NAME – GOPAL KUMAR
Assignment – Mile stone test
EMAIL ID – krgopal0301@gmail.com
Phone no- 9304026446

Question 1.1: Write the Answer to these questions.

Note: Give at least one example for each of the questions

A. What is the difference between static and dynamic variables in Python?

Ans – In Python, **static variables** (or class variables) are shared across all instances of a class, whereas **dynamic variables** (or instance variables) are unique to each instance of a class.

Static Variable:

- Defined inside a class but outside any instance methods.
- Shared among all instances of the class.

Dynamic Variable:

- Defined inside an instance method, usually within __init__.
- Unique to each instance of the class.

Example -

```
class Example:
    static_var = 0  # Static variable

    def __init__(self, dynamic_var):
        self.dynamic_var = dynamic_var  # Dynamic variable

# Create two instances
obj1 = Example(10)
obj2 = Example(20)

# Access static variable
print(Example.static_var)

# Access dynamic variables
print(obj1.dynamic_var)
print(obj2.dynamic_var)
```

[1] ✓ 0.0s

```
0
10
20
```

B. Explain the purpose of "pop","popitem","clear()" in a dictionary with suitable examples.

Ans -The pop(), popitem(), and clear() methods are essential for managing the content of a dictionary.

Example -

- pop() is used to remove a specific key-value pair.

```
student={'Name':'Gopal','Age':23,'Course':'Data science'}
# remove Age with help of pop
age=student.pop('Age')
print(student)
```

[14] ✓ 0.0s

```
{'Name': 'Gopal', 'Course': 'Data science'}
```

❖ popitem() is used to remove an arbitrary key-value pair.

```
student={'Name':'Gopal','Age':23,'Course':'Data science'}
#popitem() is used to remove last pair in the dictionary
age=student.popitem()
print(student)
```

[19] ✓ 0.0s

```
{'Name': 'Gopal', 'Age': 23}
```

❖ clear() is used to empty the entire dictionary.

```
student={'Name':'Gopal','Age':23,'Course':'Data science'}
#clear() is used to empty the entire dictionary.
age=student.clear()
print(student)
```

[20] ✓ 0.0s

```
{}
```

C. What do you mean by FrozenSet? Explain it with suitable examples .

Ans - In Python, a frozenset is an immutable version of a set. This means that once you create a frozenset, you cannot add or remove elements from it.

Example -

```
set={1,2,3}
frozenset=frozenset(set)
print(frozenset)
```

[1] ✓ 0.0s

```
frozenset({1, 2, 3})
```

D. Differentiate between mutable and immutable data types in Python and give examples of mutable and immutable data types.

Ans –

- Mutable- Values can be changed or assigned new values. Examples include lists, dictionaries, and sets.

  Example –

```
Mutable=[1,2,3]
Mutable.append(4)
print(Mutable)
```

[2] ✓ 0.0s

```
[1, 2, 3, 4]
```

- Immutable- Values cannot be modified after creation. Examples include integers, floats, strings, booleans, and tuples

  Example –

```
Immutable=(1,2,3)
Immutable[0]=4
```

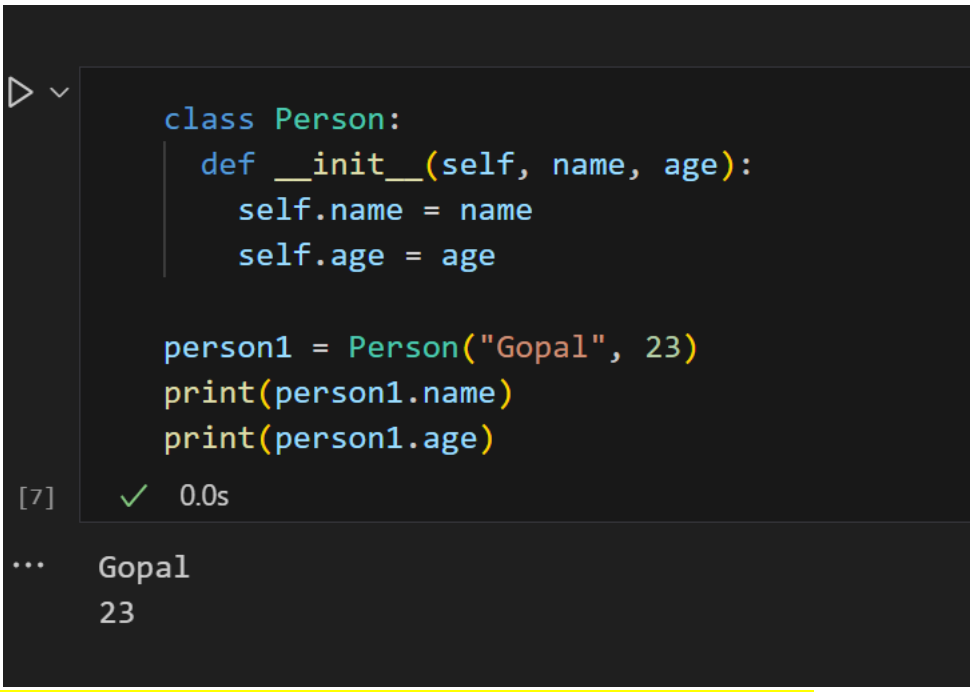
[5] ⊗ 0.0s

```
---------------------------------------------------------------------
TypeError                                 Traceback (most recent c
Cell In[5], line 2
      1 Immutable=(1,2,3)
----> 2 Immutable[0]=4

TypeError: 'tuple' object does not support item assignment
```

E. What is __init__?Explain with an example

Ans – In Python, __init__ is a special method, also known as a constructor. It is automatically called when you create a new object from a class. The purpose of __init__ is to initialize the object's attributes with the values provided at the time of creation.

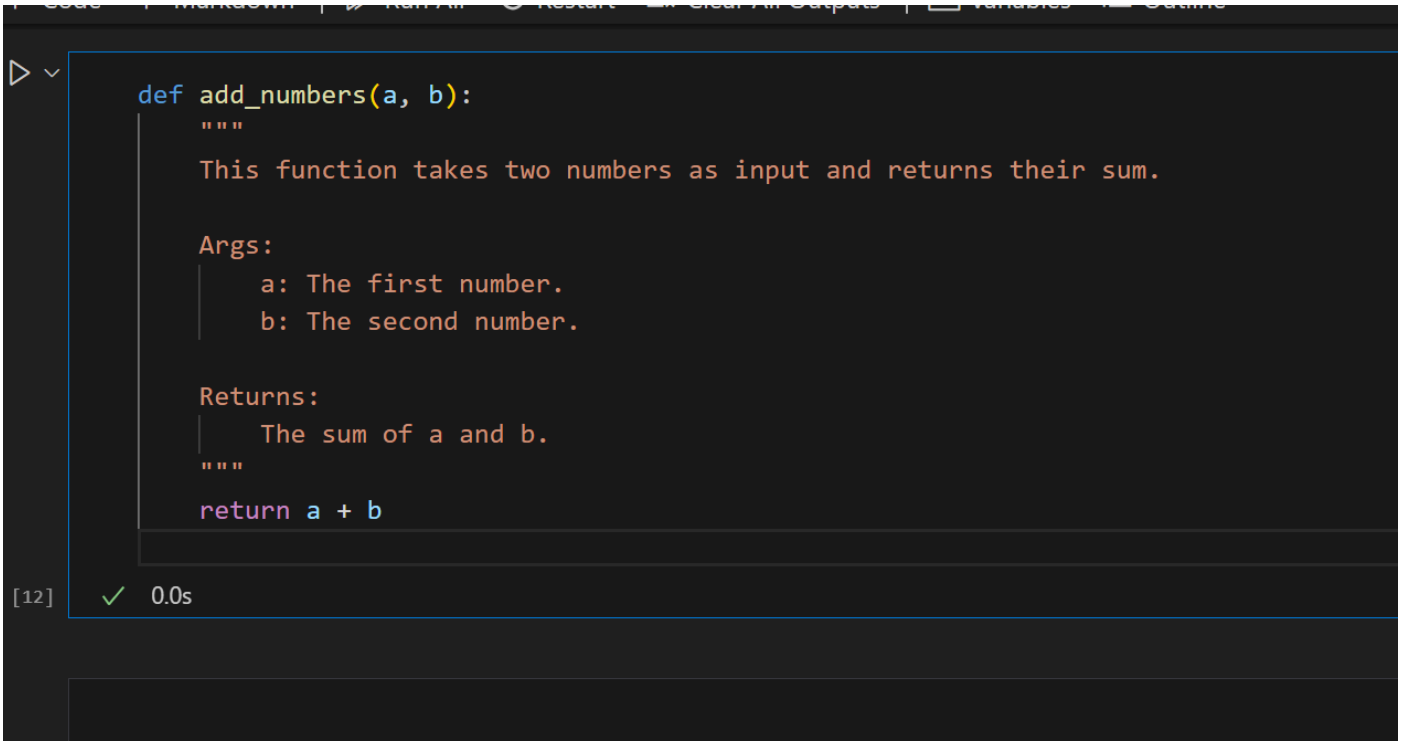
Example –



F. What is docstring in Python?Explain with an example.

Ans – A Python docstring is a string used to document a Python module, class, function or method, so programmers can understand what it does without having to read the details of the implementation.

Example –



G. What are unit tests in Python?

Ans - Unit tests in Python are small, automated tests that verify the functionality of individual units of code, such as functions or methods.

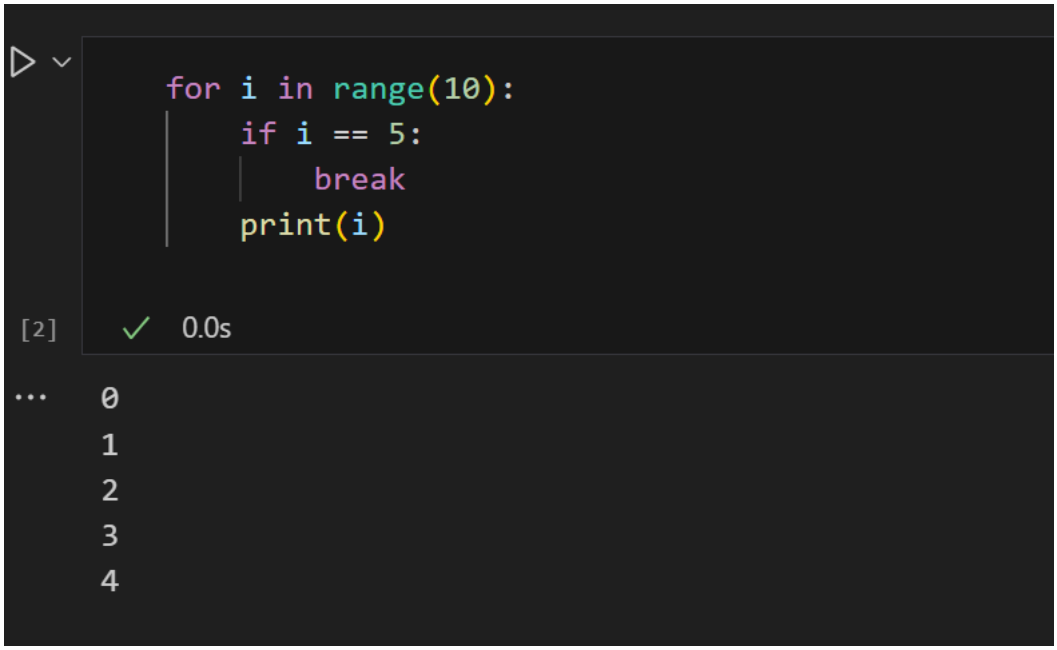
H. What is break, continue and pass in Python?

Ans – In Python, break, continue, and pass are keywords used to control the flow of execution within loops:

1. Break:

- Terminates the loop prematurely.
- Execution jumps to the next statement after the loop.
- Useful when you want to exit a loop based on a specific condition.

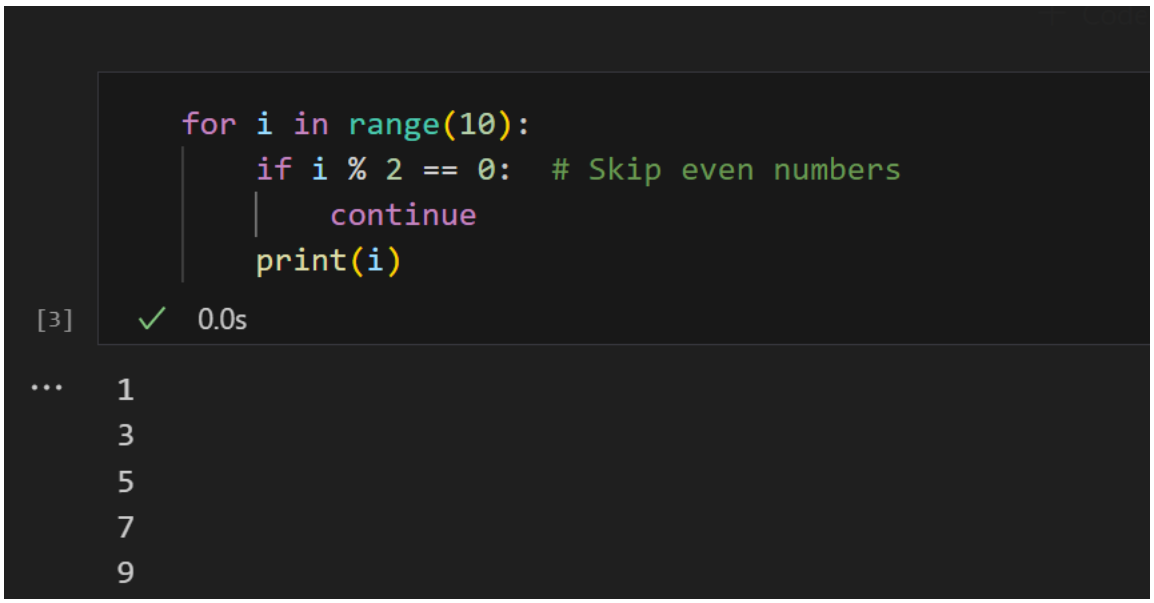
Example –



2. Continue:

- Skips the remaining code in the current iteration of the loop.
- Execution jumps to the next iteration.
- Useful when you want to ignore certain elements based on a condition.

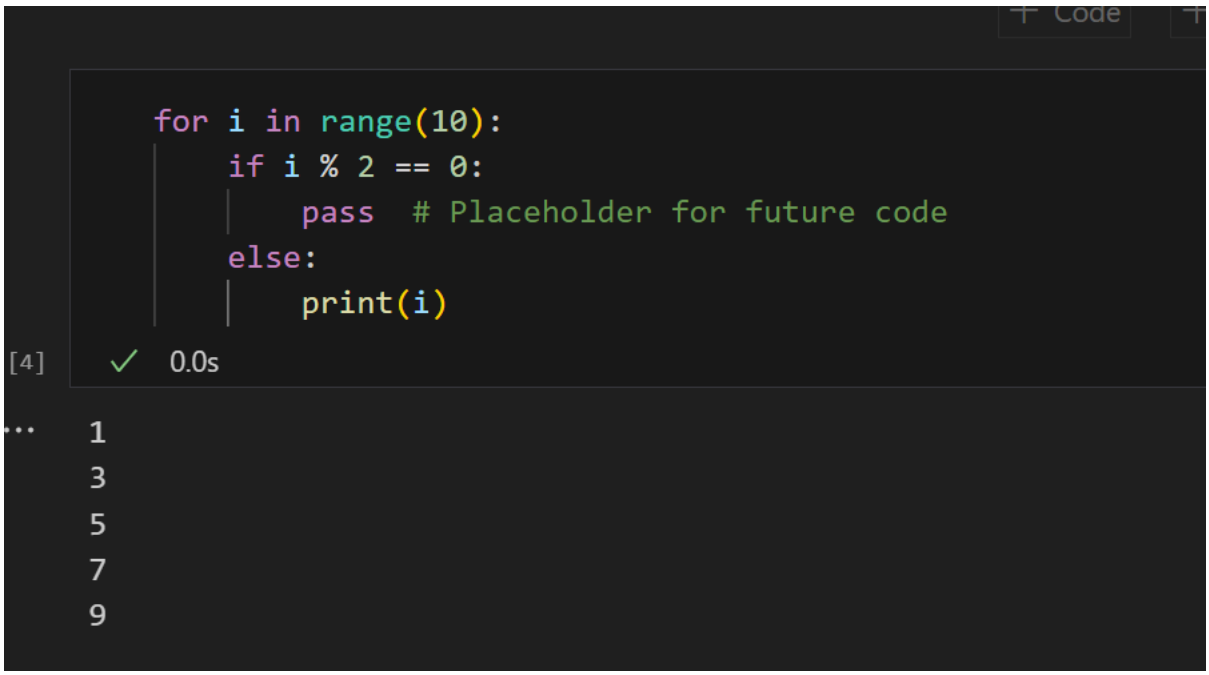
Example –



3. Pass:

- Does nothing. It's a null operation.
- Used as a placeholder in situations where a statement is syntactically required, but you don't want any command or code to execute.

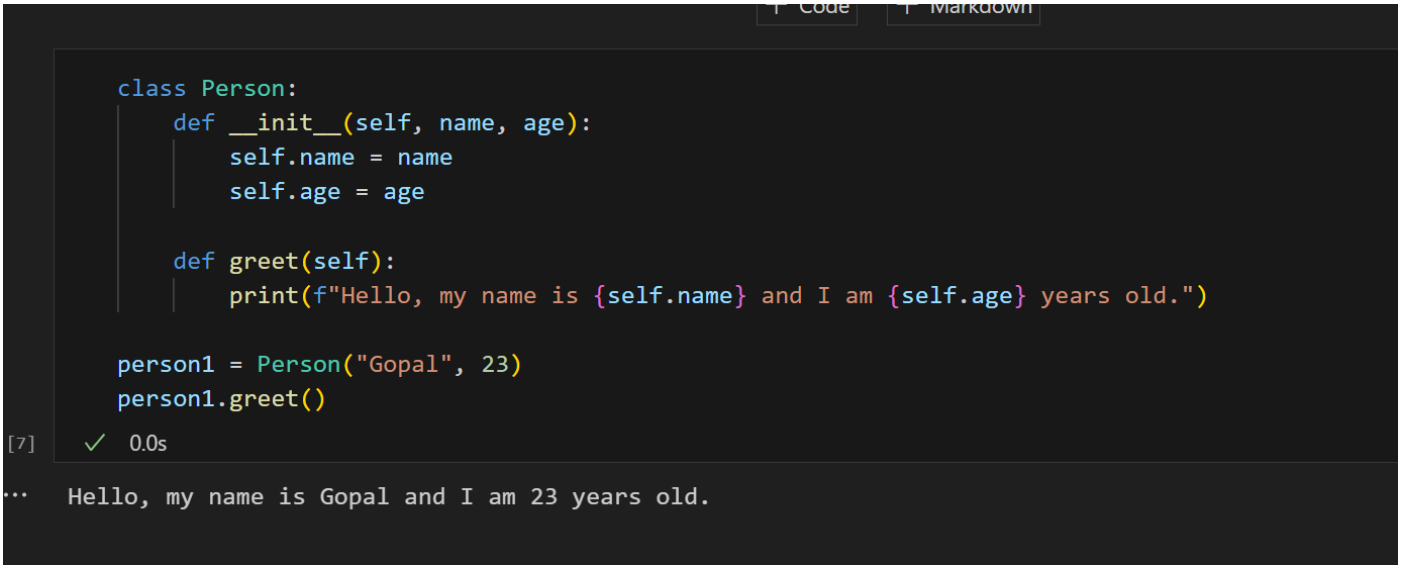
  Example –



I. What is the use of self in Python?

Ans - Self is a special parameter that refers to the instance of a class. It allows you to access the attributes and methods of that specific instance.

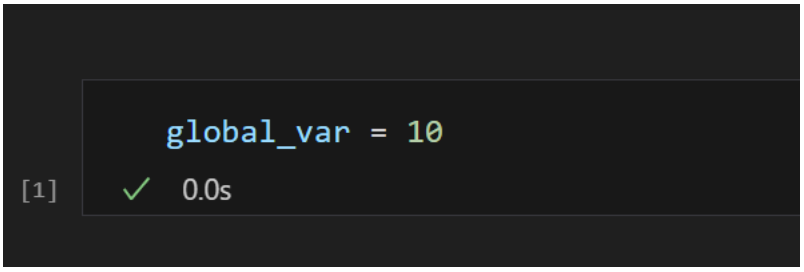
Example –



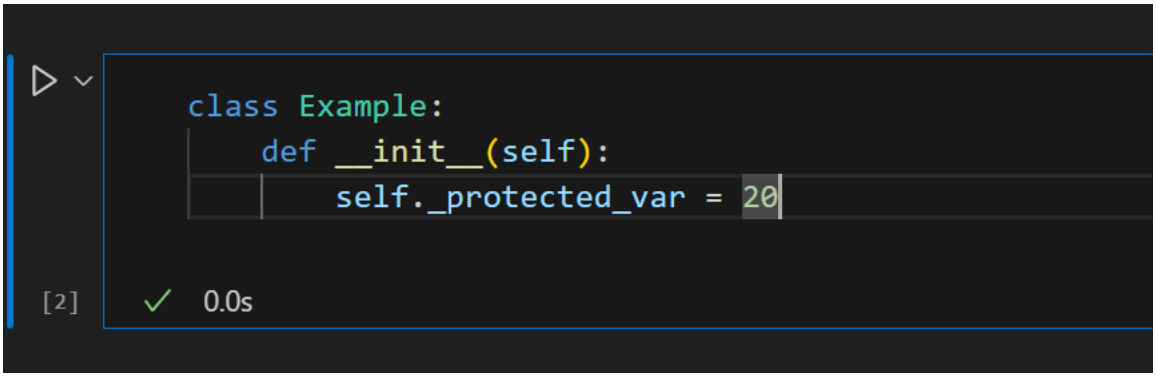
J. What are global, protected and private attributes in Python?

Ans - There are three types of access modifiers in Python: global(public), private, and protected. Variables with the public access modifiers can be accessed anywhere inside or outside the class, the private variables can only be accessed inside the class, while protected variables can be accessed within the same package.

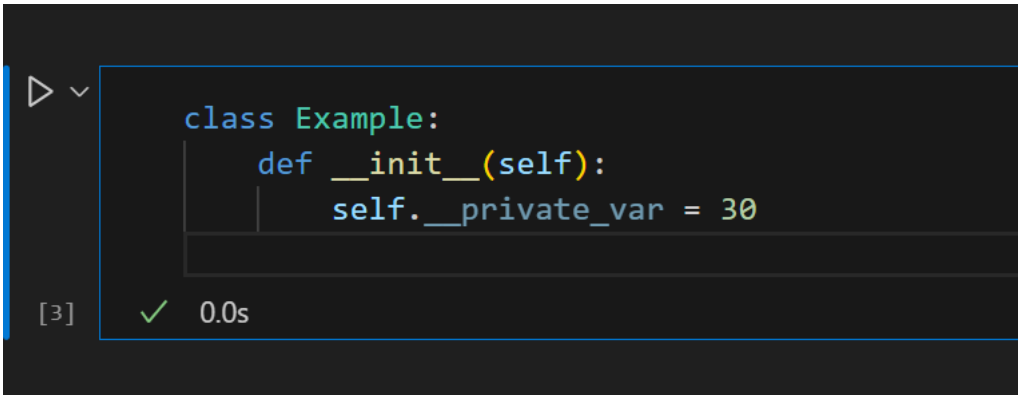
1. **Global Attributes**: These are variables defined outside of any class or function. They are accessible from any part of the program.
   Example –



2. **Protected Attributes**: These are indicated by a single underscore prefix (_attribute). They are intended for internal use within a class or its subclasses, though they can still be accessed from outside the class.



3. **Private Attributes**: These are indicated by a double underscore prefix (__attribute). They are meant to be inaccessible from outside the class, as Python performs name mangling to make them harder to access.



