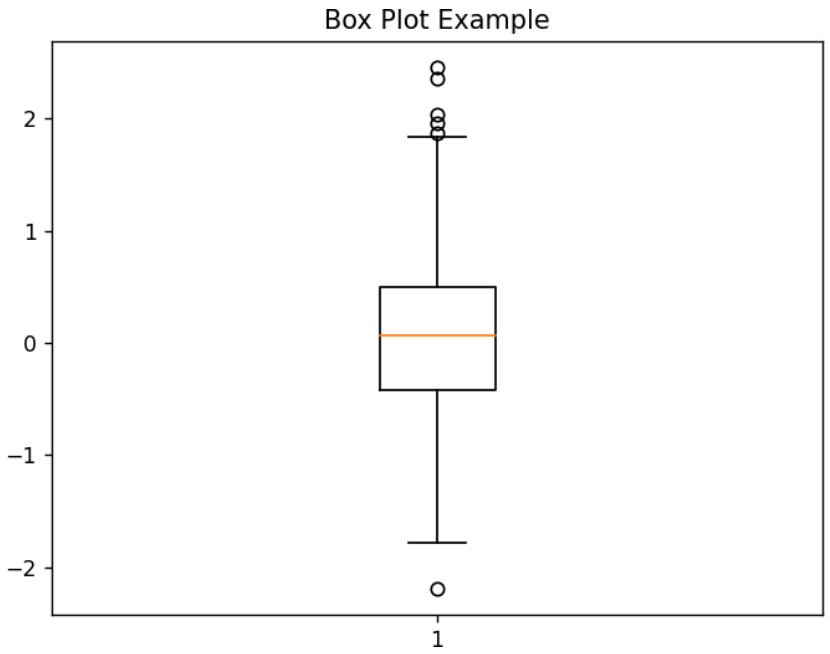
```
import matplotlib.pyplot as plt

# Data for plotting
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Creating the plot
plt.plot(x, y)

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plot to Save')

# Saving the plot to a PNG file
plt.savefig('plot_example.png')

# Display the plot
plt.show()
```



Box Plot

Create a box plot to display data distribution and outliers.

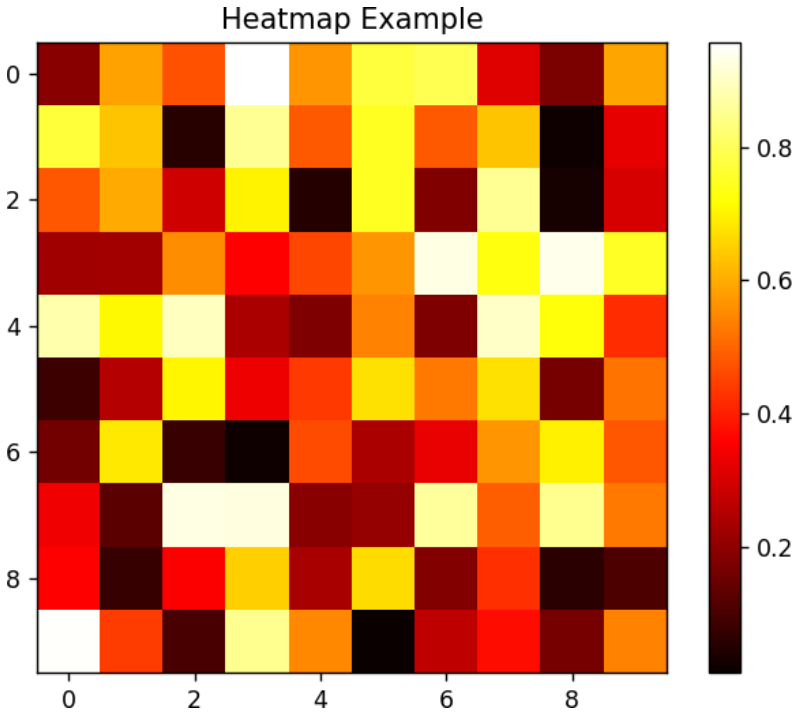
```
import matplotlib.pyplot as plt
import numpy as np

# Generating random data
data = np.random.randn(100)

# Creating the box plot
plt.boxplot(data)

# Adding title
plt.title('Box Plot Example')

# Display the plot
plt.show()
```



Heatmap

This program demonstrates how to create a heatmap to visualize matrix-like data.

```
import matplotlib.pyplot as plt
import numpy as np

# Generating data for the heatmap
data = np.random.rand(10, 10)

# Creating the heatmap
plt.imshow(data, cmap='hot', interpolation='nearest')

# Adding a color bar
plt.colorbar()

# Adding title
plt.title('Heatmap Example')

# Display the plot
plt.show()
```



Annotations on a Plot

Add annotations to highlight specific points in the plot.

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Creating the plot
plt.plot(x, y, marker='o')

# Adding annotations
plt.annotate('Max Value', xy=(5, 25), xytext=(3, 20),
             arrowprops=dict(facecolor='black', shrink=0.05))

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plot with Annotations')

# Display the plot
plt.show()
```