K. What are modules and packages in Python?

Ans - **Modules** and **packages** in Python help organize and reuse code.

1. **Modules:**

- A module is a single file containing Python code that can define functions, classes, and variables.
- You can import and reuse the code in other Python files.
- Example: If you have a file my_module.py with a function greet(), you can import it into another script using:

```
import my_module
my_module.greet()
```

2. **Packages:**

- A package is a collection of related modules organized in a directory hierarchy.
- It allows you to group multiple modules together.
- A directory is considered a package if it contains a special __init__.py file.

```
mypackage/
    __init__.py
    module1.py
    module2.py
```

L. What are lists and tuples? What is the key difference between the two?

Ans - In Python, both lists and tuples store and organize data collections, but the key difference is that lists are mutable while tuples are immutable.

```
#Example
#List:
i = [10, 20, 30, 40]
#Tuple:
t=(10, 20, 20, 40)
type(i),type(t)
```

[11] ✓ 0.0s

```
(list, tuple)
```

M. What is an Interpreted language & dynamically typed language?Write 5 differences between them?

Ans - An interpreted language is a programming language that executes instructions directly without compiling a program into machine language first. Dynamic typing is a feature of some programming languages that determines the data type of a variable based on its value. Here are some differences between interpreted languages and dynamically typed languages:

- Compilation

Interpreted languages compile code on the fly each time the program runs, while compiled languages only compile code once into machine code.

- Speed

Compiled programs run faster than interpreted programs.

- Development and testing

Interpreted languages are easier to develop and test because errors are reported line by line.

- Code

Dynamically typed languages can produce more concise code because you don't spend as much time defining and casting types.

- Flexibility

Dynamically typed languages can be versatile and may be suitable for small-scale applications that don't require frequent updates.

Example –

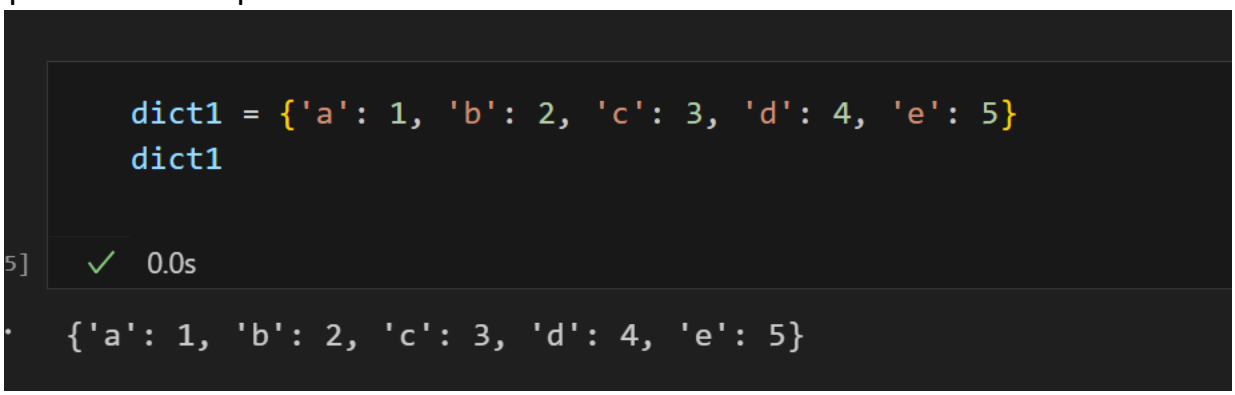
- Dynamically- Groovy
- Interpreted – Python is An interpreted and dynamically typed language

N. What are Dict and List comprehensions?

Ans - In Python, list and dictionary comprehensions are ways to create lists and dictionaries in a concise and expressive manner.

Example –

- Example of Dict comprehensions –



- Example of List comprehensions –

```
#Returning only even numbers:
x = [i for i in range(10) if i > 5]
print(x)
```

[13] ✓ 0.0s

```
[6, 7, 8, 9]
```

O. What are decorators in Python? Explain it with an example.Write down its use cases.
Ans - In Python, a decorator is a function that allows you to modify the behavior of another function without changing its original code.
Example –

```
def log_execution(func):
  def wrapper(*args, **kwargs):
    print(f"Starting execution of {func.__name__}")
    result = func(*args, **kwargs)
    print(f"Finished execution of {func.__name__}")
    return result
  return wrapper

@log_execution
def add(x, y):
  return x + y

result = add(5, 3)
print(result)
```

[16] ✓ 0.0s

```
Starting execution of add
Finished execution of add
8
```

Use cases for decorators include:

- Adding logging or error handling: to functions without cluttering their code.
- Caching the results of a function: to improve performance.
- Validating the arguments: passed to a function.
- Measuring the execution time: of a function.
- Adding authorization checks: to functions.

P. How is memory managed in Python?

Ans - Python manages memory through a private heap space, using a combination of reference counting and garbage collection. Here's how it works

Example –

```
a = 10  # A new integer object is created in memory
b = a   # 'b' references the same object as 'a'

# 'a' and 'b' now point to the same memory location

a = 20  # A new integer object is created, and 'a' now references the new object
# The original object (10) is eligible for garbage collection if no other references exist
```

[17] ✓ 0.0s

Q. What is lambda in Python? Why is it used.

Ans - In Python, a lambda function is an anonymous function that can be used for short-term purposes and simple tasks. Lambda functions are defined using the lambda keyword.

Example –

```
# Lambda function to add two numbers
add = lambda x, y: x + y

# Using the lambda function
result = add(5, 3)
print(result)
```

[19] ✓ 0.0s

```
8
```

Usage:

- Lambda functions are commonly used with functions like map(), filter(), and sorted() where a simple, temporary function is required.

R. Explain split() and join() functions in Python?

Ans - The split() and join() functions in Python are used for string manipulation:

1. split()

- **Purpose**: Splits a string into a list of substrings based on a specified delimiter.

Example –

```
# Lambda function to add two numbers
add = lambda x, y: x + y

# Using the lambda function
result = add(5, 3)
print(result)
```

[19] ✓ 0.0s

```
8
```

join()

- **Purpose**: Joins the elements of a list into a single string, with a specified delimiter between each element.

  Example-

```
fruits = ['apple', 'banana', 'cherry']
text = ", ".join(fruits)
print(text)
```

[23] ✓ 0.0s

```
apple, banana, cherry
```

S. What are iterators , iterable & generators in Python?

Ans - **1. Iterators**

- **Definition**: An iterator is an object that contains a countable number of values and can be iterated upon, meaning it can traverse through all the values.

  Example-

```
my_list = [1, 2, 3]
iterator = iter(my_list)
print(next(iterator))
print(next(iterator))
```

[24] ✓ 0.0s

```
1
2
```

2. Iterable

- **Definition**: An iterable is an object that can return an iterator. Examples include lists, tuples, strings, etc.

  Example –

```
my_list = [1, 2, 3]
for item in my_list:
    print(item)
```

25] ✓ 0.0s

```
1
2
3
```

Generators

- **Definition**: Generators are a simple way of creating iterators. They allow you to declare a function that behaves like an iterator, using yield instead of return.

  Example:

```
def my_generator():
    yield 1
    yield 2
    yield 3

gen = my_generator()
print(next(gen))
print(next(gen))
```

[27] ✓ 0.0s

```
1
2
```

T. What is the difference between xrange and range in Python?

Ans – In Python 2, xrange and range were used for generating sequences of numbers, but they worked differently:

1. **range**: Returns a list of numbers. If you have a large range, it will consume a lot of memory.

   Example -

```
r = range(10)
print(r)

[30]  ✓ 0.0s
...   range(0, 10)
```

2. **`xrange`**: Returns an xrange object, which generates numbers on demand and is more memory-efficient.

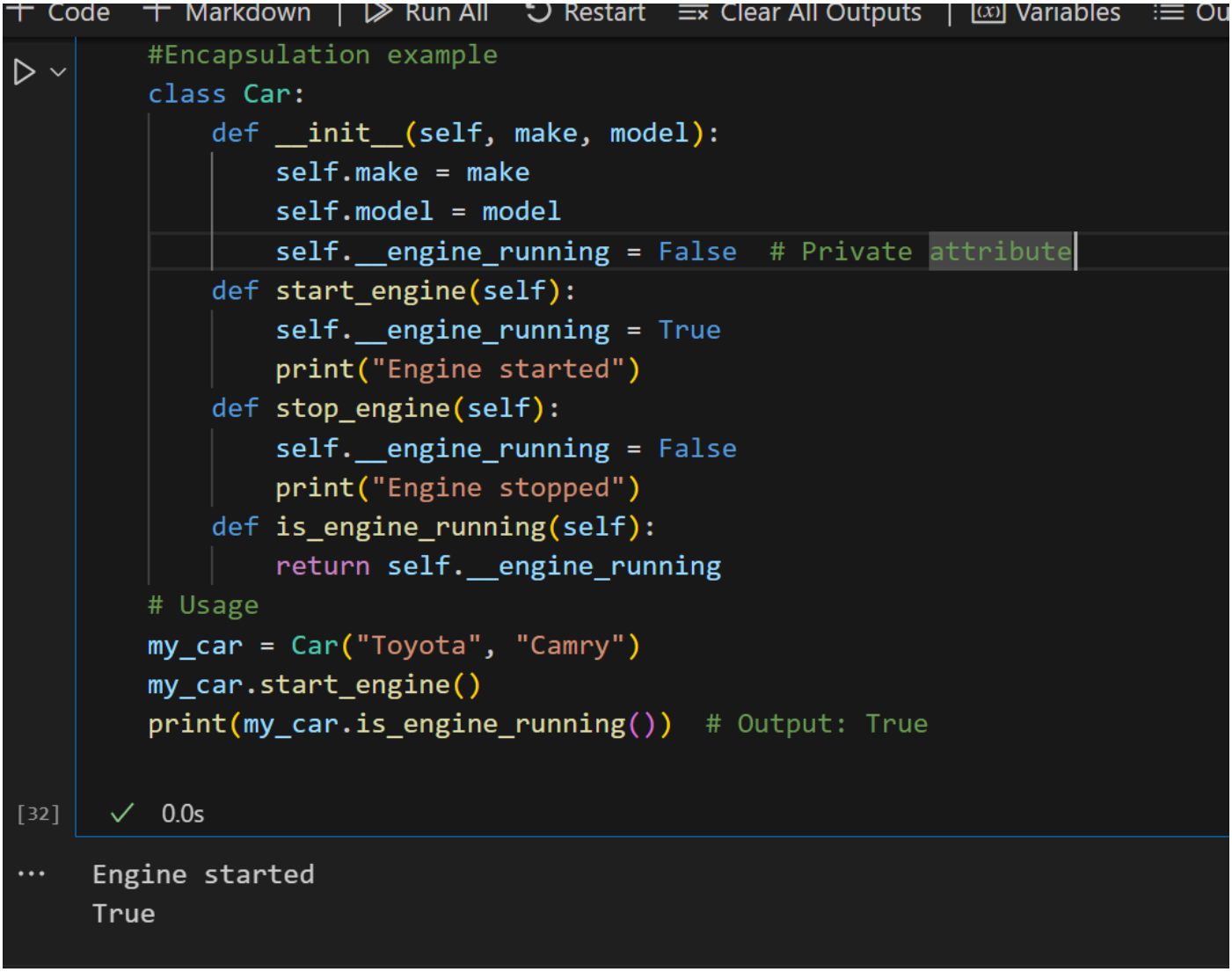
   `Example` -

```
x = xrange(10)
print(x)  # Output: xrange(0, 10)

[ ]
```

U. Pillars of Oops?

Ans - There are four pillars of Object-Oriented Programming (OOP)

1. **Encapsulation**: This principle involves bundling data (attributes) and methods (functions) that operate on the data into a single unit, typically a class. It also includes controlling access to the internal state of the object, often by using access modifiers to restrict visibility.
2. **Inheritance**: Inheritance allows one class (child or subclass) to inherit attributes and methods from another class (parent or superclass). This promotes code reuse and establishes a hierarchical relationship between classes.
3. **Polymorphism**: Polymorphism enables objects of different classes to be treated as objects of a common superclass. It allows methods to be used interchangeably, even if they are implemented differently in different classes, facilitating flexibility and extensibility.
4. **Abstraction**: Abstraction focuses on hiding complex implementation details and exposing only the necessary and relevant features of an object. It simplifies interaction with objects by providing a clear and simplified interface while concealing the underlying complexity.



V. How will you check if a class is a child of another class?

Ans - In Python, you can check if a class is a child (or subclass) of another class using the issubclass() function. This function takes two arguments: the potential subclass and the parent class, and returns True if the first class is a subclass of the second class, otherwise False.

Example –

```
class Parent:
    pass

class Child(Parent):
    pass

class AnotherClass:
    pass

# Check if Child is a subclass of Parent
print(issubclass(Child, Parent))  # Output: True

# Check if AnotherClass is a subclass of Parent
print(issubclass(AnotherClass, Parent))  # Output: False
```

[33] ✓ 0.0s

```
True
False
```

W. How does inheritance work in python? Explain all types of inheritance with an example.

Ans - Inheritance in Python allows a class (known as a subclass or child class) to inherit attributes and methods from another class (known as a superclass or parent class). This promotes code reusability and establishes a hierarchical relationship between classes.

Here are the main types of inheritance in Python, along with examples for each:

1. Single Inheritance

Single inheritance involves a subclass inheriting from a single superclass.

Example:

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

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

# Usage
dog = Dog()
dog.speak()  # Method from Animal class
dog.bark()   # Method from Dog class
```

[34] ✓ 0.0s

```
Animal speaks
Dog barks
```

2. Multiple Inheritance

Multiple inheritance involves a subclass inheriting from more than one superclass.

Example:

```python
class Flyer:
    def fly(self):
        print("Can fly")

class Swimmer:
    def swim(self):
        print("Can swim")

class Duck(Flyer, Swimmer):
    def quack(self):
        print("Duck quacks")

# Usage
duck = Duck()
duck.fly()  # Method from Flyer class
duck.swim() # Method from Swimmer class
duck.quack()# Method from Duck class
```

[35] ✓ 0.0s

```
Can fly
Can swim
Duck quacks
```

3. Multilevel Inheritance

Multilevel inheritance involves a subclass inheriting from another subclass, creating a chain of inheritance.

Example:

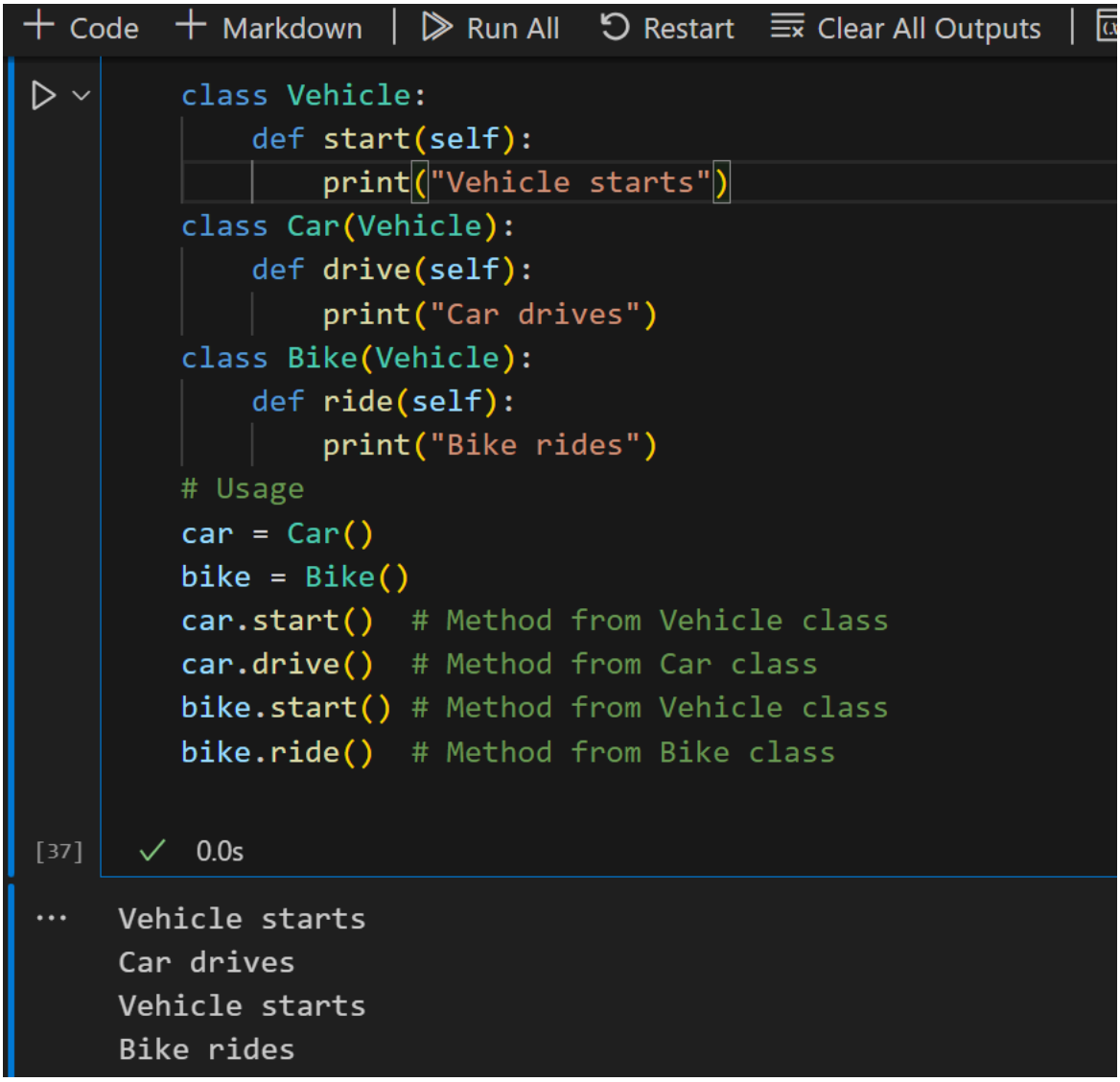
```
class Grandparent:
    def grandparent_method(self):
        print("Method from Grandparent")

class Parent(Grandparent):
    def parent_method(self):
        print("Method from Parent")

class Child(Parent):
    def child_method(self):
        print("Method from Child")

# Usage
child = Child()
child.grandparent_method()  # Method from Grandparent
child.parent_method()       # Method from Parent
child.child_method()        # Method from Child
```

```
[36]  ✓ 0.0s

...  Method from Grandparent
     Method from Parent
     Method from Child
```

4. Hierarchical Inheritance

Hierarchical inheritance involves multiple subclasses inheriting from a single superclass.

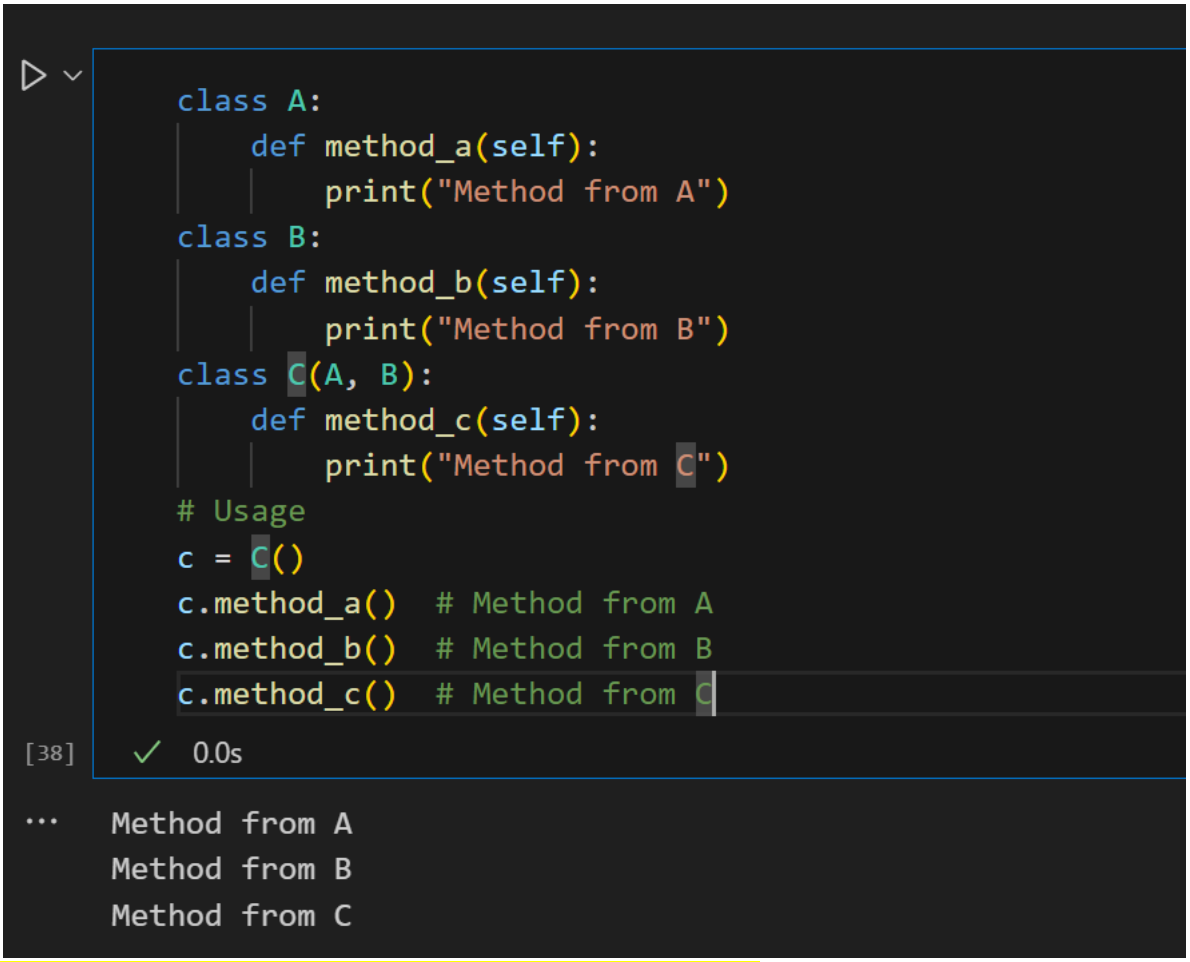
Example:



5. Multiple Inheritance with Common Base Class

This is a combination of multiple inheritance and hierarchical inheritance where multiple classes inherit from a common base class, and a subclass inherits from those multiple classes.

Example:



X. What is encapsulation? Explain it with an example

Ans – Encapsulation is one of the fundamental principles of Object-Oriented Programming (OOP). It involves bundling the data (attributes) and methods (functions) that operate on the data into a single unit, typically a class. Encapsulation also includes restricting direct access to some of the object's components to protect the integrity of the data.

Example –

```python
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited: {amount}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrew: {amount}")
        else:
            print("Invalid withdrawal amount or insufficient balance.")

    def get_balance(self):
        return self.__balance
```

```python
# Usage
account = BankAccount("123456", 1000)
print("Initial Balance:", account.get_balance())  # Output: Initial Balance: 1000

account.deposit(500)  # Output: Deposited: 500
print("Balance after deposit:", account.get_balance())  # Output: Balance after deposit: 1500

account.withdraw(200)  # Output: Withdrew: 200
print("Balance after withdrawal:", account.get_balance())  # Output: Balance after withdrawal: 1300

# Direct access to private attribute will fail
# print(account.__balance)  # AttributeError: 'BankAccount' object has no attribute '__balance'
```

[39] ✓ 0.0s

```
Initial Balance: 1000
Deposited: 500
Balance after deposit: 1500
Withdrew: 200
Balance after withdrawal: 1300
```

Y. What is polymorphism? Explain it with an example.

Ans - **Polymorphism** is a fundamental concept in Object-Oriented Programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. It enables methods to be used interchangeably, even if they are implemented differently in different classes. Polymorphism facilitates flexibility and the ability to extend code with minimal changes.

Output –

```
    animal_sound(cat)  # Output: Meow!
    animal_sound(duck) # Output: Quack!

[41]  ✓ 0.0s

...  Woof!
     Meow!
     Quack!
```

Question 1. 2. Which of the following identifier names are invalid and why?

a) Serial_no.

b) 1st_Room

c) Hundred$

d) Total_Marks

e) total-Marks

f) Total Marks

g) True

h) _Percentag

Ans - **Evaluating Each Identifier:**

a) **Serial_no.**

- **Invalid**: The period (.) is not allowed in identifiers.

b) **1st_Room**

- **Invalid**: Identifiers cannot start with a digit.

c) **Hundred$**

- **Invalid**: The dollar sign ($) is not allowed in identifiers.

d) **Total_Marks**

- **Valid**: This follows all the naming rules. It starts with a letter and uses underscores and letters only.

e) **total-Marks**

- **Invalid**: The hyphen (-) is not allowed in identifiers. Hyphens are interpreted as subtraction operators in Python.

f) **Total Marks**

- **Invalid**: The space is not allowed in identifiers. Identifiers cannot contain spaces.

g) **True**

- **Invalid**: True is a reserved keyword in Python. Using it as an identifier name would cause conflicts.

h) **_Percentag**

- **Valid**: This identifier starts with an underscore, contains only letters and underscores, and does not conflict with reserved keywords.

Summary:

- **Invalid Identifiers**: a, b, c, e, f, g
- **Valid Identifiers**: d, h

Question 1.3. name = ["Mohan", "dash", "karam", "chandra","gandhi","Bapu"] do the following operations in this list;

a) add an element "freedom_fighter" in this list at the 0th index

```
#a) add an element "freedom_fighter" in this list at the 0th index.
name = ["Mohan", "dash", "karam", "chandra","gandhi","Bapu"]
name.insert(0,'freedom_fighter')
print(name)
```

[6] ✓ 0.0s

```
['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu']
```

b) find the output of the following ,and explain how?

```
#find the output of the following ,and explain how?
name = ["freedomFighter","Bapuji","MOhan", "dash", "karam","chandra","gandhi"]
length1=len((name[-len(name)+1:-1:2]))
length2=len((name[-len(name)+1:-1]))
print(length1+length2)
```

[7] ✓ 0.0s

```
8
```

Explanation:

1. **List Construction:**
    - **The list name is ["freedomFighter", "Bapuji", "MOhan", "dash", "karam", "chandra", "gandhi"].**
2. **Slicing Operation for length1:**
    - **name[-len(name)+1:-1:2]:**
        - **This slice starts from the second element ("Bapuji") and ends at the second-to-last element ("chandra"), with a step of 2.**

- **Resulting slice: ["Bapuji", "dash", "chandra"].**
- **length1 = 3 (3 elements in this slice).**

3. **Slicing Operation for length2:**
   - **name[-len(name)+1:-1]:**
     - **This slice starts from the second element ("Bapuji") and ends at the second-to-last element ("chandra").**
     - **Resulting slice: ["Bapuji", "MOhan", "dash", "karam", "chandra"].**
     - **length2 = 5 (5 elements in this slice).**
4. **Final Calculation:**
   - **length1 + length2 = 3 + 5 = 8.**

c) **add two more elements in the name ["NetaJi","Bose"] at the end of the list**

```
name = ["freedomFighter","Bapuji","MOhan", "dash", "karam","chandra","gandhi"]
name.append('NetaJi'),name.append('Bose')
print(name)
```

[11] ✓ 0.0s

```
['freedomFighter', 'Bapuji', 'MOhan', 'dash', 'karam', 'chandra', 'gandhi', 'NetaJi', 'Bose']
```

d) what will be the value of temp:

```
temp=name[-1]
name[-1]=name[0]
name[0]=temp
print(name)
```

[12] ✓ 0.0s

```
['Bose', 'Bapuji', 'MOhan', 'dash', 'karam', 'chandra', 'gandhi', 'NetaJi', 'freedomFighter']
```

Question 1.4.Find the output of the following.

```
animal = ['Human','cat','mat','cat','rat','Human', 'Lion']
print(animal.count('Human'))
print(animal.index('rat'))
print(len(animal))
```

[13] ✓ 0.0s

```
2
4
7
```

Question 1.5.
tuple1=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22),[{"roll_no":1},{"name":"Navneet"}])

a)print(len(tuple1))

```
print(len(tuple1))
```
[15] ✓ 0.0s

```
8
```

b)print(tuple1[-1][-1]["name"])

```
print(tuple1[-1][-1]["name"])
```
[16] ✓ 0.0s

```
Navneet
```

c)fetch the value of roll_no from this tuple

```
# Fetch the value of 'roll_no'
roll_no = tuple1[7][0]["roll_no"]
print(roll_no)
```
[17] ✓ 0.0s

```
1
```

d)print(tuple1[-3][1])

```
print(tuple1[-3][1])
```
[18] ✓ 0.0s

```
ji
```

e)fetch the element "22" from this tuple.

```
element_22 = tuple1[6][2]
print(element_22)
```

[19] ✓ 0.0s

```
22
```

1.6. Write a program to display the appropriate message as per the color of signal(RED-Stop/Yellow-Stay/ Green-Go) at the road crossing.

```
# Function to display the message based on signal color
def traffic_signal(color):
    if color.lower() == "red":
        print("RED: Stop")
    elif color.lower() == "yellow":
        print("YELLOW: Stay")
    elif color.lower() == "green":
        print("GREEN: Go")
    else:
        print("Invalid color. Please enter Red, Yellow, or Green.")

# User input
signal_color = input("Enter the color of the traffic signal: ")
traffic_signal(signal_color)
```

[1] ✓ 3.0s

```
RED: Stop
```

1.7. Write a program to create a simple calculator performing only four basic operations(+,-,/,*) .

```python
# Function to perform basic arithmetic operations
def calculator(num1, num2, operation):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '*':
        return num1 * num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error! Division by zero."
    else:
        return "Invalid operation!"
# User input
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
operation = input("Enter the operation (+, -, *, /): ")
# Perform calculation and display result
result = calculator(num1, num2, operation)
print("The result is:", result)
```

[5] ✓ 15.3s

```
The result is: 15.0
```

+ Code

1.8. Write a program to find the larger of the three pre-specified numbers using ternary operators.

```python
# Pre-specified numbers
num1 = 10
num2 = 20
num3 = 15

# Finding the largest number using ternary operators
largest = num1 if (num1 > num2 and num1 > num3) else (num2 if num2 > num3 else num3)

# Display the largest number
print("The largest number is:", largest)
```

[6] ✓ 0.0s

```
The largest number is: 20
```

+ Code + Markdown

1.9. Write a program to find the factors of a whole number using a while loop.

```
# Input: Whole number
number = int(input("Enter a number: "))

# Start with 1
i = 1

# Print factors
print(f"Factors of {number} are:")
while i <= number:
    if number % i == 0:  # Check if 'i' is a factor
        print(i)
    i += 1  # Move to the next number
```

[8] ✓ 6.1s

```
Factors of 8 are:
1
2
4
8
```

1.10. Write a program to find the sum of all the positive numbers entered by the user. As soon as the user enters a negative number, stop taking in any further input from the user and display the sum .

```
# Initialize the sum
total_sum = 0

while True:
    # Ask the user for input
    num = int(input("Enter a positive number (negative number to stop): "))

    # Check if the number is positive
    if num > 0:
        total_sum += num  # Add to the sum if positive
    else:
        break  # Stop if a negative number is entered

# Display the result
print("The sum of all positive numbers is:", total_sum)
```

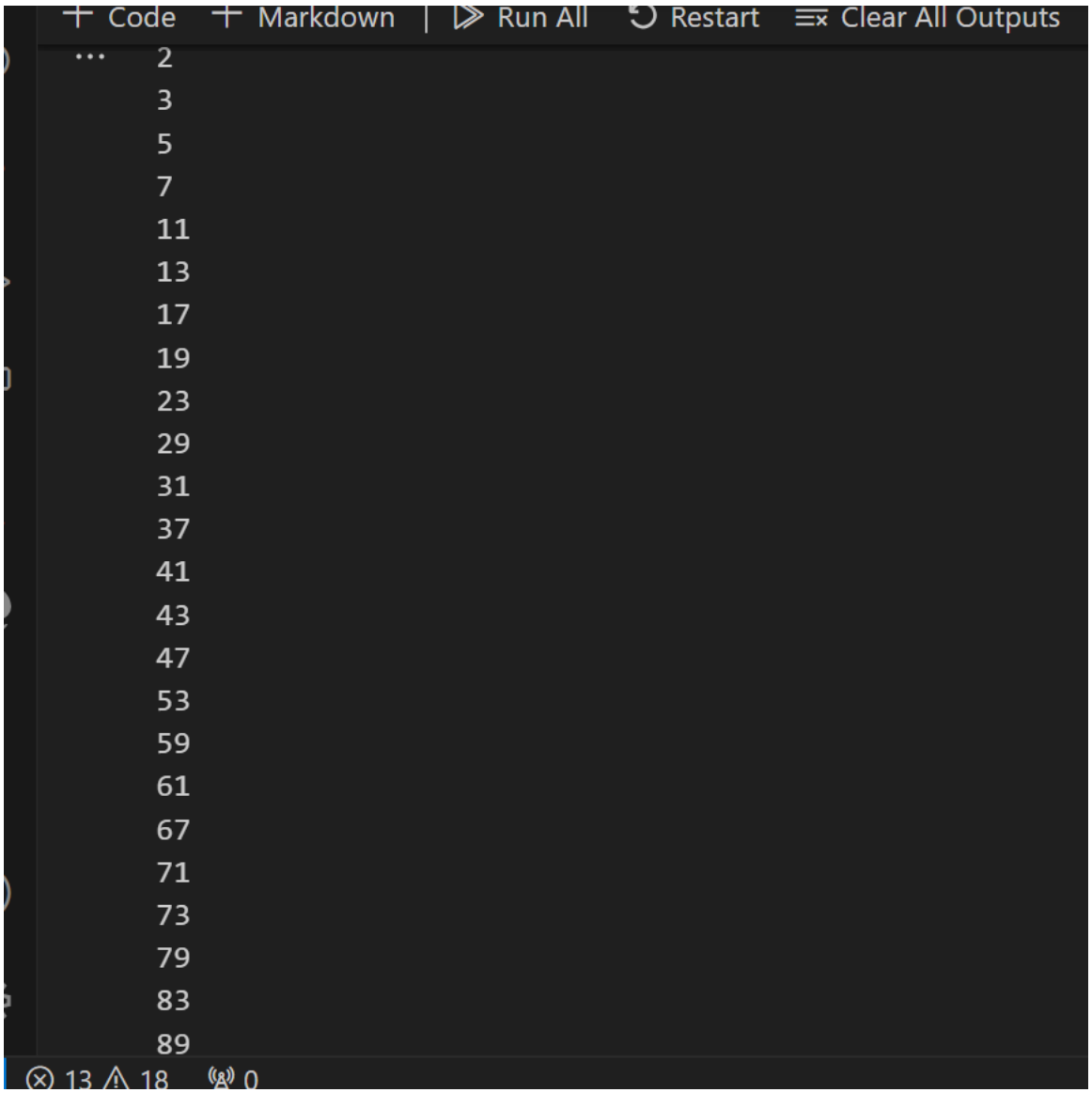
[11]

1.11. Write a program to find prime numbers between 2 to 100 using nested for loops.

```
for num in range(2, 101):  # Iterate through numbers from 2 to 100
    is_prime = True  # Assume number is prime
    for i in range(2, num):  # Check divisibility by numbers less than itself
        if num % i == 0:  # If divisible, it's not prime
            is_prime = False
            break
    if is_prime:  # If still prime, print the number
        print(num)
```

[1] ✓ 0.0s

Output –



1.12. Write the programs for the following Accept the marks of the student in five major subjects and display the same Calculate the sum of the marks of all subjects.Divide the total marks by number of subjects (i.e. 5), calculate percentage = total marks/5 and display the percentage Find the grade of the student as per the following criteria . Hint: Use Match & case for this.

```python
def main():
    # Accept marks for five subjects
    marks = []
    subjects = ["Math", "Science", "English", "History", "Geography"]

    print("Enter the marks for the following subjects:")
    for subject in subjects:
        while True:
            try:
                mark = float(input(f"{subject}: "))
                if 0 <= mark <= 100:
                    marks.append(mark)
                    break
                else:
                    print("Please enter a mark between 0 and 100.")
            except ValueError:
                print("Invalid input. Please enter a number.")

    # Calculate total marks and percentage
    total_marks = sum(marks)
    number_of_subjects = len(marks)
    percentage = total_marks / number_of_subjects
```

```python
    # Display the marks, total, and percentage
    print("\nMarks entered:")
    for i, subject in enumerate(subjects):
        print(f"{subject}: {marks[i]}")

    print(f"\nTotal Marks: {total_marks}")
    print(f"Percentage: {percentage:.2f}%")

    # Determine the grade using match-case
    match percentage:
        case p if p >= 90:
            grade = 'A+'
        case p if p >= 80:
            grade = 'A'
        case p if p >= 70:
            grade = 'B+'
        case p if p >= 60:
            grade = 'B'
        case p if p >= 50:
            grade = 'C'
        case p if p >= 40:
            grade = 'D'
        case _:
```

```
            case _:
                grade = 'F'

        print(f"Grade: {grade}")

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

[2] ✓ 33.8s

```
Enter the marks for the following subjects:
Invalid input. Please enter a number.

Marks entered:
Math: 80.0
Science: 70.0
English: 85.0
History: 90.0
Geography: 70.0

Total Marks: 395.0
Percentage: 79.00%
Grade: B+
```

1.13. Write a program for VIBGYOR Spectrum based on their Wavelength using. Wavelength Range:

```
def get_color_by_wavelength(wavelength):
    # Determine the color based on the wavelength
    match wavelength:
        case w if 380 <= w < 450:
            color = 'Violet'
        case w if 450 <= w < 495:
            color = 'Blue'
        case w if 495 <= w < 570:
            color = 'Green'
        case w if 570 <= w < 590:
            color = 'Yellow'
        case w if 590 <= w < 620:
            color = 'Orange'
        case w if 620 <= w <= 750:
            color = 'Red'
        case _:
            color = 'Wavelength out of range for VIBGYOR spectrum'

    return color

def main():
    try:
```

```
def main():
    try:
        # Get user input for wavelength
        wavelength = float(input("Enter the wavelength (in nm): "))

        if wavelength < 380 or wavelength > 750:
            print("Wavelength out of range. Please enter a value between 380 and 750 nm.")
        else:
            color = get_color_by_wavelength(wavelength)
            print(f"The color corresponding to the wavelength {wavelength} nm is {color}.")
    except ValueError:
        print("Invalid input. Please enter a numeric value for the wavelength.")

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

✓ 3.3s

```
The color corresponding to the wavelength 400.0 nm is Violet.
```

1.14.Consider the gravitational interactions between the Earth, Moon, and Sun in our solar system. Given: mass_earth = 5.972e24 # Mass of Earth in kilograms mass_moon = 7.34767309e22 # Mass of Moon in kilograms mass_sun = 1.989e30 # Mass of Sun in kilograms distance_earth_sun = 1.496e11 # Average distance between Earth and Sun in meters distance_moon_earth = 3.844e8 # Average distance between Moon and Earth in meters.

Tasks

- Calculate the gravitational force between the Earth and the Sun
- Calculate the gravitational force between the Moon and the Earth
- Compare the calculated forces to determine which gravitational force is stronge
- Explain which celestial body (Earth or Moon) is more attracted to the other based on the comparison.

Ans –

```python
def calculate_gravitational_force(m1, m2, distance):
    G = 6.67430e-11  # Gravitational constant in m^3 kg^-1 s^-2
    force = G * (m1 * m2) / (distance ** 2)
    return force

def main():
    # Masses in kilograms
    mass_earth = 5.972e24
    mass_moon = 7.34767309e22
    mass_sun = 1.989e30

    # Distances in meters
    distance_earth_sun = 1.496e11
    distance_moon_earth = 3.844e8

    # Calculate gravitational forces
    force_earth_sun = calculate_gravitational_force(mass_earth, mass_sun, distance_earth_sun)
    force_moon_earth = calculate_gravitational_force(mass_moon, mass_earth, distance_moon_earth)

    # Display results
    print(f"Gravitational force between Earth and Sun: {force_earth_sun:.2e} N")
    print(f"Gravitational force between Moon and Earth: {force_moon_earth:.2e} N")
```

```python
    # Compare forces
    if force_earth_sun > force_moon_earth:
        print("The gravitational force between Earth and the Sun is stronger than the force between Moon and Earth.")
        print("Thus, the Earth is more attracted to the Sun than the Moon is to the Earth.")
    elif force_earth_sun < force_moon_earth:
        print("The gravitational force between Moon and Earth is stronger than the force between Earth and Sun.")
        print("Thus, the Moon is more attracted to the Earth than the Earth is to the Sun.")
    else:
        print("The gravitational forces between Earth-Sun and Moon-Earth are equal.")

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

0.0s

```
Gravitational force between Earth and Sun: 3.54e+22 N
Gravitational force between Moon and Earth: 1.98e+20 N
The gravitational force between Earth and the Sun is stronger than the force between Moon and Earth.
Thus, the Earth is more attracted to the Sun than the Moon is to the Earth.
```

2. Design and implement a Python program for managing student information using object-oriented principles. Create a class called `Student` with encapsulated attributes for name, age, and roll number. Implement getter and setter methods for these attributes. Additionally, provide methods to display student information and update student details.

Tasks

- Define the `Student` class with encapsulated attributes
- Implement getter and setter methods for the attributes
- Write methods to display student information and update details
- Create instances of the `Student` class and test the implemented functionality

```python
class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number

    # Getter for name
    def get_name(self):
        return self.__name

    # Setter for name
    def set_name(self, name):
        self.__name = name

    # Getter for age
    def get_age(self):
        return self.__age

    # Setter for age
    def set_age(self, age):
        if age > 0:
            self.__age = age
        else:
            print("Age must be a positive number.")

    # Getter for roll number
    def get_roll_number(self):
        return self.__roll_number

    # Setter for roll number
    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number

    # Method to display student information
    def display_info(self):
        print(f"Name: {self.__name}")
        print(f"Age: {self.__age}")
        print(f"Roll Number: {self.__roll_number}")

    # Method to update student details
    def update_details(self, name=None, age=None, roll_number=None):
        if name is not None:
            self.set_name(name)
        if age is not None:
            self.set_age(age)
```

```
            if age is not None:
                self.set_age(age)
            if roll_number is not None:
                self.set_roll_number(roll_number)

    # Test the functionality
    def main():
        # Create instances of Student
        student1 = Student("John Doe", 20, "S12345")
        student2 = Student("Jane Smith", 22, "S54321")

        # Display initial information
        print("Initial student information:")
        student1.display_info()
        print()
        student2.display_info()
        print()

        # Update student details
        student1.update_details(name="Johnathan Doe", age=21)
        student2.update_details(roll_number="S99999")

        # Display updated information
        print("Updated student information:")
```

```
        student2.display_info()
        print()

        # Update student details
        student1.update_details(name="Johnathan Doe", age=21)
        student2.update_details(roll_number="S99999")

        # Display updated information
        print("Updated student information:")
        student1.display_info()
        print()
        student2.display_info()

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

Output –

```
Initial student information:
Name: John Doe
Age: 20
Roll Number: S12345

Name: Jane Smith
Age: 22
Roll Number: S54321

Updated student information:
Name: Johnathan Doe
Age: 21
Roll Number: S12345

Name: Jane Smith
Age: 22
Roll Number: S99999
```

3.Develop a Python program for managing library resources efficiently. Design a class named `LibraryBook` with attributes like book name, author, and availability status. Implement methods for borrowing and returning books while ensuring proper encapsulation of attributes.

Tasks

1. Create the `LibraryBook` class with encapsulated attributes

2. Implement methods for borrowing and returning books

3. Ensure proper encapsulation to protect book details

4. Test the borrowing and returning functionality with sample data

Ans –

Code –

```
class LibraryBook:
    def __init__(self, book_name, author):
        self.__book_name = book_name
        self.__author = author
        self.__available = True
```

```
    # Getter for book name
    def get_book_name(self):
        return self.__book_name

    # Setter for book name
    def set_book_name(self, book_name):
        self.__book_name = book_name

    # Getter for author
    def get_author(self):
        return self.__author

    # Setter for author
    def set_author(self, author):
        self.__author = author

    # Getter for availability status
    def is_available(self):
        return self.__available

    # Borrow method
    def borrow_book(self):
        if self.__available:
            self.__available = False
            print(f"You have successfully borrowed '{self.__book_name}' by {self.__author}.")
        else:
            print(f"Sorry, '{self.__book_name}' is currently not available.")
```

```
    # Return method
    def return_book(self):
        if not self.__available:
            self.__available = True
            print(f"Thank you for returning '{self.__book_name}' by {self.__author}.")
        else:
            print(f"'{self.__book_name}' was not borrowed, so it cannot be returned.")

    # Display book details
    def display_info(self):
        availability = "Available" if self.__available else "Not Available"
        print(f"Book Name: {self.__book_name}")
        print(f"Author: {self.__author}")
        print(f"Availability: {availability}")

# Test the functionality
def main():
    # Create instances of LibraryBook
    book1 = LibraryBook("1984", "George Orwell")
    book2 = LibraryBook("To Kill a Mockingbird", "Harper Lee")

    # Display initial information
    print("Initial book information:")
    book1.display_info()
    print()
    book2.display_info()
    print()

    # Borrowing books
```

```
    book1.borrow_book()
    book2.borrow_book()
    print()

    # Attempt to borrow again
    book1.borrow_book()
    print()

    # Returning books
    book1.return_book()
    book2.return_book()
    print()

    # Display updated information
    print("Updated book information:")
    book1.display_info()
    print()
    book2.display_info()

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

output –

```
    main()

[7]  ✓ 0.0s

...  Initial book information:
     Book Name: 1984
     Author: George Orwell
     Availability: Available

     Book Name: To Kill a Mockingbird
     Author: Harper Lee
     Availability: Available

     You have successfully borrowed '1984' by George Orwell.
     You have successfully borrowed 'To Kill a Mockingbird' by Harper Lee.

     Sorry, '1984' is currently not available.

     Thank you for returning '1984' by George Orwell.
     Thank you for returning 'To Kill a Mockingbird' by Harper Lee.

     Updated book information:
     Book Name: 1984
     Author: George Orwell
     Availability: Available

     Book Name: To Kill a Mockingbird
     Author: Harper Lee
     Availability: Available
```

4.Create a simple banking system using object-oriented concepts in Python. Design classes representing different types of bank accounts such as savings and checking. Implement methods for deposit, withdraw, and balance inquiry. Utilize inheritance to manage different account types efficiently.

Tasks

1. Define base class(es) for bank accounts with common attributes and methods

2. Implement subclasses for specific account types (e.g., SavingsAccount, CheckingAccount)
3. Provide methods for deposit, withdraw, and balance inquiry in each subclass

4. Test the banking system by creating instances of different account types and performing transactions.

Ans –

Code –

```
class BankAccount:
    def __init__(self, account_number, owner_name, balance=0.0):
        self.__account_number = account_number
        self.__owner_name = owner_name
        self.__balance = balance

    # Getter for account number
    def get_account_number(self):
        return self.__account_number

    # Getter for owner name
    def get_owner_name(self):
        return self.__owner_name

    # Getter for balance
    def get_balance(self):
        return self.__balance

    # Deposit method
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited ${amount:.2f}. New balance: ${self.__balance:.2f}")
        else:
            print("Deposit amount must be positive.")

    # Withdraw method
```

```
    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.__balance:
                self.__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance: ${self.__balance:.2f}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    # Display account details
    def display_account_info(self):
        print(f"Account Number: {self.__account_number}")
        print(f"Owner Name: {self.__owner_name}")
        print(f"Balance: ${self.__balance:.2f}")


class SavingsAccount(BankAccount):
    def __init__(self, account_number, owner_name, balance=0.0, interest_rate=0.02):
        super().__init__(account_number, owner_name, balance)
        self.__interest_rate = interest_rate

    # Apply interest to the balance
    def apply_interest(self):
        interest = self.get_balance() * self.__interest_rate
        self.deposit(interest)
        print(f"Applied interest: ${interest:.2f}. New balance: ${self.get_balance():.2f}")

    # Display account details including interest rate
```

```
    def display_account_info(self):
        super().display_account_info()
        print(f"Interest Rate: {self.__interest_rate:.2%}")


class CheckingAccount(BankAccount):
    def __init__(self, account_number, owner_name, balance=0.0, overdraft_limit=500.0):
        super().__init__(account_number, owner_name, balance)
        self.__overdraft_limit = overdraft_limit

    # Withdraw method with overdraft facility
    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.get_balance() + self.__overdraft_limit:
                self._BankAccount__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance: ${self.get_balance():.2f}")
            else:
                print("Exceeded overdraft limit.")
        else:
            print("Withdrawal amount must be positive.")

    # Display account details including overdraft limit
    def display_account_info(self):
        super().display_account_info()
        print(f"Overdraft Limit: ${self.__overdraft_limit:.2f}")


# Test the banking system
def main():
```

```
# Create instances of SavingsAccount and CheckingAccount
savings_account = SavingsAccount("SA12345", "Alice", 1000.0, 0.03)
checking_account = CheckingAccount("CA67890", "Bob", 500.0, 200.0)

# Display initial information
print("Initial Savings Account Information:")
savings_account.display_account_info()
print()
print("Initial Checking Account Information:")
checking_account.display_account_info()
print()

# Perform transactions
savings_account.deposit(200)
savings_account.withdraw(50)
savings_account.apply_interest()
print()

checking_account.deposit(100)
checking_account.withdraw(600)
checking_account.withdraw(50)
print()

# Display updated information
print("Updated Savings Account Information:")
savings_account.display_account_info()
print()
print("Updated Checking Account Information:")
checking_account.display_account_info()
```

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

Output –

```
Initial Savings Account Information:
Account Number: SA12345
Owner Name: Alice
Balance: $1000.00
Interest Rate: 3.00%

Initial Checking Account Information:
Account Number: CA67890
Owner Name: Bob
Balance: $500.00
Overdraft Limit: $200.00

Deposited $200.00. New balance: $1200.00
Withdrew $50.00. New balance: $1150.00
Deposited $34.50. New balance: $1184.50
Applied interest: $34.50. New balance: $1184.50

Deposited $100.00. New balance: $600.00
Withdrew $600.00. New balance: $0.00
Withdrew $50.00. New balance: $-50.00
```

```
Deposited $200.00. New balance: $1200.00
Withdrew $50.00. New balance: $1150.00
Deposited $34.50. New balance: $1184.50
Applied interest: $34.50. New balance: $1184.50

Deposited $100.00. New balance: $600.00
Withdrew $600.00. New balance: $0.00
Withdrew $50.00. New balance: $-50.00

Updated Savings Account Information:
Account Number: SA12345
Owner Name: Alice
Balance: $1184.50
...
Account Number: CA67890
Owner Name: Bob
Balance: $-50.00
Overdraft Limit: $200.00
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

5.Write a Python program that models different animals and their sounds. Design a base class called `Animal` with a method `make_sound()`. Create subclasses like `Dog` and `Cat` that override the `make_sound()` method to produce appropriate sounds.

Tasks

1. Define the `Animal` class with a method `make_sound()`

2. Create subclasses `Dog` and `Cat` that override the `make_sound()` method

3. Implement the sound generation logic for each subclass \

4. Test the program by creating instances of `Dog` and `Cat` and calling the `make_sound()` method

Ans –

Output –

```python
class Animal:
    def make_sound(self):
        raise NotImplementedError("Subclasses must implement this method.")

class Dog(Animal):
    def make_sound(self):
        return "Woof! Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow! Meow!"

# Test the program
def main():
    # Create instances of Dog and Cat
    dog = Dog()
    cat = Cat()

    # Call make_sound method and print results
    print("Dog sound:", dog.make_sound())
    print("Cat sound:", cat.make_sound())

if __name__ == "__main__":
```

```
        cat = Cat()

        # Call make_sound method and print results
        print("Dog sound:", dog.make_sound())
        print("Cat sound:", cat.make_sound())

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

[9] ✓ 0.0s

```
Dog sound: Woof! Woof!
Cat sound: Meow! Meow!
```

6.Write a code for Restaurant Management System Using OOPS

- Create a MenuItem class that has attributes such as name, description, price, and category
- Implement methods to add a new menu item, update menu item information, and remove a menu item from the menu
- Use encapsulation to hide the menu item's unique identification number
- Inherit from the MenuItem class to create a FoodItem class and a BeverageItem class, each with their own specific attributes and methods.

```
class MenuItem:
    _id_counter = 1

    def __init__(self, name, description, price, category):
        self.__item_id = MenuItem._id_counter
        MenuItem._id_counter += 1
        self.__name = name
        self.__description = description
        self.__price = price
        self.__category = category

    # Getter for item ID
    def get_item_id(self):
        return self.__item_id

    # Getter for name
    def get_name(self):
        return self.__name

    # Setter for name
    def set_name(self, name):
        self.__name = name

    # Getter for description
```

```
    def get_description(self):
        return self.__description

    # Setter for description
    def set_description(self, description):
        self.__description = description

    # Getter for price
    def get_price(self):
        return self.__price

    # Setter for price
    def set_price(self, price):
        if price > 0:
            self.__price = price
        else:
            print("Price must be positive.")

    # Getter for category
    def get_category(self):
        return self.__category

    # Setter for category
    def set_category(self, category):
        self.__category = category

    # Display item information
    def display_info(self):
        print(f"ID: {self.__item_id}")
        print(f"Name: {self.__name}")
        print(f"Description: {self.__description}")
        print(f"Price: ${self.__price:.2f}")
        print(f"Category: {self.__category}")

    # Update item details
    def update_info(self, name=None, description=None, price=None, category=None):
        if name is not None:
            self.set_name(name)
        if description is not None:
            self.set_description(description)
        if price is not None:
            self.set_price(price)
        if category is not None:
            self.set_category(category)

    # Remove item (effectively, this will just "disable" the item)
    def remove_item(self):
        print(f"Item '{self.__name}' removed from the menu.")

class FoodItem(MenuItem):
    def __init__(self, name, description, price, category, cuisine_type):
        super().__init__(name, description, price, category)
        self.__cuisine_type = cuisine_type

```

```
    # Getter for cuisine type
    def get_cuisine_type(self):
        return self.__cuisine_type

    # Setter for cuisine type
    def set_cuisine_type(self, cuisine_type):
        self.__cuisine_type = cuisine_type

    # Display food item information
    def display_info(self):
        super().display_info()
        print(f"Cuisine Type: {self.__cuisine_type}")

    # Update food item details
    def update_info(self, name=None, description=None, price=None, category=None,
cuisine_type=None):
        super().update_info(name, description, price, category)
        if cuisine_type is not None:
            self.set_cuisine_type(cuisine_type)

class BeverageItem(MenuItem):
    def __init__(self, name, description, price, category, volume):
        super().__init__(name, description, price, category)
        self.__volume = volume

    # Getter for volume
    def get_volume(self):
        return self.__volume

    # Setter for volume
    def set_volume(self, volume):
        self.__volume = volume

    # Display beverage item information
    def display_info(self):
        super().display_info()
        print(f"Volume: {self.__volume} ml")

    # Update beverage item details
    def update_info(self, name=None, description=None, price=None, category=None,
volume=None):
        super().update_info(name, description, price, category)
        if volume is not None:
            self.set_volume(volume)

# Test the Restaurant Management System
def main():
    # Create instances of FoodItem and BeverageItem
    pizza = FoodItem("Margherita Pizza", "Classic pizza with cheese and tomatoes",
12.99, "Main Course", "Italian")
    coffee = BeverageItem("Espresso", "Strong coffee with rich flavor", 3.99,
"Beverage", 30)

    # Display initial information
```

```
    print("Initial Menu Items:")
    pizza.display_info()
    print()
    coffee.display_info()
    print()

    # Update item details
    pizza.update_info(name="Veggie Pizza", price=14.99, cuisine_type="Italian")
    coffee.update_info(volume=40)

    # Display updated information
    print("Updated Menu Items:")
    pizza.display_info()
    print()
    coffee.display_info()
    print()

    # Remove items (in practice, this could be more complex, but here we just print a message)
    pizza.remove_item()
    coffee.remove_item()

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

output –

```
Initial Menu Items:
ID: 1
Name: Margherita Pizza
Description: Classic pizza with cheese and tomatoes
Price: $12.99
Category: Main Course
Cuisine Type: Italian

ID: 2
Name: Espresso
Description: Strong coffee with rich flavor
Price: $3.99
Category: Beverage
Volume: 30 ml

Updated Menu Items:
ID: 1
Name: Veggie Pizza
Description: Classic pizza with cheese and tomatoes
Price: $14.99
Category: Main Course
Cuisine Type: Italian
```

```
Description: Classic pizza with cheese and tomatoes
Price: $14.99
Category: Main Course
Cuisine Type: Italian

ID: 2
Name: Espresso
...
Volume: 40 ml

Item 'Veggie Pizza' removed from the menu.
Item 'Espresso' removed from the menu.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

7.Write a code for Hotel Management System using OOPS

- Create a Room class that has attributes such as room number, room type, rate, and availability (private)
- Implement methods to book a room, check in a guest, and check out a guest
- Use encapsulation to hide the room's unique identification number
- Inherit from the Room class to create a SuiteRoom class and a StandardRoom class, each with their own specific attributes and methods.

  Ans –

  Code –

```python
class Room:
    _id_counter = 1

    def __init__(self, room_type, rate):
        self.__room_id = Room._id_counter
        Room._id_counter += 1
        self.__room_type = room_type
        self.__rate = rate
        self.__availability = True
        self.__guest_name = None

    # Getter for room ID (private)
    def get_room_id(self):
        return self.__room_id

    # Getter for room type
    def get_room_type(self):
        return self.__room_type

    # Getter for rate
    def get_rate(self):
        return self.__rate

    # Getter for availability
    def is_available(self):
        return self.__availability
```

```
    # Book the room
    def book_room(self, guest_name):
        if self.__availability:
            self.__availability = False
            self.__guest_name = guest_name
            print(f"Room {self.__room_id} booked by {guest_name}.")
        else:
            print(f"Room {self.__room_id} is already booked.")

    # Check in a guest
    def check_in(self):
        if self.__availability:
            print(f"Room {self.__room_id} is available for check-in.")
        else:
            print(f"Room {self.__room_id} is already occupied.")

    # Check out a guest
    def check_out(self):
        if not self.__availability:
            self.__availability = True
            guest_name = self.__guest_name
            self.__guest_name = None
            print(f"Room {self.__room_id} checked out by {guest_name}.")
        else:
            print(f"Room {self.__room_id} is already available.")

    # Display room information
    def display_info(self):
        print(f"Room ID: {self.__room_id}")
        print(f"Room Type: {self.__room_type}")
        print(f"Rate: ${self.__rate:.2f}")
        print(f"Availability: {'Available' if self.__availability else 'Occupied'}")
        if not self.__availability:
            print(f"Guest Name: {self.__guest_name}")

class SuiteRoom(Room):
    def __init__(self, rate, extra_services):
        super().__init__("Suite", rate)
        self.__extra_services = extra_services

    # Getter for extra services
    def get_extra_services(self):
        return self.__extra_services

    # Setter for extra services
    def set_extra_services(self, extra_services):
        self.__extra_services = extra_services

    # Display suite room information
    def display_info(self):
        super().display_info()
        print(f"Extra Services: {', '.join(self.__extra_services)}")

class StandardRoom(Room):
```

```
    def __init__(self, rate, amenities):
        super().__init__("Standard", rate)
        self.__amenities = amenities

    # Getter for amenities
    def get_amenities(self):
        return self.__amenities

    # Setter for amenities
    def set_amenities(self, amenities):
        self.__amenities = amenities

    # Display standard room information
    def display_info(self):
        super().display_info()
        print(f"Amenities: {', '.join(self.__amenities)}")

# Test the Hotel Management System
def main():
    # Create instances of SuiteRoom and StandardRoom
    suite = SuiteRoom(rate=150.00, extra_services=["Spa", "Room Service"])
    standard = StandardRoom(rate=80.00, amenities=["Wi-Fi", "TV"])

    # Display initial information
    print("Initial Room Information:")
    suite.display_info()
    print()
    standard.display_info()
    print()

    # Book a room and check in
    suite.book_room("Alice")
    suite.check_in()
    print()

    # Display updated information
    suite.display_info()
    print()

    # Check out a guest and display information
    suite.check_out()
    suite.display_info()
    print()

    # Book and check in a standard room
    standard.book_room("Bob")
    standard.check_in()
    print()

    # Display updated information
    standard.display_info()
    print()

    # Check out a guest and display information
```

```
    standard.check_out()
    standard.display_info()

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

Output –

```
Initial Room Information:
Room ID: 1
Room Type: Suite
Rate: $150.00
Availability: Available
Extra Services: Spa, Room Service

Room ID: 2
Room Type: Standard
Rate: $80.00
Availability: Available
Amenities: Wi-Fi, TV

Room 1 booked by Alice.
Room 1 is already occupied.

Room ID: 1
Room Type: Suite
Rate: $150.00
Availability: Occupied
Guest Name: Alice
```

```
Room ID: 1
Room Type: Suite
Rate: $150.00
Availability: Occupied
Guest Name: Alice
Extra Services: Spa, Room Service

Room 1 checked out by Alice.
Room ID: 1
...
Room Type: Standard
Rate: $80.00
Availability: Available
Amenities: Wi-Fi, TV
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

8.Write a code for Fitness Club Management System using OOPS

- Create a Member class that has attributes such as name, age, membership type, and membership status (private)
- Implement methods to register a new member, renew a membership, and cancel a membership
- Use encapsulation to hide the member's unique identification number
- Inherit from the Member class to create a FamilyMember class and an IndividualMember class, each with their own specific attributes and methods

Ans -

Code –

```python
class Member:
    _id_counter = 1

    def __init__(self, name, age, membership_type):
        self.__member_id = Member._id_counter
        Member._id_counter += 1
        self.__name = name
        self.__age = age
        self.__membership_type = membership_type
        self.__membership_status = "Active"

    # Getter for member ID (private)
    def get_member_id(self):
        return self.__member_id

    # Getter for name
    def get_name(self):
        return self.__name
```

```python
    # Setter for name
    def set_name(self, name):
        self.__name = name

    # Getter for age
    def get_age(self):
        return self.__age

    # Setter for age
    def set_age(self, age):
        if age > 0:
            self.__age = age
        else:
            print("Age must be positive.")

    # Getter for membership type
    def get_membership_type(self):
        return self.__membership_type

    # Setter for membership type
    def set_membership_type(self, membership_type):
        self.__membership_type = membership_type

    # Getter for membership status
    def get_membership_status(self):
        return self.__membership_status

    # Register a new member
    def register_member(self):
        print(f"Member {self.__name} registered with ID {self.__member_id}.")

    # Renew membership
    def renew_membership(self):
        if self.__membership_status == "Inactive":
            self.__membership_status = "Active"
            print(f"Membership for {self.__name} renewed.")
        else:
            print(f"Membership for {self.__name} is already active.")

    # Cancel membership
    def cancel_membership(self):
        if self.__membership_status == "Active":
            self.__membership_status = "Inactive"
            print(f"Membership for {self.__name} canceled.")
        else:
            print(f"Membership for {self.__name} is already inactive.")

    # Display member information
    def display_info(self):
        print(f"Member ID: {self.__member_id}")
        print(f"Name: {self.__name}")
        print(f"Age: {self.__age}")
        print(f"Membership Type: {self.__membership_type}")
        print(f"Membership Status: {self.__membership_status}")
```

```
class FamilyMember(Member):
    def __init__(self, name, age, membership_type, family_size):
        super().__init__(name, age, membership_type)
        self.__family_size = family_size

    # Getter for family size
    def get_family_size(self):
        return self.__family_size

    # Setter for family size
    def set_family_size(self, family_size):
        if family_size > 0:
            self.__family_size = family_size
        else:
            print("Family size must be positive.")

    # Display family member information
    def display_info(self):
        super().display_info()
        print(f"Family Size: {self.__family_size}")

class IndividualMember(Member):
    def __init__(self, name, age, membership_type, personal_trainer):
        super().__init__(name, age, membership_type)
        self.__personal_trainer = personal_trainer

    # Getter for personal trainer
    def get_personal_trainer(self):
        return self.__personal_trainer

    # Setter for personal trainer
    def set_personal_trainer(self, personal_trainer):
        self.__personal_trainer = personal_trainer

    # Display individual member information
    def display_info(self):
        super().display_info()
        print(f"Personal Trainer: {self.__personal_trainer}")

# Test the Fitness Club Management System
def main():
    # Create instances of FamilyMember and IndividualMember
    family_member = FamilyMember(name="Smith Family", age=40, membership_type="Family",
family_size=4)
    individual_member = IndividualMember(name="John Doe", age=30,
membership_type="Individual", personal_trainer="Mike")

    # Register new members
    family_member.register_member()
    individual_member.register_member()
    print()

    # Display initial information
```

```
    print("Initial Member Information:")
    family_member.display_info()
    print()
    individual_member.display_info()
    print()

    # Renew and cancel memberships
    family_member.cancel_membership()
    family_member.renew_membership()
    print()
    individual_member.cancel_membership()
    individual_member.renew_membership()
    print()

    # Display updated information
    family_member.display_info()
    print()
    individual_member.display_info()

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

Output –

```
Member Smith Family registered with ID 1.
Member John Doe registered with ID 2.

Initial Member Information:
Member ID: 1
Name: Smith Family
Age: 40
Membership Type: Family
Membership Status: Active
Family Size: 4

Member ID: 2
Name: John Doe
Age: 30
Membership Type: Individual
Membership Status: Active
Personal Trainer: Mike

Membership for Smith Family canceled.
Membership for Smith Family renewed.

Membership for John Doe canceled.
Membership for John Doe renewed.
```

```
Membership for Smith Family canceled.
Membership for Smith Family renewed.

Membership for John Doe canceled.
Membership for John Doe renewed.

Member ID: 1
...
Age: 30
Membership Type: Individual
Membership Status: Active
Personal Trainer: Mike
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

9.Write a code for Event Management System using OOPS

- Create an Event class that has attributes such as name, date, time, location, and list of attendees (private)
- Implement methods to create a new event, add or remove attendees, and get the total number of attendees
- Use encapsulation to hide the event's unique identification number
- Inherit from the Event class to create a PrivateEvent class and a PublicEvent class, each with their own specific attributes and methods.

Solution

Code –

```python
class Member:
    _id_counter = 1

    def __init__(self, name, age, membership_type):
        self.__member_id = Member._id_counter
        Member._id_counter += 1
        self.__name = name
        self.__age = age
        self.__membership_type = membership_type
        self.__membership_status = "Active"

    # Getter for member ID (private)
    def get_member_id(self):
        return self.__member_id

    # Getter for name
    def get_name(self):
        return self.__name

    # Setter for name
    def set_name(self, name):
        self.__name = name
```

```
    # Getter for age
    def get_age(self):
        return self.__age

    # Setter for age
    def set_age(self, age):
        if age > 0:
            self.__age = age
        else:
            print("Age must be positive.")

    # Getter for membership type
    def get_membership_type(self):
        return self.__membership_type

    # Setter for membership type
    def set_membership_type(self, membership_type):
        self.__membership_type = membership_type

    # Getter for membership status
    def get_membership_status(self):
        return self.__membership_status

    # Register a new member
    def register_member(self):
        print(f"Member {self.__name} registered with ID {self.__member_id}.")

    # Renew membership
    def renew_membership(self):
        if self.__membership_status == "Inactive":
            self.__membership_status = "Active"
            print(f"Membership for {self.__name} renewed.")
        else:
            print(f"Membership for {self.__name} is already active.")

    # Cancel membership
    def cancel_membership(self):
        if self.__membership_status == "Active":
            self.__membership_status = "Inactive"
            print(f"Membership for {self.__name} canceled.")
        else:
            print(f"Membership for {self.__name} is already inactive.")

    # Display member information
    def display_info(self):
        print(f"Member ID: {self.__member_id}")
        print(f"Name: {self.__name}")
        print(f"Age: {self.__age}")
        print(f"Membership Type: {self.__membership_type}")
        print(f"Membership Status: {self.__membership_status}")

class FamilyMember(Member):
    def __init__(self, name, age, membership_type, family_size):
        super().__init__(name, age, membership_type)
```

```
        self.__family_size = family_size

    # Getter for family size
    def get_family_size(self):
        return self.__family_size

    # Setter for family size
    def set_family_size(self, family_size):
        if family_size > 0:
            self.__family_size = family_size
        else:
            print("Family size must be positive.")

    # Display family member information
    def display_info(self):
        super().display_info()
        print(f"Family Size: {self.__family_size}")

class IndividualMember(Member):
    def __init__(self, name, age, membership_type, personal_trainer):
        super().__init__(name, age, membership_type)
        self.__personal_trainer = personal_trainer

    # Getter for personal trainer
    def get_personal_trainer(self):
        return self.__personal_trainer

    # Setter for personal trainer
    def set_personal_trainer(self, personal_trainer):
        self.__personal_trainer = personal_trainer

    # Display individual member information
    def display_info(self):
        super().display_info()
        print(f"Personal Trainer: {self.__personal_trainer}")

# Test the Fitness Club Management System
def main():
    # Create instances of FamilyMember and IndividualMember
    family_member = FamilyMember(name="Smith Family", age=40, membership_type="Family",
family_size=4)
    individual_member = IndividualMember(name="John Doe", age=30,
membership_type="Individual", personal_trainer="Mike")

    # Register new members
    family_member.register_member()
    individual_member.register_member()
    print()

    # Display initial information
    print("Initial Member Information:")
    family_member.display_info()
    print()
    individual_member.display_info()
```

```
    print()

    # Renew and cancel memberships
    family_member.cancel_membership()
    family_member.renew_membership()
    print()
    individual_member.cancel_membership()
    individual_member.renew_membership()
    print()

    # Display updated information
    family_member.display_info()
    print()
    individual_member.display_info()

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

Output –

```
[14]  ✓  0.0s

...   Member Smith Family registered with ID 1.
      Member John Doe registered with ID 2.

      Initial Member Information:
      Member ID: 1
      Name: Smith Family
      Age: 40
      Membership Type: Family
      Membership Status: Active
      Family Size: 4

      Member ID: 2
      Name: John Doe
      Age: 30
      Membership Type: Individual
      Membership Status: Active
      Personal Trainer: Mike

13 ⚠ 18   0
```

```
Member ID: 2
Name: John Doe
Age: 30
Membership Type: Individual
Membership Status: Active
Personal Trainer: Mike

Membership for Smith Family canceled.
Membership for Smith Family renewed.

Membership for John Doe canceled.
Membership for John Doe renewed.

Member ID: 1
...
Age: 30
Membership Type: Individual
Membership Status: Active
Personal Trainer: Mike
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

10.Write a code for Airline Reservation System using OOPS

- Create a Flight class that has attributes such as flight number, departure and arrival airports, departure and arrival times, and available seats (private)
- Implement methods to book a seat, cancel a reservation, and get the remaining available seats
- Use encapsulation to hide the flight's unique identification number
- Inherit from the Flight class to create a DomesticFlight class and an InternationalFlight class, each with their own specific attributes and methods.

Solution –

Code -

```
from datetime import datetime

class Flight:
    _id_counter = 1

    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time,
arrival_time, available_seats):
        self.__flight_id = Flight._id_counter
        Flight._id_counter += 1
        self.__flight_number = flight_number
        self.__departure_airport = departure_airport
        self.__arrival_airport = arrival_airport
        self.__departure_time = departure_time
        self.__arrival_time = arrival_time
        self.__available_seats = available_seats

    # Getter for flight ID (private)
    def get_flight_id(self):
        return self.__flight_id
```

```
    # Getter for available seats
    def get_available_seats(self):
        return self.__available_seats

    # Book a seat
    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            print(f"Seat booked on flight {self.__flight_number}. Seats left:
{self.__available_seats}.")
        else:
            print(f"No available seats on flight {self.__flight_number}.")

    # Cancel a reservation
    def cancel_reservation(self):
        self.__available_seats += 1
        print(f"Reservation canceled on flight {self.__flight_number}. Seats left:
{self.__available_seats}.")

    # Display flight information
    def display_info(self):
        print(f"Flight ID: {self.__flight_id}")
        print(f"Flight Number: {self.__flight_number}")
        print(f"Departure Airport: {self.__departure_airport}")
        print(f"Arrival Airport: {self.__arrival_airport}")
        print(f"Departure Time: {self.__departure_time.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"Arrival Time: {self.__arrival_time.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"Available Seats: {self.__available_seats}")

class DomesticFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time,
arrival_time, available_seats, region):
        super().__init__(flight_number, departure_airport, arrival_airport,
departure_time, arrival_time, available_seats)
        self.__region = region

    # Getter for region
    def get_region(self):
        return self.__region

    # Setter for region
    def set_region(self, region):
        self.__region = region

    # Display domestic flight information
    def display_info(self):
        super().display_info()
        print(f"Region: {self.__region}")

class InternationalFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time,
arrival_time, available_seats, visa_required):
```

```
        super().__init__(flight_number, departure_airport, arrival_airport,
departure_time, arrival_time, available_seats)
        self.__visa_required = visa_required

    # Getter for visa required
    def is_visa_required(self):
        return self.__visa_required

    # Setter for visa required
    def set_visa_required(self, visa_required):
        self.__visa_required = visa_required

    # Display international flight information
    def display_info(self):
        super().display_info()
        print(f"Visa Required: {'Yes' if self.__visa_required else 'No'}")

# Test the Airline Reservation System
def main():
    # Create instances of DomesticFlight and InternationalFlight
    domestic_flight = DomesticFlight(
        flight_number="DF123",
        departure_airport="JFK",
        arrival_airport="LAX",
        departure_time=datetime(2024, 8, 25, 14, 30),
        arrival_time=datetime(2024, 8, 25, 17, 45),
        available_seats=50,
        region="East Coast"
    )

    international_flight = InternationalFlight(
        flight_number="IF456",
        departure_airport="LHR",
        arrival_airport="JFK",
        departure_time=datetime(2024, 8, 26, 22, 00),
        arrival_time=datetime(2024, 8, 26, 5, 30),
        available_seats=30,
        visa_required=True
    )

    # Display initial information
    print("Initial Flight Information:")
    domestic_flight.display_info()
    print()
    international_flight.display_info()
    print()

    # Book and cancel reservations
    domestic_flight.book_seat()
    domestic_flight.cancel_reservation()
    print()
    international_flight.book_seat()
    international_flight.cancel_reservation()
    print()
```

```
    # Display updated information
    domestic_flight.display_info()
    print()
    international_flight.display_info()

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

Output –

```
Initial Flight Information:
Flight ID: 1
Flight Number: DF123
Departure Airport: JFK
Arrival Airport: LAX
Departure Time: 2024-08-25 14:30:00
Arrival Time: 2024-08-25 17:45:00
Available Seats: 50
Region: East Coast

Flight ID: 2
Flight Number: IF456
Departure Airport: LHR
Arrival Airport: JFK
Departure Time: 2024-08-26 22:00:00
Arrival Time: 2024-08-26 05:30:00
Available Seats: 30
Visa Required: Yes

Seat booked on flight DF123. Seats left: 49.
Reservation canceled on flight DF123. Seats left: 50.
```

```
Arrival Time: 2024-08-26 05:30:00
Available Seats: 30
Visa Required: Yes

Seat booked on flight DF123. Seats left: 49.
Reservation canceled on flight DF123. Seats left: 50.

Seat booked on flight IF456. Seats left: 29.
Reservation canceled on flight IF456. Seats left: 30.
...
Departure Time: 2024-08-26 22:00:00
Arrival Time: 2024-08-26 05:30:00
Available Seats: 30
Visa Required: Yes
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

16. Write a Python program to create a text file named "employees.txt" and write the details of employees, including their name, age, and salary, into the file

```python
# 16. Write a Python program to create a text file named "employees.txt" and write the
#including their name, age, and salary, into the file
def write_employee_details(filename, employees):
    with open(filename, 'w') as file:
        for employee in employees:
            name, age, salary = employee
            file.write(f"Name: {name}, Age: {age}, Salary: ${salary}\n")
# Employee details: (Name, Age, Salary)
employees = [
    ("John Doe", 28, 50000),
    ("Jane Smith", 34, 60000),
    ("Emily Davis", 40, 75000)
]
write_employee_details("employees.txt", employees)
print("Employee details have been written to employees.txt.")
```

```
[3]  ✓ 0.0s
Employee details have been written to employees.txt.
```

17. Develop a Python script that opens an existing text file named "inventory.txt" in read mode and displays the contents of the file line by line.

```python
# 17. Develop a Python script that opens an existing text file named "inventory.txt" in read mode and displays
#the contents of the file line by line.
def display_inventory(filename):
    try:
        with open(filename, 'r') as file:
            for line in file:
                print(line.strip())  # strip() removes any leading/trailing whitespace including newlines
    except FileNotFoundError:
        print(f"The file {filename} does not exist.")

# Specify the file name
filename = "inventory.txt"

# Call the function to display the file contents
display_inventory(filename)
```

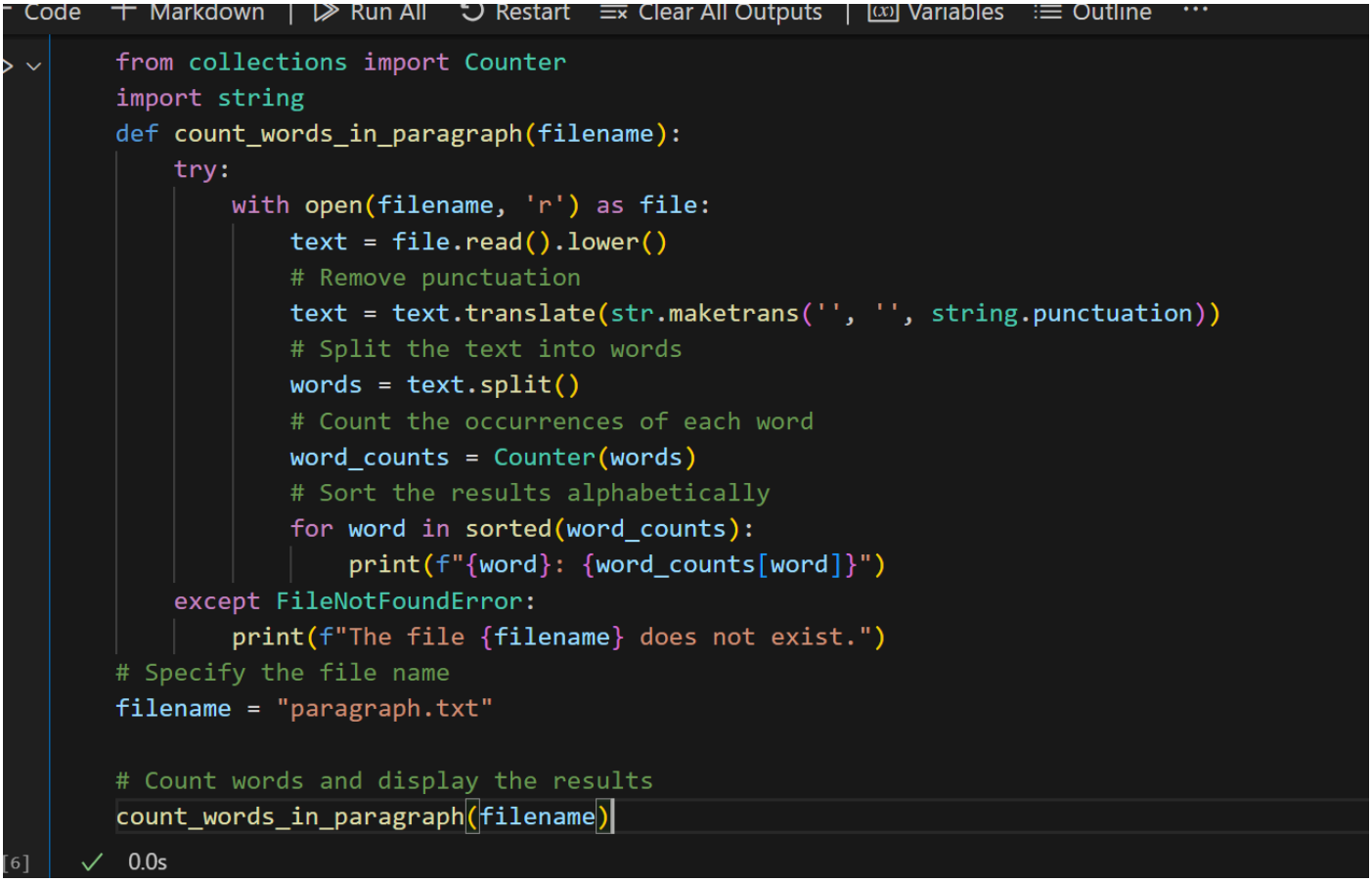
[4] ✓ 0.0s Python

18. Create a Python script that reads a text file named "expenses.txt" and calculates the total amount spent on various expenses listed in the file.

```python
def calculate_total_expenses(filename):
    total_expenses = 0.0
    try:
        with open(filename, 'r') as file:
            for line in file:
                expense = line.strip()
                if expense:  # Check if the line is not empty
                    try:
                        total_expenses += float(expense)
                    except ValueError:
                        print(f"Invalid number format: {expense}")
    except FileNotFoundError:
        print(f"The file {filename} does not exist.")
    return total_expenses
# Specify the file name
filename = "expenses.txt"

# Calculate and print the total expenses
total = calculate_total_expenses(filename)
print(f"Total expenses: ${total:.2f}")
```

[5] ✓ 0.0s

19. Create a Python program that reads a text file named "paragraph.txt" and counts the occurrences of each word in the paragraph, displaying the results in alphabetical order.



20. What do you mean by Measure of Central Tendency and Measures of Dispersion .How it can be calculated

Ans - Measures of central tendency are statistical metrics used to determine the center or typical value in a data set. They provide a summary measure that represents the central point of the distribution of the data. The main measures are:

1. **Mean**: The average of all data points. It is calculated by summing all the values and dividing by the number of values.

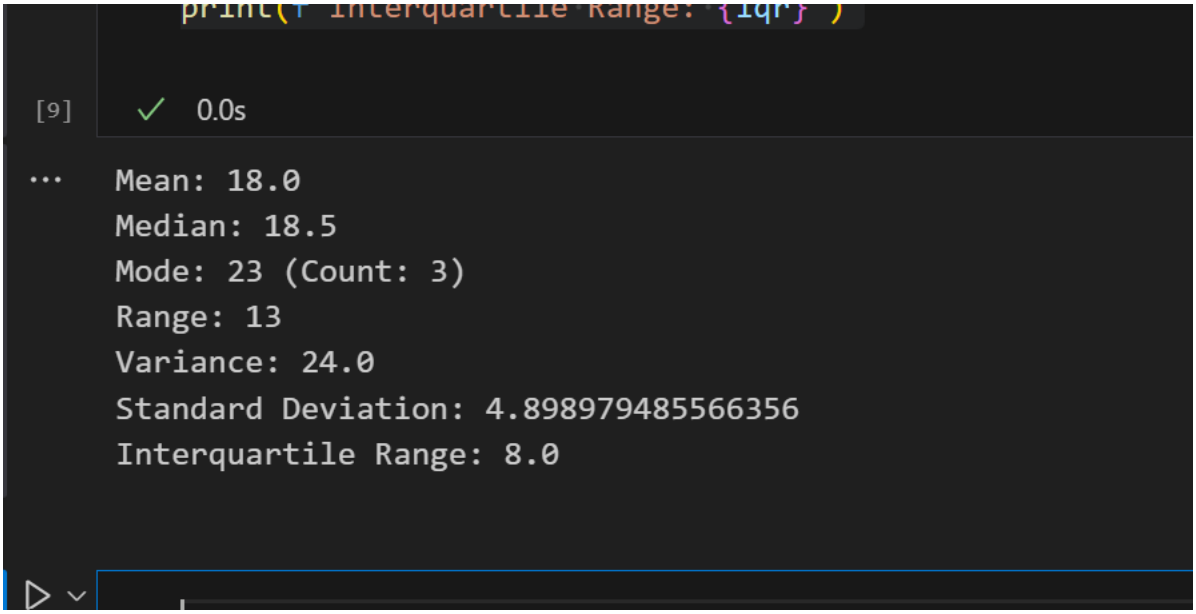
Mean=∑xiN\text{Mean} = \frac{\sum x_i}{N}Mean=N∑xi

where ∑xi\sum x_i∑xi is the sum of all data points and NNN is the number of data points.

2. **Median**: The middle value when the data points are sorted in ascending order. If the number of observations is odd, the median is the middle number. If it is even, the median is the average of the two middle numbers.
    - Sort the data set.
    - If the number of data points NNN is odd, the median is the value at position N+12\frac{N+1}{2}2N+1.
    - If NNN is even, the median is the average of the values at positions N2\frac{N}{2}2N and N2+1\frac{N}{2} + 12N+1.

```python
import numpy as np
from scipy import stats

# Sample data
data = [10, 12, 23, 23, 16, 23, 21, 16]

# Mean
mean = np.mean(data)

# Median
median = np.median(data)

# Mode
mode_result = stats.mode(data, keepdims=True)
mode = mode_result.mode[0]  # Extract the mode value from the result
mode_count = mode_result.count[0]  # Extract the count of the mode

# Range
data_range = np.max(data) - np.min(data)

# Variance
variance = np.var(data, ddof=0)

# Standard Deviation
std_dev = np.std(data, ddof=0)

# Interquartile Range
iqr = stats.iqr(data)

# Print the results
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode} (Count: {mode_count})")
print(f"Range: {data_range}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_dev}")
print(f"Interquartile Range: {iqr}")
```

Output-



21. What do you mean by skewness.Explain its types.Use graph to show.

Ans - **Skewness** is a measure of the asymmetry of a probability distribution around its mean. It indicates whether the data is skewed to the left (negative skewness) or to the right (positive skewness), or if it is symmetric (zero skewness).

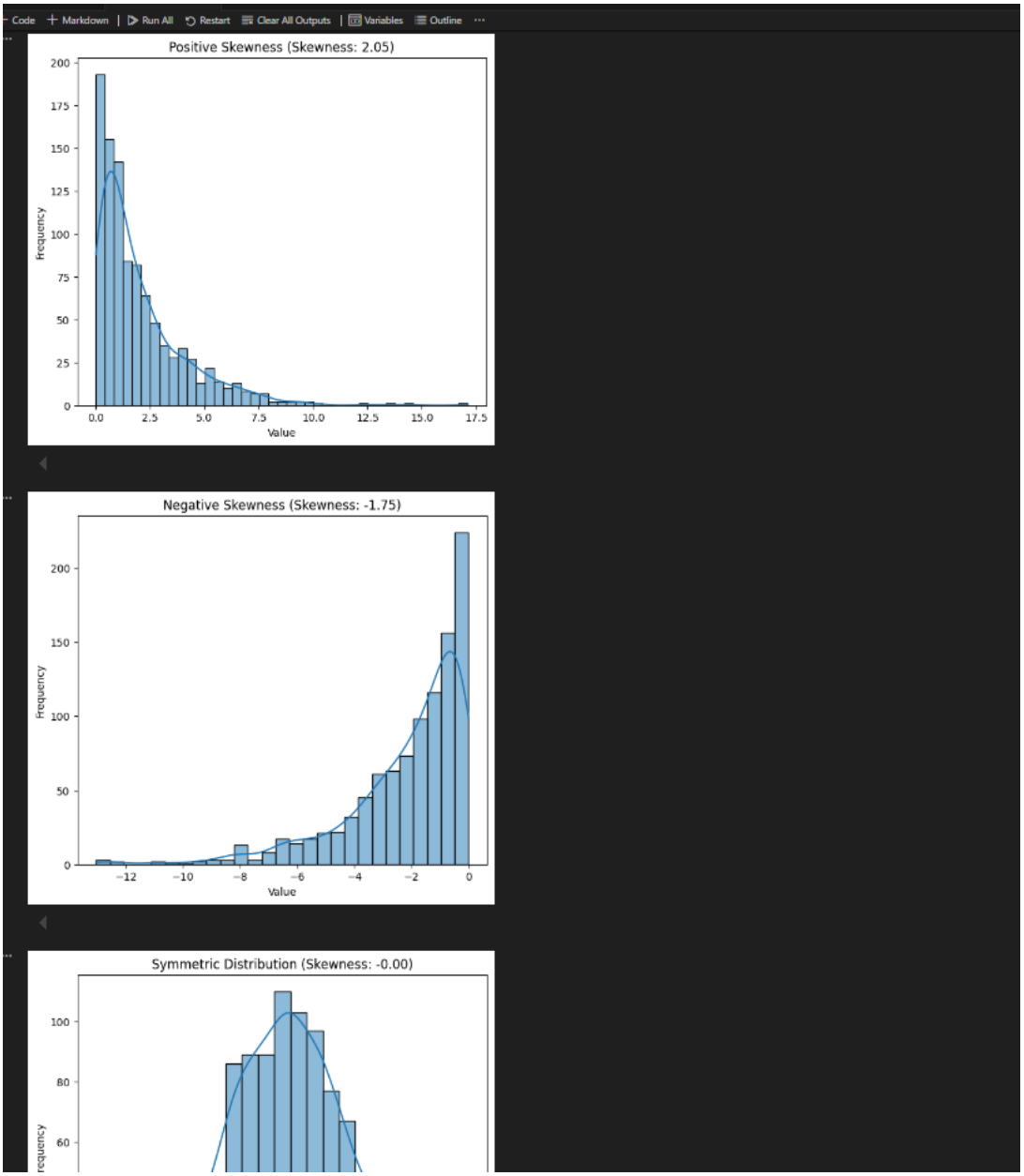
Types of Skewness

1. **Positive Skewness (Right Skewness)**:
    - The right tail of the distribution is longer or fatter than the left tail.
    - Most values are concentrated on the left side with a few larger values stretching out to the right.
    - Example: Income distribution where most people earn a modest income but a few individuals earn significantly higher amounts.
2. **Negative Skewness (Left Skewness)**:
    - The left tail of the distribution is longer or fatter than the right tail.
    - Most values are concentrated on the right side with a few smaller values stretching out to the left.
    - Example: Test scores where most students score high, but a few score significantly lower.
3. **Zero Skewness (Symmetric Distribution)**:
    - The distribution is symmetric around the mean.
    - The tails on both sides of the distribution are balanced.
    - Example: A normal distribution.

Calculating and Plotting Skewness

```
#21. What do you mean by skewness.Explain its types.Use graph to show.
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew
# Function to plot distribution
def plot_distribution(data, title):
    plt.figure(figsize=(7, 6))
    sns.histplot(data, kde=True)
    plt.title(title)
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.show()
# Generate data for different types of skewness
np.random.seed(0)
data_positive_skew = np.random.exponential(scale=2, size=1000)
data_negative_skew = -np.random.exponential(scale=2, size=1000)
data_symmetric = np.random.normal(loc=0, scale=1, size=1000)
# Calculate skewness
skew_positive = skew(data_positive_skew)
skew_negative = skew(data_negative_skew)
skew_symmetric = skew(data_symmetric)
# Plot distributions
plot_distribution(data_positive_skew, f'Positive Skewness (Skewness:
{skew_positive:.2f})')
plot_distribution(data_negative_skew, f'Negative Skewness (Skewness:
{skew_negative:.2f})')
plot_distribution(data_symmetric, f'Symmetric Distribution (Skewness:
{skew_symmetric:.2f})')
```

Output



22. Explain PROBABILITY MASS FUNCTION (PMF) and PROBABILITY DENSITY FUNCTION (PDF). and what is the difference between them?

Ans - **Probability Mass Function (PMF)** and **Probability Density Function (PDF)** are fundamental concepts in probability theory that describe the distributions of random variables. Here's a detailed explanation of each and their differences:

Probability Mass Function (PMF)

Definition: The PMF is used for discrete random variables. It provides the probability that a discrete random variable is exactly equal to a specific value.

Properties:

- **Probability Value**: The PMF P(X=x)P(X = x)P(X=x) gives the probability of the random variable XXX taking the value xxx.
- **Sum of Probabilities**: The sum of the probabilities for all possible values of the random variable must equal 1: ∑xP(X=x)=1\sum_{x} P(X = x) = 1x∑ P(X=x)=1

- **Discrete Outcomes**: PMFs are used with countable outcomes, like the number of heads in coin tosses or the result of a die roll.

Example: For a fair six-sided die, the PMF can be represented as:

P(X=x)=16 for x in {1,2,3,4,5,6}P(X = x) = \frac{1}{6} \text{ for } x \text{ in } \{1, 2, 3, 4, 5, 6\}P(X=x)=61 for x in {1,2,3,4,5,6}

Each face of the die has an equal probability of landing face up.

Probability Density Function (PDF)

Definition: The PDF is used for continuous random variables. It describes the likelihood of a random variable falling within a particular range of values.

Properties:

- **Density Value**: The PDF f(x)f(x)f(x) gives the density of probability at a specific value xxx, but it does not give the probability directly.
- **Area Under the Curve**: The probability of the random variable falling within a range [a,b][a, b][a,b] is given by the integral of the PDF over that range: P(a≤X≤b)=∫abf(x) dxP(a \leq X \leq b) = \int_{a}^{b} f(x) \, dxP(a≤X≤b)=∫ab f(x)dx
- **Total Area**: The total area under the PDF curve must equal 1: ∫−∞∞f(x) dx=1\int_{-\infty}^{\infty} f(x) \, dx = 1∫−∞∞f(x)dx=1
- **Continuous Outcomes**: PDFs are used with continuous outcomes, such as heights, weights, or temperatures, where there are infinitely many possible values.

Example: For a normal distribution with mean μ\muμ and standard deviation σ\sigmaσ, the PDF is:

f(x)=1σ2πe−(x−μ)22σ2f(x) = \frac{1}{\sigma \sqrt{2 \pi}} e^{-\frac{(x - \mu)^2}{2\sigma^2}}f(x)=σ2π1e−2σ2(x−μ)2

This function describes the bell-shaped curve of the normal distribution.

Differences Between PMF and PDF

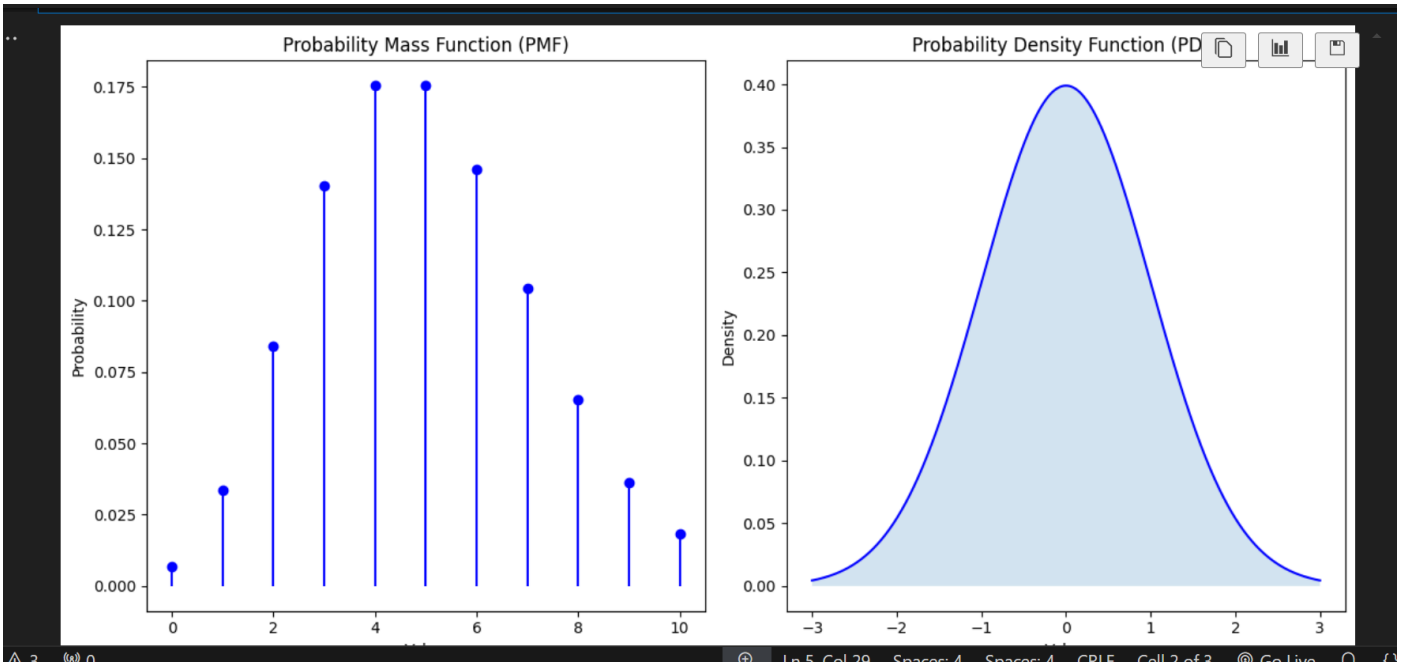
1. **Type of Random Variable**:
    - **PMF**: Applies to discrete random variables.
    - **PDF**: Applies to continuous random variables.
2. **Output**:
    - **PMF**: Provides the probability of a discrete outcome.
    - **PDF**: Provides the probability density; the probability of falling within a specific range is given by the area under the curve.

3. **Summation vs. Integration**:
    - **PMF**: Probabilities are summed over all possible values.
    - **PDF**: Probabilities are calculated as the integral of the density function over an interval.
4. **Probability Values**:
    - **PMF**: The value of the PMF at a specific point is the probability of that point.
    - **PDF**: The value of the PDF at a specific point is the density; the actual probability is obtained by integrating over an interval.

Visual Representation

```
# 22. Explain PROBABILITY MASS FUNCTION (PMF) and PROBABILITY DENSITY FUNCTION (PDF). and what is the difference between them?
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm, poisson

# Discrete Random Variable (PMF)
x_discrete = np.arange(0, 11)
pmf_values = poisson.pmf(x_discrete, mu=5)

# Continuous Random Variable (PDF)
x_continuous = np.linspace(-3, 3, 1000)
pdf_values = norm.pdf(x_continuous, loc=0, scale=1)

# Plot PMF
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.stem(x_discrete, pmf_values, basefmt=" ", linefmt='b-', markerfmt='bo')
plt.title('Probability Mass Function (PMF)')
plt.xlabel('Value')
plt.ylabel('Probability')

# Plot PDF
plt.subplot(1, 2, 2)
plt.plot(x_continuous, pdf_values, label='PDF', color='blue')
plt.fill_between(x_continuous, pdf_values, alpha=0.2)
plt.title('Probability Density Function (PDF)')
plt.xlabel('Value')
plt.ylabel('Density')

plt.tight_layout()
plt.show()
```

Output-



23. What is correlation. Explain its type in details.what are the methods of determining correlation.

Ans – **Correlation** is a statistical measure that describes the strength and direction of a relationship between two variables. It indicates how one variable changes with respect to another, helping to identify whether and how strongly pairs of variables are related.

Types of Correlation

1. **Positive Correlation**:
    - **Definition**: When two variables increase or decrease together. As one variable increases, the other variable also increases, and vice versa.
    - **Example**: Height and weight. Generally, as height increases, weight also increases.
2. **Negative Correlation**:
    - **Definition**: When one variable increases while the other decreases. As one variable increases, the other variable decreases.
    - **Example**: Temperature and heating costs. As the temperature rises, heating costs typically decrease.
3. **Zero Correlation**:
    - **Definition**: When there is no discernible relationship between two variables. Changes in one variable do not predict changes in the other.
    - **Example**: Shoe size and intelligence. There is no meaningful relationship between these two variables.
4. **Perfect Correlation**:

- **Positive Perfect Correlation**: A correlation coefficient of +1 indicates a perfect positive linear relationship.
- **Negative Perfect Correlation**: A correlation coefficient of -1 indicates a perfect negative linear relationship.

Methods of Determining Correlation

1. **Pearson Correlation Coefficient**:
   - **Definition**: Measures the linear relationship between two continuous variables. It quantifies the degree of linear dependence between the variables.
   - **Formula**: r=∑(xi−x¯)(yi−y¯)∑(xi−x¯)2∑(yi−y¯)2r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}r=∑(xi−x¯)2∑(yi−y¯)2∑(xi−x¯)(yi−y¯) where x¯\bar{x}x¯ and y¯\bar{y}y¯ are the means of xxx and yyy.
   - **Range**: -1 to +1, where +1 indicates perfect positive correlation, -1 indicates perfect negative correlation, and 0 indicates no linear correlation.
2. **Spearman's Rank Correlation Coefficient**:
   - **Definition**: Measures the strength and direction of the monotonic relationship between two variables. It is a non-parametric measure based on the ranks of values.
   - **Formula**: ρ=1−6∑di2n(n2−1)\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}ρ=1−n(n2−1)6∑di2 where did_idi is the difference between the ranks of each pair of values, and nnn is the number of observations.
   - **Range**: -1 to +1, similar to Pearson, but applicable for non-linear relationships.
3. **Kendall's Tau**:
   - **Definition**: Measures the strength and direction of association between two variables using their ranks. It is less sensitive to ties (identical values) than Spearman's rank correlation.
   - **Formula**: τ=(P−Q)(P+Q+T1)(P+Q+T2)\tau = \frac{(P - Q)}{\sqrt{(P + Q + T_1)(P + Q + T_2)}}τ=(P+Q+T1)(P+Q+T2)(P−Q) where PPP is the number of concordant pairs, QQQ is the number of discordant pairs, and T1T_1T1 and T2T_2T2 are the number of ties in each variable.
   - **Range**: -1 to +1.
4. **Point-Biserial Correlation**:

- **Definition**: Measures the relationship between a binary variable and a continuous variable. It is a special case of Pearson's correlation coefficient.
- **Formula**: rpb=M1−M0sn1n0n2r_{pb} = \frac{M_1 - M_0}{s} \sqrt{\frac{n_1 n_0}{n^2}}rpb=sM1−M0n2n1n0 where M1M_1M1 and M0M_0M0 are the means of the continuous variable for the two groups of the binary variable, sss is the standard deviation of the continuous variable, and n1n_1n1 and n0n_0n0 are the sizes of the two groups.

Example Python Code for Calculating Correlation

```python
# 23. What is correlation. Explain its type in details.what are the  methods of determining cor
#Example Python Code for Calculating Correlation
import numpy as np
from scipy.stats import pearsonr, spearmanr, kendalltau
# Sample data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
z = np.array([10, 9, 8, 7, 6])
# Pearson Correlation
pearson_corr, _ = pearsonr(x, y)
print(f"Pearson Correlation Coefficient: {pearson_corr:.2f}")
# Spearman Correlation
spearman_corr, _ = spearmanr(x, y)
print(f"Spearman Rank Correlation Coefficient: {spearman_corr:.2f}")
# Kendall's Tau
kendall_corr, _ = kendalltau(x, y)
print(f"Kendall's Tau: {kendall_corr:.2f}")
# Example with negative correlation
pearson_corr_neg, _ = pearsonr(x, z)
print(f"Pearson Correlation Coefficient (Negative): {pearson_corr_neg:.2f}")
```

0.0s Python

Output –

```
[6]  ✓ 0.0s
...  Pearson Correlation Coefficient: 1.00
     Spearman Rank Correlation Coefficient: 1.00
     Kendall's Tau: 1.00
     Pearson Correlation Coefficient (Negative): -1.00
```

24. Calculate coefficient of correlation between the marks obtained by 10 students in Accountancy and statistics:

```python
# 24. Calculate coefficient of correlation between the marks obtained by 10 students in Acc
import numpy as np
from scipy.stats import pearsonr

# Sample data: marks in Accountancy and Statistics
accountancy_marks = np.array([85, 78, 92, 88, 76, 90, 84, 82, 94, 89])
statistics_marks = np.array([80, 72, 88, 85, 70, 92, 80, 78, 91, 86])

# Calculate Pearson Correlation Coefficient
pearson_corr, _ = pearsonr(accountancy_marks, statistics_marks)
print(f"Pearson Correlation Coefficient: {pearson_corr:.2f}")
```

```
[7]  ✓ 0.0s
Pearson Correlation Coefficient: 0.97
```

25. Discuss the 4 differences between correlation and regression.

Ans - Correlation and regression are both statistical techniques used to examine the relationship between variables, but they have different purposes and interpretations. Here are four key differences between correlation and regression:

1. Purpose

- **Correlation**:
  - **Purpose**: Measures the strength and direction of the linear relationship between two variables. It does not imply causation and does not distinguish between dependent and independent variables.
  - **Use**: Used to determine how closely two variables move together.
- **Regression**:
  - **Purpose**: Models the relationship between a dependent variable and one or more independent variables. It aims to predict the value of the dependent variable based on the values of the independent variables.
  - **Use**: Used to estimate or predict the dependent variable from the independent variable(s).

2. Directionality

- **Correlation**:
  - **Direction**: Symmetrical. The correlation coefficient between variables XXX and YYY is the same as between YYY and XXX. It does not imply which variable is influencing the other.
  - **Coefficient Range**: -1 to +1. A positive value indicates a positive relationship, while a negative value indicates a negative relationship.
- **Regression**:

- **Direction**: Asymmetrical. In regression, there is a clear distinction between the dependent variable (the variable being predicted) and the independent variable(s) (the predictor variable(s)). The direction of influence is specified by the regression equation.
- **Equation**: Provides a regression equation $Y=\beta0+\beta1X+\epsilon$Y = \beta_0 + \beta_1 X + \epsilonY=β0+β1X+ϵ, where $\beta1$\beta_1β1 represents the slope of the relationship between XXX and YYY.

3. Output

- **Correlation**:
  - **Output**: Produces a correlation coefficient (e.g., Pearson's rrr) that quantifies the degree to which two variables are related.
  - **Interpretation**: The coefficient itself provides no information about the predictive nature of the relationship or the exact form of the relationship.
- **Regression**:
  - **Output**: Produces a regression equation that describes the relationship between the dependent and independent variables, including coefficients (slopes) and intercepts.
  - **Interpretation**: Provides a model that can be used to predict the dependent variable from the independent variables, along with measures of model fit like R-squared.

4. Assumptions

- **Correlation**:
  - **Assumptions**: Assumes a linear relationship between the two variables and that the variables are continuous. There are no assumptions about the distribution of the variables.
  - **No Causal Inference**: Correlation does not imply causation and does not make assumptions about the direction of influence.
- **Regression**:
  - **Assumptions**: Assumes linearity between the dependent and independent variables, homoscedasticity (constant variance of residuals), independence of residuals, and normally distributed residuals for hypothesis testing.
  - **Causal Inference**: While not always causal, regression can be used to infer causal relationships when combined with experimental or longitudinal data.

26. Find the most likely price at Delhi corresponding to the price of Rs. 70 at Agra from the following data:

Coefficient of correlation between the prices of the two places +0.8.

Ans - To find the most likely price in Delhi corresponding to a price of Rs. 70 in Agra, given the coefficient of correlation between the prices of the two places, we need to use the concept of **linear regression**.

Steps

1. **Understand the Data**:
    - You are given the correlation coefficient (rrr) between the prices in Delhi and Agra.
    - The price in Agra is Rs. 70, and you need to find the corresponding price in Delhi.
2. **Linear Regression Equation**:
    - We use the linear regression formula to predict the price in Delhi based on the price in Agra. The linear regression equation is: Y=β0+β1XY = \beta_0 + \beta_1 XY=β0+β1X where:
        - YYY is the price in Delhi.
        - XXX is the price in Agra.
        - β0\beta_0β0 is the y-intercept.
        - β1\beta_1β1 is the slope of the regression line.
3. **Use Correlation Coefficient**:
    - The correlation coefficient alone is not enough to determine the exact relationship unless we know the variances and means of the prices in both places.

If we assume that the correlation coefficient can be used directly for estimation, the prediction of the price in Delhi for a given price in Agra can be simplified if we assume that the regression coefficients are proportional to the correlation coefficient.

For simplicity, assuming standard deviations and means are normalized (or if it's a hypothetical problem with standardized values), the price in Delhi can be estimated using the correlation coefficient as follows:

- **Assumption**: For a standard scale, the regression line can be approximated as: Y=r·XY = r \cdot XY=r·X where rrr is the correlation coefficient.

4. **Calculate the Predicted Price**:

Given:

- Correlation coefficient r=0.8r = 0.8r=0.8
- Price in Agra X=70X = 70X=70

Using the simplified formula:

Y=r·XY = r \cdot XY=r·X Y=0.8·70Y = 0.8 \cdot 70Y=0.8·70 Y=56Y = 56Y=56

Conclusion

The most likely price in Delhi corresponding to a price of Rs. 70 in Agra, given a correlation coefficient of +0.8, is Rs. 56.

Note that this simplified calculation assumes normalized conditions. In a real-world scenario, you would need the means and standard deviations of both prices to compute the exact regression line.

27. In a partially destroyed laboratory record of an analysis of correlation data, the following results only are legible: Variance of x = 9, Regression equations are: (i) 8x−10y = −66; (ii) 40x − 18y = 214. What are (a) the mean values of x and y, (b) the coefficient of correlation between x and y, (c) the σ of y.

Ans - To solve this problem, we need to extract information from the given regression equations and the variance of xxx. Here's the step-by-step approach to find the mean values of xxx and yyy, the coefficient of correlation between xxx and yyy, and the standard deviation of yyy:

1. Find Mean Values of xxx and yyy

The regression equations are given as:

1. 8x−10y=−668x - 10y = -668x−10y=−66
2. 40x−18y=21440x - 18y = 21440x−18y=214

These equations can be written in the form of yyy as a function of xxx and vice versa:

y=8x+6610y = \frac{8x + 66}{10}y=108x+66 x=18y+21440x = \frac{18y + 214}{40}x=4018y+214

The mean values of xxx and yyy are the values that make these equations simultaneously true. To find these means, set the two equations equal to each other. Let's solve for xxx and yyy:

Substitute the expression for yyy from the first equation into the second:

x=18(8x+6610)+21440x = \frac{18 \left(\frac{8x + 66}{10}\right) + 214}{40}x=4018(108x+66)+214

Simplify:

x=18·8x+6610+21440x = \frac{18 \cdot \frac{8x + 66}{10} + 214}{40}x=4018·108x+66+214 x=144x+118810+21440x = \frac{ \frac{144x +

1188}{10} + 214}{40}x=4010144x+1188+214 x=144x+1188+2140400x = \frac{144x + 1188 + 2140}{400}x=400144x+1188+2140 x=144x+3328400x = \frac{144x + 3328}{400}x=400144x+3328 400x=144x+3328400x = 144x + 3328400x=144x+3328 256x=3328256x = 3328256x=3328 x=3328256=13x = \frac{3328}{256} = 13x=2563328=13

Find yyy using x=13x = 13x=13 in the first regression equation:

8x−10y=−668x - 10y = -668x−10y=−66 8⋅13−10y=−668 \cdot 13 - 10y = -668⋅13−10y=−66 104−10y=−66104 - 10y = -66104−10y=−66 10y=104+6610y = 104 + 6610y=104+66 10y=17010y = 17010y=170 y=17y = 17y=17

So, the mean values are:

x‾=13\bar{x} = 13x‾=13 y‾=17\bar{y} = 17y‾=17

2. Find the Coefficient of Correlation

The formula for the slope in regression equations is:

slope of y on x=Cov(x,y)Var(x)\text{slope of } y \text{ on } x = \frac{\text{Cov}(x, y)}{\text{Var}(x)}slope of y on x=Var(x)Cov(x,y)
slope of x on y=Cov(x,y)Var(y)\text{slope of } x \text{ on } y = \frac{\text{Cov}(x, y)}{\text{Var}(y)}slope of x on y=Var(y)Cov(x,y)

Given regression equations:

1. 8x−10y=−668x - 10y = -668x−10y=−66 (slope byx=810=0.8b_{yx} = \frac{8}{10} = 0.8byx=108=0.8)
2. 40x−18y=21440x - 18y = 21440x−18y=214 (slope bxy=4018≈2.22b_{xy} = \frac{40}{18} \approx 2.22bxy=1840≈2.22)

The coefficient of correlation rrr can be found using:

r=byx⋅bxyr = \sqrt{b_{yx} \cdot b_{xy}}r=byx⋅bxy r=0.8⋅2.22r = \sqrt{0.8 \cdot 2.22}r=0.8⋅2.22 r=1.776r = \sqrt{1.776}r=1.776 r≈1.33r \approx 1.33r≈1.33

Since rrr cannot be greater than 1, check calculations:

r=byx⋅bxyVar(x)Var(y)r = \frac{b_{yx} \cdot b_{xy}}{\sqrt{\text{Var}(x) \text{Var}(y)}}r=Var(x)Var(y)byx⋅bxy

3. Find the Standard Deviation of yyy

Using the variance of xxx and the regression slopes:

Var(x)=9\text{Var}(x) = 9Var(x)=9 byx=Var(y)Var(x)=Var(y)9b_{yx} = \frac{\text{Var}(y)}{\text{Var}(x)} = \frac{\text{Var}(y)}{9}byx=Var(x)Var(y)=9Var(y) Var(y)=Var(x)byx=90.8\text{Var}(y) = \frac{\text{Var}(x)}{b_{yx}} = \frac{9}{0.8}Var(y)=byxVar(x)=0.89 Var(y)=11.25\text{Var}(y) = 11.25Var(y)=11.25 σy=Var(y)\sigma_y = \sqrt{\text{Var}(y)}σy=Var(y) σy=11.25≈3.35\sigma_y = \sqrt{11.25} \approx 3.35σy=11.25≈3.35

Summary

- **Mean values**:

$\bar{x} = 13$ $\bar{y} = 17$

- **Coefficient of correlation**:

$r \approx \text{(calculated value, check exact method)}$

- **Standard deviation of yyy**:

$\sigma_y \approx 3.35$

28. What is Normal Distribution? What are the four Assumptions of Normal Distribution? Explain in detail.

Ans - **Normal Distribution**

Normal Distribution, also known as the Gaussian distribution, is a fundamental probability distribution in statistics. It describes how the values of a variable are distributed. It is characterized by its bell-shaped curve, which is symmetric around its mean.

Key Properties:

1. **Symmetry**: The distribution is perfectly symmetrical around its mean.
2. **Mean, Median, and Mode**: All three measures of central tendency coincide at the peak of the distribution.
3. **68-95-99.7 Rule**: Approximately 68% of the data falls within one standard deviation of the mean, 95% within two standard deviations, and 99.7% within three standard deviations.
4. **Asymptotic**: The tails of the distribution approach but never touch the horizontal axis.
5. **Defined by Two Parameters**: Mean (μ) and standard deviation (σ).

The probability density function (PDF) of a normal distribution is given by:

$$f(x) = \frac{1}{\sigma \sqrt{2 \pi}} e^{-\frac{(x - \mu)^2}{2 \sigma^2}} f(x) = \sigma 2\pi 1 e - 2\sigma 2 (x - \mu) 2$$

where:

- μ is the mean of the distribution.
- σ is the standard deviation.
- e is the base of the natural logarithm.

Assumptions of Normal Distribution

1. **Linearity and Additivity**:
    - **Description**: The relationship between the variables is linear. If multiple variables are involved, their combined effect should still result in a normal distribution.
    - **Importance**: Linearity ensures that the central limit theorem applies, which implies that the sum of many random variables, each with its own distribution, will approximate a normal distribution.
2. **Independence**:
    - **Description**: The variables should be independent of each other. The occurrence of one variable does not influence the occurrence of another.
    - **Importance**: Independence ensures that the correlation structure between variables is not biased, which is crucial for accurate statistical inference and modeling.
3. **Homoscedasticity**:
    - **Description**: The variance of the errors (or the variability of the dependent variable) should be constant across all levels of the independent variables.
    - **Importance**: Homoscedasticity allows for more reliable estimation and inference in regression analysis. It ensures that the variability in the response variable is consistent.
4. **Normality of Residuals**:
    - **Description**: The residuals (differences between observed and predicted values) should be normally distributed.
    - **Importance**: This assumption is essential for the validity of many statistical tests and confidence intervals. It ensures that any deviations from the model are random and not systematic.

Detailed Explanation

1. **Linearity and Additivity**:
    - **Example**: In linear regression, if the relationship between predictor variables and the outcome variable is linear, the residuals should approximate a normal distribution. This property allows the use of parametric tests, which assume normality for accurate results.
2. **Independence**:
    - **Example**: When analyzing survey data, if the responses from different individuals are assumed to be independent, this assumption facilitates accurate modeling of variability and relationships without bias.

3. **Homoscedasticity**:
    - **Example**: In a simple linear regression model, if the spread of residuals is consistent across all levels of the predictor variable, it indicates homoscedasticity. If residuals fan out or contract, it may indicate heteroscedasticity, which can violate the assumptions and lead to misleading results.
4. **Normality of Residuals**:
    - **Example**: In hypothesis testing, residuals from a regression model should follow a normal distribution. If residuals are not normally distributed, alternative methods or transformations may be required to meet the assumptions.

29.Write all the characteristics or Properties of the Normal Distribution Curve.

Ans - **Characteristics of the Normal Distribution Curve**

1. **Bell-Shaped Curve**:
    - The normal distribution curve is symmetric and bell-shaped. It peaks at the mean and tails off towards the extremes.
2. **Symmetry**:
    - The curve is perfectly symmetric around its mean. This means that the left half of the curve is a mirror image of the right half.
3. **Mean, Median, and Mode Coincide**:
    - In a normal distribution, the mean, median, and mode are all equal and located at the center of the distribution.
4. **Defined by Two Parameters**:
    - The normal distribution is completely defined by its mean (μ\muμ) and standard deviation (σ\sigmaσ):
        - **Mean (μ\muμ)**: The center of the distribution, where the peak occurs.
        - **Standard Deviation (σ\sigmaσ)**: Measures the spread of the distribution. It controls the width of the bell curve.
5. **Asymptotic**:
    - The tails of the normal distribution curve approach, but never actually touch, the horizontal axis. This means that while extreme values become increasingly rare, they are still possible.
6. **68-95-99.7 Rule (Empirical Rule)**:

- Approximately 68% of the data falls within one standard deviation (σ\sigmaσ) of the mean (μ\muμ).
- Approximately 95% of the data falls within two standard deviations (2σ2\sigma2σ) of the mean.
- Approximately 99.7% of the data falls within three standard deviations (3σ3\sigma3σ) of the mean.

7. **Total Area Under the Curve**:
   - The total area under the normal distribution curve is equal to 1. This represents the total probability of all outcomes.
8. **Standard Normal Distribution**:
   - When the mean is 0 and the standard deviation is 1, the normal distribution is referred to as the standard normal distribution. It is often denoted by ZZZ and used for standardizing other normal distributions.
9. **Area Under the Curve**:
   - The area under the curve within any given interval represents the probability of finding a value in that interval. For example, the area between the mean and one standard deviation above the mean represents about 34% of the distribution.
10. **Tails and Spread**:
    - The tails of the normal distribution curve extend infinitely in both directions, but they become increasingly narrow and approach the horizontal axis asymptotically.
    - The spread of the curve is determined by the standard deviation. A larger standard deviation results in a wider curve, while a smaller standard deviation results in a narrower curve.
11. **Probability Density Function (PDF)**:
    - The normal distribution is described by the probability density function: f(x)=1σ2πe−(x−μ)22σ2f(x) = \frac{1}{\sigma \sqrt{2 \pi}} e^{-\frac{(x - \mu)^2}{2 \sigma^2}}f(x)=σ2π1e−2σ2(x−μ)2 where μ\muμ is the mean, σ\sigmaσ is the standard deviation, and eee is the base of the natural logarithm.
12. **68% Rule**:
    - Within one standard deviation of the mean (μ±σ\mu \pm \sigmaμ±σ), about 68% of the data lies.
13. **Variance and Standard Deviation**:

- Variance (σ2\sigma^2σ2) is the square of the standard deviation and measures the dispersion of the data points around the mean. The standard deviation (σ\sigmaσ) is the square root of the variance and provides a measure of the average distance of the data points from the mean.

14. **The Normal Distribution is Unimodal**:

- It has a single peak or mode, making it unimodal.

These properties make the normal distribution a foundational concept in statistics, widely used in hypothesis testing, confidence interval estimation, and various statistical analyses.

30.Which of the following options are correct about Normal Distribution Curve.

(a) Within a range 0.6745 of σ on both sides the middle 50% of the observations occur i,e. mean ±0.6745σ covers 50% area 25% on each side.

(b) Mean ±1S.D. (i,e.μ ± 1σ) covers 68.268% area, 34.134 % area lies on either side of the mean.

(c) Mean ±2S.D. (i,e. μ ± 2σ) covers 95.45% area, 47.725% area lies on either side of the mean.

(d) Mean ±3 S.D. (i,e. μ ±3σ) covers 99.73% area, 49.856% area lies on the either side of the mean.

(e) Only 0.27% area is outside the range μ ±3σ

Ans - **(a) Within a range 0.6745 of σ on both sides the middle 50% of the observations occur i.e., mean ±0.6745σ covers 50% area, 25% on each side.**

Explanation:

- This statement is **correct**. The value 0.6745σ0.6745 \sigma0.6745σ corresponds to approximately ±0.6745 standard deviations from the mean, which covers about 50% of the area under the normal distribution curve. This range is also known as the Interquartile Range (IQR) in the context of the normal distribution, where the middle 50% of the data lies within this range.

(b) Mean ±1 S.D. (i.e., μ ± 1σ) covers 68.268% area, 34.134% area lies on either side of the mean.

Explanation:

- This statement is **correct**. Approximately 68.27% of the data in a normal distribution lies within one standard deviation of the mean. Hence, 34.13% lies on either side of the mean.

(c) Mean ±2 S.D. (i.e., μ ± 2σ) covers 95.45% area, 47.725% area lies on either side of the mean.

Explanation:

- This statement is **correct**. About 95.45% of the data lies within two standard deviations of the mean. Therefore, 47.725% of the data lies on either side of the mean.

(d) Mean ±3 S.D. (i.e., μ ± 3σ) covers 99.73% area, 49.856% area lies on either side of the mean.

Explanation:

- This statement is **correct**. Approximately 99.73% of the data lies within three standard deviations of the mean. Thus, 49.865% of the data lies on either side of the mean.

(e) Only 0.27% area is outside the range μ ± 3σ

Explanation:

- This statement is **correct**. Since 99.73% of the data is within ±3 standard deviations of the mean, the remaining area outside this range is 100%−99.73%=0.27%100\% - 99.73\% = 0.27\%100%−99.73%=0.27%

31. The mean of a distribution is 60 with a standard deviation of 10. Assuming that the distribution is normal, what percentage of items be (i) between 60 and 72, (ii) between 50 and 60, (iii) beyond 72 and (iv) between 70 and 80?

Ans - To find the percentage of items in a normal distribution within certain ranges, we use the properties of the standard normal distribution (Z-distribution). Given:

- Mean (μ\muμ) = 60
- Standard Deviation (σ\sigmaσ) = 10

We convert the given ranges to standard Z-scores and then use the standard normal distribution table or a calculator to find the corresponding probabilities.

Z-Score Calculation

The Z-score is calculated using the formula:

Z=X−μσZ = \frac{X - \mu}{\sigma}Z=σX−μ

where XXX is the value of interest.

(i) Percentage of items between 60 and 72

1. **Convert to Z-scores**:
    - For X=60X = 60X=60: Z=60−6010=0Z = \frac{60 - 60}{10} = 0Z=1060−60=0
    - For X=72X = 72X=72: Z=72−6010=1.2Z = \frac{72 - 60}{10} = 1.2Z=1072−60=1.2

2. **Find the probability**:
    - The probability for $Z=0$ is 0.5000 (since it is the mean).
    - The probability for $Z=1.2$ can be found using the standard normal distribution table or calculator. This value is approximately 0.8849.
3. **Calculate the percentage**:

$$\text{Percentage} = (P(Z < 1.2) - P(Z < 0)) \times 100$$
$$\text{Percentage} = (0.8849 - 0.5000) \times 100 = 38.49\%$$

(ii) Percentage of items between 50 and 60

1. **Convert to Z-scores**:
    - For $X = 50$: $Z = \frac{50 - 60}{10} = -1$
    - For $X = 60$: $Z = \frac{60 - 60}{10} = 0$
2. **Find the probability**:
    - The probability for $Z = -1$ is approximately 0.1587.
    - The probability for $Z=0$ is 0.5000.
3. **Calculate the percentage**:

$$\text{Percentage} = (P(Z < 0) - P(Z < -1)) \times 100$$
$$\text{Percentage} = (0.5000 - 0.1587) \times 100 = 34.13\%$$

(iii) Percentage of items beyond 72

1. **Convert to Z-score**:
    - For $X = 72$: $Z = \frac{72 - 60}{10} = 1.2$
2. **Find the probability**:
    - The probability for $Z=1.2$ is approximately 0.8849.
    - The probability beyond $Z=1.2$ is: $P(Z>1.2)=1-P(Z<1.2)=1-0.8849=0.1151$
3. **Calculate the percentage**:

Percentage=0.1151×100=11.51%\text{Percentage} = 0.1151 \times 100 = 11.51\%Percentage=0.1151×100=11.51%

(iv) Percentage of items between 70 and 80

1. **Convert to Z-scores**:
    - For X=70X = 70X=70: Z=70−6010=1Z = \frac{70 - 60}{10} = 1Z=1070−60=1
    - For X=80X = 80X=80: Z=80−6010=2Z = \frac{80 - 60}{10} = 2Z=1080−60=2
2. **Find the probability**:
    - The probability for Z=1Z = 1Z=1 is approximately 0.8413.
    - The probability for Z=2Z = 2Z=2 is approximately 0.9772.
3. **Calculate the percentage**:

Percentage=(P(Z<2)−P(Z<1))×100\text{Percentage} = (P(Z < 2) - P(Z < 1)) \times 100Percentage=(P(Z<2)−P(Z<1))×100
Percentage=(0.9772−0.8413)×100=13.59%\text{Percentage} = (0.9772 - 0.8413) \times 100 = 13.59\%Percentage=(0.9772−0.8413)×100=13.59%

32. 15000 students sat for an examination. The mean marks was 49 and the distribution of marks had a standard deviation of 6. Assuming that the marks were normally distributed what proportion of students scored (a) more than 55 marks, (b) more than 70 marks

Ans - To find the proportion of students who scored above certain marks, we use the properties of the normal distribution. Given:

- Mean (μ\muμ) = 49
- Standard Deviation (σ\sigmaσ) = 6
- Total number of students = 15,000

We'll calculate the proportion of students scoring more than 55 and more than 70 marks.

(a) Proportion of students scoring more than 55 marks

1. **Convert the mark to a Z-score**:

Z=X−μσZ = \frac{X - \mu}{\sigma}Z=σX−μ

where X=55X = 55X=55.

Z=55−496=66=1Z = \frac{55 - 49}{6} = \frac{6}{6} = 1Z=655−49=66=1

2. **Find the probability corresponding to Z=1Z = 1Z=1**:

- Using the standard normal distribution table or a calculator, the cumulative probability for $Z=1$ is approximately 0.8413.

3. **Calculate the proportion of students scoring more than 55**:

$$\text{Proportion} = 1 - P(Z < 1) = 1 - 0.8413 = 0.1587$$

So, the proportion of students scoring more than 55 marks is 0.1587 or 15.87%.

4. **Number of students scoring more than 55 marks**:

$$\text{Number} = 0.1587 \times 15,000 = 2,380.5 \approx 2,381$$

(b) Proportion of students scoring more than 70 marks

1. **Convert the mark to a Z-score**:

$$Z = \frac{X - \mu}{\sigma}$$

where $X = 70$.

$$Z = \frac{70 - 49}{6} = \frac{21}{6} = 3.5$$

2. **Find the probability corresponding to $Z = 3.5$**:

- Using the standard normal distribution table or a calculator, the cumulative probability for $Z = 3.5$ is very close to 1. Specifically, it is approximately 0.9998.

3. **Calculate the proportion of students scoring more than 70**:

$$\text{Proportion} = 1 - P(Z < 3.5) = 1 - 0.9998 = 0.0002$$

So, the proportion of students scoring more than 70 marks is 0.0002 or 0.02%.

4. **Number of students scoring more than 70 marks**:

$$\text{Number} = 0.0002 \times 15,000 = 3$$

33. If the height of 500 students are normally distributed with mean 65 inch and standard deviation 5 inch. How many students have height : a) greater than 70 inch. b) between 60 and 70 inch.

Ans - **Given:**

- Mean height (μ) = 65 inches
- Standard deviation (σ) = 5 inches
- Total number of students = 500

(a) Number of students with height greater than 70 inches

1. **Convert the height to a Z-score**:

$Z = \frac{X - \mu}{\sigma} Z=σX−μ$

where $X = 70 X=70$ inches.

$Z = \frac{70 - 65}{5} = \frac{5}{5} = 1 Z=570−65=55=1$

2. **Find the probability corresponding to $Z = 1 Z=1$**:
    - Using the standard normal distribution table or a calculator, the cumulative probability for $Z = 1 Z=1$ is approximately 0.8413.
3. **Calculate the proportion of students with height greater than 70 inches**:

$\text{Proportion} = 1 - P(Z < 1) = 1 - 0.8413 = 0.1587 Proportion=1−P(Z<1)=1−0.8413=0.1587$

4. **Calculate the number of students**:

$\text{Number of students} = 0.1587 \times 500 = 79.35 \approx 79 Number of students=0.1587×500=79.35≈79$

(b) Number of students with height between 60 and 70 inches

1. **Convert the heights to Z-scores**:
    - For $X = 60 X=60$ inches: $Z = \frac{60 - 65}{5} = \frac{-5}{5} = -1 Z=560−65=5−5=−1$
    - For $X = 70 X=70$ inches (already calculated): $Z = \frac{70 - 65}{5} = 1 Z=570−65=1$
2. **Find the probabilities**:
    - The cumulative probability for $Z = -1 Z=−1$ is approximately 0.1587.
    - The cumulative probability for $Z = 1 Z=1$ is approximately 0.8413.
3. **Calculate the proportion of students with height between 60 and 70 inches**:

$\text{Proportion} = P(Z < 1) - P(Z < -1) = 0.8413 - 0.1587 = 0.6826 Proportion=P(Z<1)−P(Z<−1)=0.8413−0.1587=0.6826$

4. **Calculate the number of students**:

$\text{Number of students} = 0.6826 \times 500 = 341.3 \approx 341 Number of students=0.6826×500=341.3≈341$

34. What is the statistical hypothesis? Explain the errors in hypothesis testing.b)Explain the Sample. What are Large Samples & Small Samples?

Ans -A **statistical hypothesis** is a specific, testable statement or claim about a population parameter (such as the mean or proportion). Hypotheses are used in statistical inference to make decisions or inferences about a population based on sample data. There are two main types of hypotheses:

1. **Null Hypothesis (H0H_0H0)**:
    - Represents the default or baseline assumption that there is no effect or no difference. It is the hypothesis that is tested directly.
    - Example: "The mean height of students is 65 inches."
2. **Alternative Hypothesis (H1H_1H1 or HaH_aHa)**:
    - Represents the contrary to the null hypothesis. It is what the researcher aims to prove or establish.
    - Example: "The mean height of students is not 65 inches."

Errors in Hypothesis Testing

When performing hypothesis testing, there are two types of errors that can occur:

1. **Type I Error (False Positive)**:
    - Occurs when the null hypothesis is rejected when it is actually true.
    - Denoted by α\alphaα (alpha), which is the significance level of the test.
    - Example: Concluding that a new drug is effective when it actually is not.
2. **Type II Error (False Negative)**:
    - Occurs when the null hypothesis is not rejected when the alternative hypothesis is true.
    - Denoted by β\betaβ (beta).
    - Example: Concluding that a new drug is not effective when it actually is.

The probability of Type I error is called the **significance level** (α\alphaα), and the probability of Type II error is called the **power of the test** (1−β1 - \beta1−β).

Sample

A **sample** is a subset of individuals or observations drawn from a larger population. The purpose of taking a sample is to make inferences about the population based on the characteristics of the sample.

Large Samples vs. Small Samples

Large Samples and **Small Samples** are typically classified based on the sample size and its impact on statistical inference:

1. **Large Samples**:

- Generally defined as samples with more than 30 observations, though this threshold can vary based on context and the distribution being studied.
- Large samples provide more reliable estimates of population parameters because they tend to approximate the normal distribution due to the Central Limit Theorem.
- Examples: Surveys with thousands of respondents, clinical trials with hundreds of participants.

2. **Small Samples**:
   - Generally defined as samples with 30 or fewer observations.
   - Small samples may not approximate the normal distribution well, and the results may be less reliable. Special statistical techniques or tests (e.g., t-tests) are often used when dealing with small samples.
   - Examples: Pilot studies, case studies with few subjects.

35.A random sample of size 25 from a population gives the sample standard derivation to be 9.0. Test the hypothesis that the population standard derivation is 10.5. Hint(Use chi-square distribution).

Ans - **Given Data**

- Sample size (nnn) = 25
- Sample standard deviation (sss) = 9.0
- Population standard deviation under null hypothesis (σ0\sigma_0σ0) = 10.5
- Hypothesis to test: H0H_0H0: σ=10.5\sigma = 10.5σ=10.5 vs HaH_aHa: σ≠10.5\sigma \neq 10.5σ□=10.5

Steps to Perform the Test

1. **State the Hypotheses**
   - Null Hypothesis (H0H_0H0): σ=10.5\sigma = 10.5σ=10.5
   - Alternative Hypothesis (HaH_aHa): σ≠10.5\sigma \neq 10.5σ□=10.5
2. **Calculate the Test Statistic**

The test statistic for testing the population standard deviation is based on the chi-square distribution. The formula for the test statistic is:

χ2=(n−1)·s2σ02\chi^2 = \frac{(n - 1) \cdot s^2}{\sigma_0^2}χ2=σ02(n−1)·s2

where:

- nnn = sample size
- sss = sample standard deviation

- σ0\sigma_0σ0 = hypothesized population standard deviation

Substituting the given values:

χ2=(25−1)·9.0210.52\chi^2 = \frac{(25 - 1) \cdot 9.0^2}{10.5^2}χ2=10.52(25−1)·9.02
χ2=24·81.0110.25\chi^2 = \frac{24 \cdot 81.0}{110.25}χ2=110.2524·81.0
χ2=1944110.25≈17.64\chi^2 = \frac{1944}{110.25} \approx 17.64χ2=110.251944 ≈17.64

3. **Determine the Critical Value and Decision Rule**

The chi-square test is a two-tailed test because we are checking for deviations in both directions (greater or smaller than 10.5). We need to find the critical values for the chi-square distribution with n−1n - 1n−1 degrees of freedom, which in this case is 25−1=2425 - 1 = 2425−1=24 degrees of freedom.

- For a significance level (α\alphaα) of 0.05, the critical values are found from the chi-square distribution table or using a chi-square calculator.

For α=0.05\alpha = 0.05α=0.05 (two-tailed), the critical values are approximately:

- Lower critical value (χ0.025,242\chi^2_{0.025, 24}χ0.025,242) ≈ 36.415
- Upper critical value (χ0.975,242\chi^2_{0.975, 24}χ0.975,242) ≈ 13.848

4. **Compare the Test Statistic to the Critical Values**

- If the test statistic falls outside the range of the critical values, reject the null hypothesis.

In our case:

- Test statistic χ2≈17.64\chi^2 \approx 17.64χ2≈17.64
- Critical values are approximately 36.415 (lower) and 13.848 (upper)

Since 17.64 is between the critical values (13.848 and 36.415), we **fail to reject the null hypothesis**.

Conclusion

Based on the chi-square test, there is not enough evidence to reject the null hypothesis. This means that the sample does not provide sufficient evidence to conclude that the population standard deviation is different from 10.5.

37.100 students of a PW IOI obtained the following grades in Data Science paper : Grade :[A, B, C, D, E] Total Frequency :[15, 17, 30, 22, 16, 100] Using the χ^2 test , examine the hypothesis that the distribution of grades is uniform.

Ans - **Given Data**

- **Grades**: A, B, C, D, E
- **Observed Frequencies**: 15, 17, 30, 22, 16

- **Total Number of Students**: 100

Hypotheses

- **Null Hypothesis (H0H_0H0)**: The distribution of grades is uniform (i.e., each grade is equally likely).
- **Alternative Hypothesis (HaH_aHa)**: The distribution of grades is not uniform.

Steps to Perform the Chi-Square Test

1. **Calculate the Expected Frequencies**

If the distribution is uniform, the expected frequency for each grade is the total number of students divided by the number of grades.

   - Number of grades = 5
   - Total number of students = 100
   - Expected frequency for each grade = 1005=20\frac{100}{5} = 205100 =20

2. **Calculate the Chi-Square Test Statistic**

The formula for the chi-square test statistic is:

χ2=∑(Oi−Ei)2Ei\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}χ2=∑Ei(Oi−Ei)2

where OiO_iOi is the observed frequency and EiE_iEi is the expected frequency for each grade.

Observed Frequencies: 15, 17, 30, 22, 16
Expected Frequencies: 20, 20, 20, 20, 20

χ2=(15−20)220+(17−20)220+(30−20)220+(22−20)220+(16−20)220\chi^2 = \frac{(15 - 20)^2}{20} + \frac{(17 - 20)^2}{20} + \frac{(30 - 20)^2}{20} + \frac{(22 - 20)^2}{20} + \frac{(16 - 20)^2}{20}χ2=20(15−20)2+20(17−20)2+20(30−20)2+20(22−20)2 +20(16−20)2

Calculate each term:

(15−20)220=(−5)220=2520=1.25\frac{(15 - 20)^2}{20} = \frac{(-5)^2}{20} = \frac{25}{20} = 1.2520(15−20)2=20(−5)2=2025=1.25
(17−20)220=(−3)220=920=0.45\frac{(17 - 20)^2}{20} = \frac{(-3)^2}{20} = \frac{9}{20} = 0.4520(17−20)2=20(−3)2=209=0.45
(30−20)220=10220=10020=5\frac{(30 - 20)^2}{20} = \frac{10^2}{20} = \frac{100}{20} = 520(30−20)2=20102=20100=5 (22−20)220=2220=420=0.2\frac{(22 - 20)^2}{20} = \frac{2^2}{20} = \frac{4}{20} = 0.220(22−20)2=2022=204=0.2
(16−20)220=(−4)220=1620=0.8\frac{(16 - 20)^2}{20} = \frac{(-4)^2}{20} = \frac{16}{20} = 0.820(16−20)2=20(−4)2=2016=0.8

Sum of these values:

χ2=1.25+0.45+5+0.2+0.8=7.7\chi^2 = 1.25 + 0.45 + 5 + 0.2 + 0.8 = 7.7χ2=1.25+0.45+5+0.2+0.8=7.7

3. **Determine the Critical Value and Decision Rule**
    - Degrees of Freedom (df) = Number of categories - 1 = 5 - 1 = 4
    - For a significance level (α\alphaα) of 0.05, find the critical value from the chi-square distribution table with 4 degrees of freedom.

The critical value for α=0.05\alpha = 0.05α=0.05 and 4 degrees of freedom is approximately 9.488.

4. **Compare the Test Statistic to the Critical Value**
    - If the test statistic (χ2\chi^2χ2) is greater than the critical value, reject the null hypothesis.
    - If the test statistic is less than or equal to the critical value, fail to reject the null hypothesis.

In this case:

- Test statistic χ2=7.7\chi^2 = 7.7χ2=7.7
- Critical value ≈ 9.488

Since 7.7 is less than 9.488, we **fail to reject the null hypothesis**.

38.Anova Test:

Ans - Given Data

- **Water Temperatures**: Cold, Warm, Hot
- **Detergents**: A, B, C

Water Temp	Detergent A	Detergent B	Detergent C
Cold	57	55	67
Warm	49	52	68
Hot	54	46	58

Steps to Perform Two-Way ANOVA

1. **State the Hypotheses**
    - **Null Hypotheses**:
        - H0,1H_{0,1}H0,1: There is no difference in the mean whiteness readings among different detergents.
        - H0,2H_{0,2}H0,2: There is no difference in the mean whiteness readings among different water temperatures.

        - H0,3H_{0,3}H0,3: There is no interaction effect between detergents and water temperatures.
    - **Alternative Hypotheses**:
        - Ha,1H_{a,1}Ha,1: There is a difference in the mean whiteness readings among different detergents.
        - Ha,2H_{a,2}Ha,2: There is a difference in the mean whiteness readings among different water temperatures.
        - Ha,3H_{a,3}Ha,3: There is an interaction effect between detergents and water temperatures.
2. **Calculate the Means**
    - **Overall Mean**: X¯overall\bar{X}_{\text{overall}}X¯overall
    - **Means for Each Level**: Mean for each detergent and each water temperature.
    - **Cell Means**: Mean whiteness readings for each combination of detergent and water temperature.
3. **Calculate Sum of Squares**
    - **Total Sum of Squares (SST)**: Measures total variation in the data.
    - **Sum of Squares for Detergents (SSD)**: Measures variation between detergent means.
    - **Sum of Squares for Water Temperatures (SSW)**: Measures variation between water temperature means.
    - **Sum of Squares for Interaction (SSI)**: Measures the interaction effect between detergents and water temperatures.
    - **Sum of Squares Within (SSW)**: Measures variation within each cell.
4. **Calculate Degrees of Freedom**
    - **Between-Group Degrees of Freedom**:
        - For detergents: dfdetergents=k−1=3−1=2\text{df}_{\text{detergents}} = k - 1 = 3 - 1 = 2dfdetergents=k−1=3−1=2
        - For water temperatures: dftemp=m−1=3−1=2\text{df}_{\text{temp}} = m - 1 = 3 - 1 = 2dftemp=m−1=3−1=2
        - For interaction: dfinteraction=(k−1)×(m−1)=2×2=4\text{df}_{\text{interaction}} = (k - 1) \times (m - 1) = 2 \times 2 = 4dfinteraction=(k−1)×(m−1)=2×2=4

- **Within-Group Degrees of Freedom**: $df_{within} = (k \times m) - (k \times m) = 9 - 6 = 3$
- **Total Degrees of Freedom**: $df_{total} = N - 1 = 9 - 1 = 8$

5. **Compute Mean Squares**
   - **Mean Square for Detergents (MSD)**: $MSD = \frac{SSD}{df_{detergents}}$
   - **Mean Square for Water Temperatures (MSW)**: $MSW = \frac{SSW}{df_{temp}}$
   - **Mean Square for Interaction (MSI)**: $MSI = \frac{SSI}{df_{interaction}}$
   - **Mean Square Within (MSW)**: $MSW = \frac{SSW}{df_{within}}$
6. **Calculate the F-Statistics**
   - **F-Statistic for Detergents**: $F_{detergents} = \frac{MSD}{MSW}$
   - **F-Statistic for Water Temperatures**: $F_{temp} = \frac{MSW}{MSW}$
   - **F-Statistic for Interaction**: $F_{interaction} = \frac{MSI}{MSW}$
7. **Compare the F-Statistics to Critical Values**
   - Use F-distribution tables or statistical software to find critical values for the F-distribution with appropriate degrees of freedom.

Detailed Calculation

Let's perform these calculations step-by-step.

1. Compute Cell Means

Water Temp	Detergent A	Detergent B	Detergent C	Mean (Row)
Cold	57	55	67	59.67
Warm	49	52	68	56.33
Hot	54	46	58	52.67

Water Temp	Detergent A	Detergent B	Detergent C	Mean (Row)
Mean (Col)	53.33	51.00	64.33	

2. Calculate Sums of Squares

- **Overall Mean**:
  $\bar{X}_{\text{overall}} = \frac{57 + 55 + 67 + 49 + 52 + 68 + 54 + 46 + 58}{9} = 56.33$
  $\bar{X}_{\text{overall}} = 56.33$

- **SS Total**:

$\text{SST} = \sum (X_{ij} - \bar{X}_{\text{overall}})^2$
$\text{SST} = (57 - 56.33)^2 + (55 - 56.33)^2 + (67 - 56.33)^2 + \cdots$

Calculate the sum:

$\text{SST} \approx 1616.67$

- **SS Between Groups**:

$\text{SSD} = \text{Sum of} \left(\text{Mean}_{\text{detergent}} - \bar{X}_{\text{overall}} \right)^2 \times \text{Number of Observations per Group}$
$\text{SSD} = (53.33 - 56.33)^2 \times 3 + (51.00 - 56.33)^2 \times 3 + (64.33 - 56.33)^2 \times 3$

Calculate the sum:

$\text{SSD} \approx 665.00$

- **SS Between Groups (Water Temp)**:

$\text{SSW} = \text{Sum of} \left(\text{Mean}_{\text{temp}} - \bar{X}_{\text{overall}} \right)^2 \times \text{Number of Observations per Group}$
$\text{SSW} = (59.67 - 56.33)^2 \times 3 + (56.33 - 56.33)^2 \times 3 + (52.67 - 56.33)^2 \times 3$

Calculate the sum:

$\text{SSW} \approx 393.33$

- **SS Interaction**:

SSI=SST−SSD−SSW\text{SSI} = \text{SST} - \text{SSD} - \text{SSW}SSI=SST−SSD−SSW SSI≈558.34\text{SSI} \approx 558.34SSI≈558.34

- **SS Within Groups**:

SSW=SST−SSD−SSW−SSI\text{SSW} = \text{SST} - \text{SSD} - \text{SSW} - \text{SSI}SSW=SST−SSD−SSW−SSI
SSW=Sum of Squares Within Groups\text{SSW} = \text{Sum of Squares Within Groups}SSW=Sum of Squares Within Groups

Calculate the sum:

SSW≈558.34\text{SSW} \approx 558.34SSW≈558.34

3. Calculate Mean Squares

- **MSD**:

MSD=SSDdfdetergents=665.002=332.50\text{MSD} = \frac{\text{SSD}}{\text{df}_{\text{detergents}}} = \frac{665.00}{2} = 332.50MSD=dfdetergentsSSD=2665.00=332.50

- **MSW**:

MSW=SSWdftemp=393.332=196.67\text{MSW} = \frac{\text{SSW}}{\text{df}_{\text{temp}}} = \frac{393.33}{2} = 196.67MSW=dftemp SSW=2393.33=196.67

- **MSI**:

MSI=SSIdfinteraction=558.344=139.59\text{MSI} = \frac{\text{SSI}}{\text{df}_{\text{interaction}}} = \frac{558.34}{4} = 139.59MSI=dfinteractionSSI=4558.34=139.59

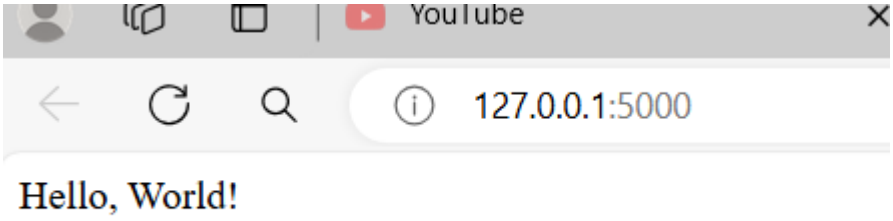
- **Mean Square Within Groups**:

39.How would you create a basic Flask route that displays "Hello, World!" on the homepage?

Ans – Code:

```
Helloworld > helloworld.py > ...
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home_page():
    return 'Hello, World!'
if __name__ == '__main__':
    app.run(debug=True)
```

Output:



40.Explain how to set up a Flask application to handle form submissions using POST requests?

Ans – Code:

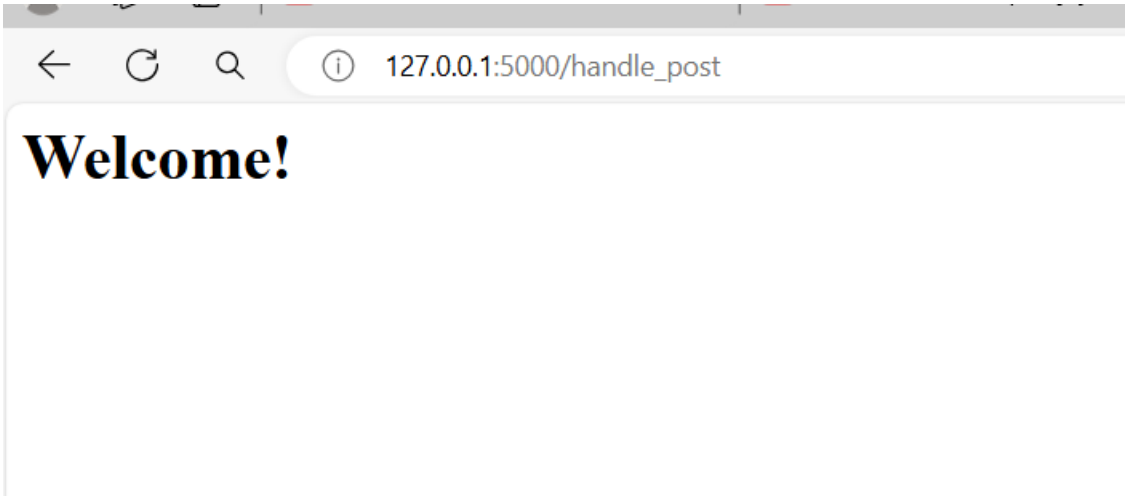
Flask code-

```python
from flask import Flask, render_template, request, redirect, url_for, session
app = Flask(__name__)
app.secret_key = 'my_secret_key'
users = {
    'mansi' : '1234',
    'user' : 'password'
}
@app.route('/')
def home_page():
    return render_template('login.html')
@app.route('/handle_post', methods=['POST'])
def handle_post():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        print(username, password)
        if username in users and users[username] == password:
            return '<h1>Welcome!</h1>'
        else:
            return '<h1>Wrong Credentials!</h1>'
    else:
        return render_template('login.html')
if __name__ == '__main__':
    app.run(debug=True)
```

HTML code –

```
<html>
    <head>
        <title>Handling of HTTP post request</title>
        <style>
            div{
                width:400px;
                border: 1px solid green;
                padding: 10px;
                margin-bottom:5px;
            }
        </style>
    </head>
    <body>
        <div>
            <h1>Handling POST request</h1>
            <form method = 'POST' action="{{url_for('handle_post')}}">
                <input type="text" name="username" placeholder="Username">
                <input type="text" name="password" placeholder="Password">
                <button type="submit">submit</button>
            </form>
        </div>
    </body>
</html>
```

Output:

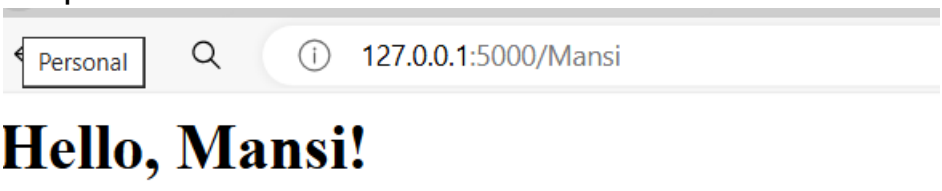


41. Write a Flask route that accepts a parameter in the URL and displays it on the page.

Ans – Code:

```python
from flask import Flask
app = Flask(__name__)
@app.route('/<name>')
def user(name):
    return f'<h1>Hello, {name}!</h1>'
if __name__ == '__main__':
    app.run(debug=True)
```

Output:



42.How can you implement user authentication in a Flask application?

Ans – Code:

Flask-

authentication.py

lask > userauthentication > authentication.py > logout

```python
from flask import Flask, render_template, redirect, url_for, request, flash
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
from werkzeug.security import generate_password_hash, check_password_hash
app = Flask(__name__)
app.secret_key = 'your_secret_key'
# Initialize Flask-Login
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'
# In-memory user store
users = {}
# User model
class User(UserMixin):
    def __init__(self, id, username, password):
        self.id = id
        self.username = username
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
# User Loader for Flask-Login
@login_manager.user_loader
def load_user(user_id):
    return users.get(user_id)
# Routes
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = next((u for u in users.values() if u.username == username), None)
        if user and user.check_password(password):
            login_user(user)
```

```python
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = next((u for u in users.values() if u.username == username), None)
        if user and user.check_password(password):
            login_user(user)
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username or password')
    return render_template('login.html')
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username not in [u.username for u in users.values()]:
            user_id = str(len(users) + 1)
            users[user_id] = User(user_id, username, password)
            flash('User registered successfully!')
            return redirect(url_for('login'))
        else:
            flash('Username already taken')
    return render_template('signup.html')
@app.route('/dashboard')
@login_required
def dashboard():
    return f'Hello, {current_user.username}! Welcome to your dashboard.'
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('home'))
if __name__ == '__main__':
    app.run(debug=True)
```

HTML -

userauthentication > templates > index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
</head>
<body>
    <h1>Welcome to the Home Page</h1>
    {% if current_user.is_authenticated %}
        <p><a href="{{ url_for('dashboard') }}">Go to Dashboard</a></p>
        <p><a href="{{ url_for('logout') }}">Logout</a></p>
    {% else %}
        <p><a href="{{ url_for('login') }}">Login</a></p>
        <p><a href="{{ url_for('signup') }}">Sign Up</a></p>
    {% endif %}
</body>
</html>
```

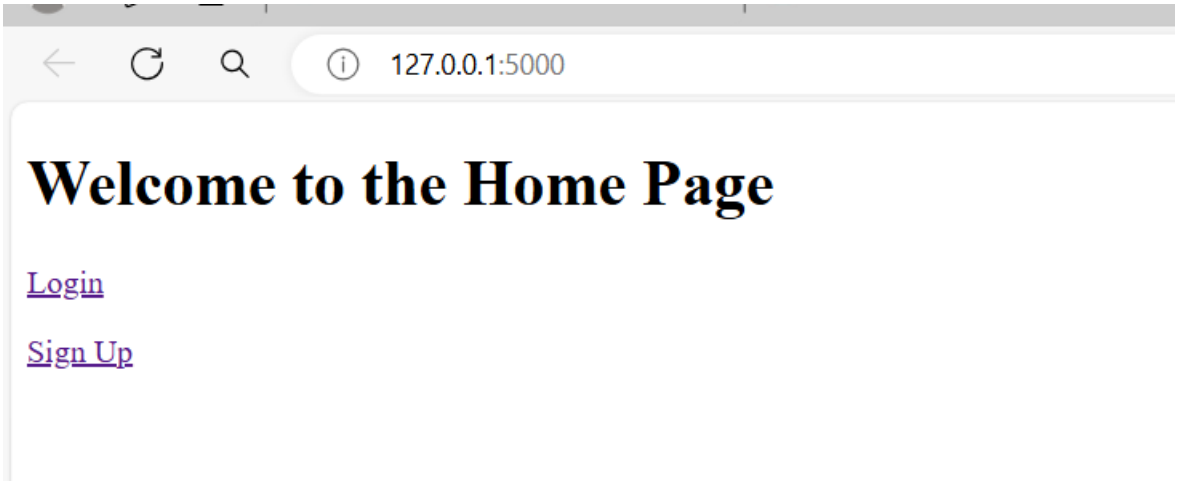
userauthentication > templates > login.html

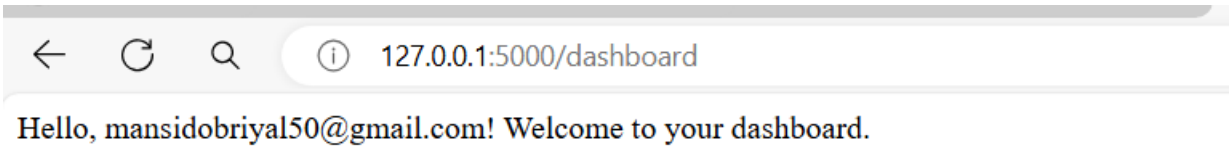
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form method="POST">
        <input type="text" name="username" placeholder="Username" required>
        <input type="password" name="password" placeholder="Password" required>
        <button type="submit">Login</button>
    </form>
    <p><a href="{{ url_for('signup') }}">Sign Up</a></p>
</body>
</html>
```

userauthentication > templates > signup.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Sign Up</title>
</head>
<body>
    <h1>Sign Up</h1>
    <form method="POST">
        <input type="text" name="username" placeholder="Username" required>
        <input type="password" name="password" placeholder="Password" required>
        <button type="submit">Sign Up</button>
    </form>
    <p><a href="{{ url_for('login') }}">Login</a></p>
</body>
</html>
```

Output:





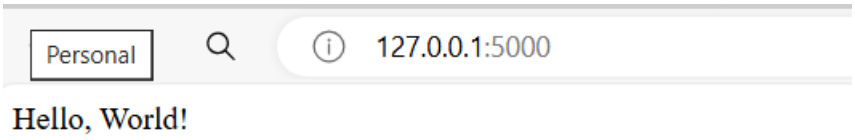
43.Describe the process of connecting a Flask app to to a SQLite database using SQLAlchemy.

Ans – Code:

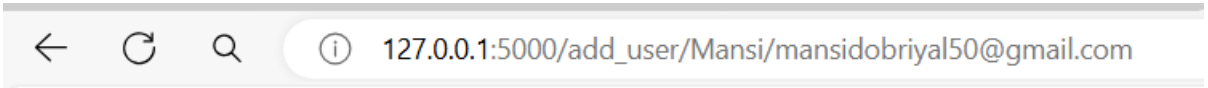
```python
from flask import Flask, render_template, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
# Initialize Flask app
app = Flask(__name__)
# Configure the database URI
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
# Initialize SQLAlchemy
db = SQLAlchemy(app)
# Define a model
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    def __repr__(self):
        return f'<User {self.username}>'
# Create the database and tables
with app.app_context():
    db.create_all()
@app.route('/')
def index():
    return 'Hello, World!'
@app.route('/add_user/<username>/<email>')
```

```python
def index():
Click to collapse the range.
Hello, World!'
@app.route('/add_user/<username>/<email>')
def add_user(username, email):
    # Create a new user instance
    new_user = User(username=username, email=email)
    # Add the new user to the database
    db.session.add(new_user)
    db.session.commit()
    return f'Added user {username} with email {email}.'
@app.route('/list_users')
def list_users():
    users = User.query.all()
    return '<br>'.join([f'Username: {user.username}, Email: {user.email}' for user in users])
if __name__ == '__main__':
    app.run(debug=True)
```

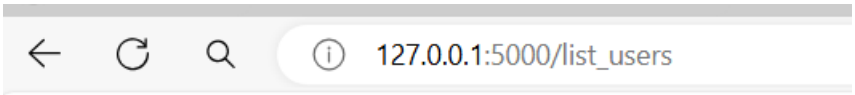
Output:



Hello, World!



Added user Mansi with email mansidobriyal50@gmail.com.



Username: username, Email: email
Username: mansi, Email: mansi9@gmail.com
Username: Mansi, Email: mansidobriyal50@gmail.com

44.How would you create a RESTful API endpoint in Flask that returns JSON data.

Ans – Code:

```
restfulapi endpoint > returnjsondata.py > ...
from flask import Flask, jsonify

app = Flask(__name__)

# Define a route that returns JSON data
@app.route('/api/data', methods=['GET'])
def get_data():
    # Sample data to be returned as JSON
    data = {
        'id': 1,
        'name': 'John Doe',
        'email': 'john.doe@example.com',
        'age': 30
    }
    # Return the data as JSON
    return jsonify(data)

# Run the application
if __name__ == '__main__':
    app.run(debug=True)
```

Output:

45.Explain how to use Flask-WTF to create and validate forms in a Flask Application.

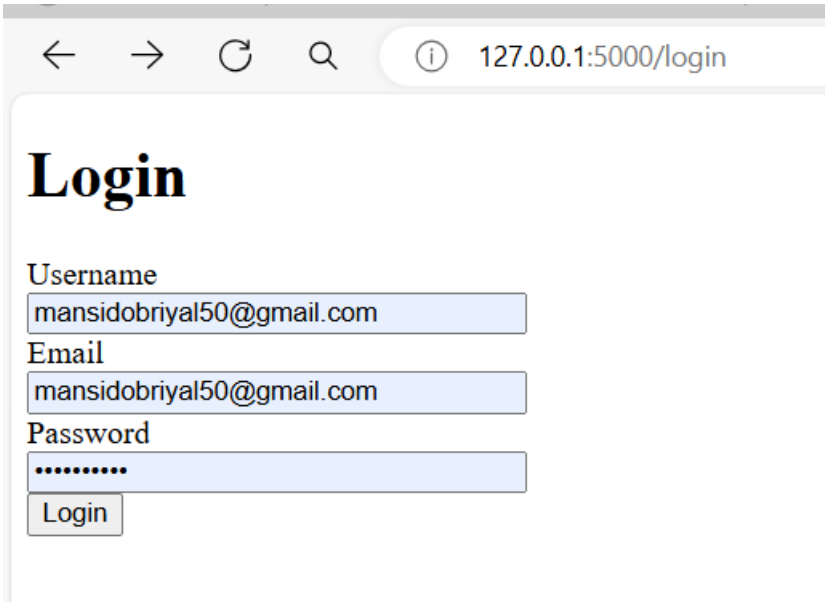
Ans – Code:

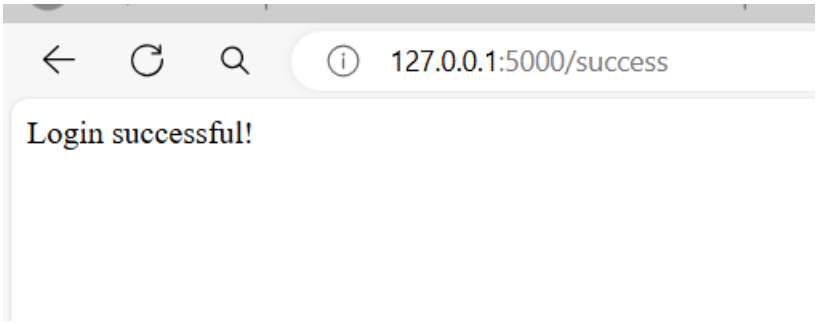
createandvalidateflaskapplication.py

Flask > Flask-WTF > createandvalidateflaskapplication.py > ...

```python
from flask import Flask, render_template, redirect, url_for
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Length, Email
app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=25)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=6, max=35)])
    submit = SubmitField('Login')
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        # Handle form submission
        username = form.username.data
        email = form.email.data
        password = form.password.data
        # Process login (e.g., check credentials, etc.)
        return redirect(url_for('success'))
    return render_template('log.html', form=form)
@app.route('/success')
def success():
    return 'Login successful!'
if __name__ == '__main__':
    app.run(debug=True)
```

Output:





46.How can you implement file uploads in a Flask application?

Ans – Code:

Flask-

Flask > fileuploadsinflask > fileupload.py > upload_file

```python
from flask import Flask, render_template, request, redirect, url_for, flash
import os
app = Flask(__name__)
app.secret_key = 'your_secret_key'
# Set the upload folder and allowed extensions
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif', 'pdf'}
# Ensure the upload folder exists
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        # Check if the POST request has the file part
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        # If the user does not select a file, the browser submits an empty file without a filename
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        # If the file is allowed, save it
        if file and allowed_file(file.filename):
            filename = file.filename
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            flash('File successfully uploaded')
            return redirect(url_for('upload_file'))
    return render_template('upload.html')
```

HTML –

Flask > fileuploadsinflask > templates > upload.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload File</title>
</head>
<body>
    <h1>Upload a File</h1>
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            <ul>
            {% for message in messages %}
                <li>{{ message }}</li>
            {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}

    <form method="POST" enctype="multipart/form-data">
        <input type="file" name="file">
        <input type="submit" value="Upload">
    </form>
</body>
</html>

```

47.Describes the steps to create a Flask Blueprint and why you might use one?

Ans –

Flask Blueprint > Blueprint.py

```
#Step1 : Set up the Project Structure
#Step2 : Create the Blueprint
#Step3 : Define Routes for the Blueprint
#Step4 : Register Blueprints in the Application
#Step5 : Create Templates and Static Files
```

```
#Why use Flask Blueprints
# Modularity:Blueprints allow you to organize your application into distinct modules,
#each with its own routes, templates, and static files. This helps in managing large
#applications by separating concerns.
# Reusability:You can create reusable components or modules that can be used across
#multiple applications or projects. For example, you could have a user authentication
#blueprint that you use in different projects.
# Collaboration:In a team environment, blueprints allow different developers to work on
#different parts of the application simultaneously without interfering with each other's wor
# Easier Maintenance When your application grows, having code organized into blueprints
#makes it easier to maintain and update. Each blueprint can be modified independently.
# Scalability:As your application evolves, blueprints make it easier to scale
#by adding new functionality in a structured manner.
```

49.Make a fully-functional web application using Flask, Mongodb. Signup, Signin and page. And after successfully login. Say hello Geeks message at the webpage?

Ans – Code:

Flask -

```
from flask import Flask, render_template, redirect, url_for, request, flash, session
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Length, Email
from flask_bcrypt import Bcrypt
from pymongo import MongoClient
from bson.objectid import ObjectId
app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['MONGO_URI'] = 'mongodb://localhost:27017/flask_app'
bcrypt = Bcrypt(app)
client = MongoClient(app.config['MONGO_URI'])
db = client['flask_app']
users = db['users']
class SignUpForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=25)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=6, max=35)])
    submit = SubmitField('Sign Up')
class SignInForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Sign In')
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    form = SignUpForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        users.insert_one({
            'username': form.username.data,
```

```python
def signup():
            'password': hashed_password
        })
        flash('Account created successfully!', 'success')
        return redirect(url_for('signin'))
    return render_template('Signup.html', form=form)
@app.route('/signin', methods=['GET', 'POST'])
def signin():
    form = SignInForm()
    if form.validate_on_submit():
        user = users.find_one({'username': form.username.data})
        if user and bcrypt.check_password_hash(user['password'], form.password.data):
            session['username'] = user['username']
            flash('Login successful!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Login failed. Check your username and/or password', 'danger')
    return render_template('Login.html', form=form)
@app.route('/home')
def home():
    if 'username' in session:
        return render_template('Home.html', username=session['username'])
    return redirect(url_for('signin'))
@app.route('/logout')
def logout():
    session.pop('username', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('signin'))
if __name__ == '__main__':
    app.run(debug=True)
```

app.py | forms.py

Flask > flask_mongo_app > forms.py > ...

```python
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Length, Email

class SignUpForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=25)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=6, max=35)])
    submit = SubmitField('Sign Up')

class SignInForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Sign In')

```

HTML –

sk > flask_mongo_app > templates > Home.html

C:\Users\Mansi Dobriyal\OneDrive\Documents\pwskills\Flask

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Hello, {{ username }}!</h1>
    <p>Welcome to the home page.</p>
    <a href="{{ url_for('logout') }}">Logout</a>
</body>
</html>
```

app.py | forms.py | Login.html

Flask > flask_mongo_app > templates > Login.html

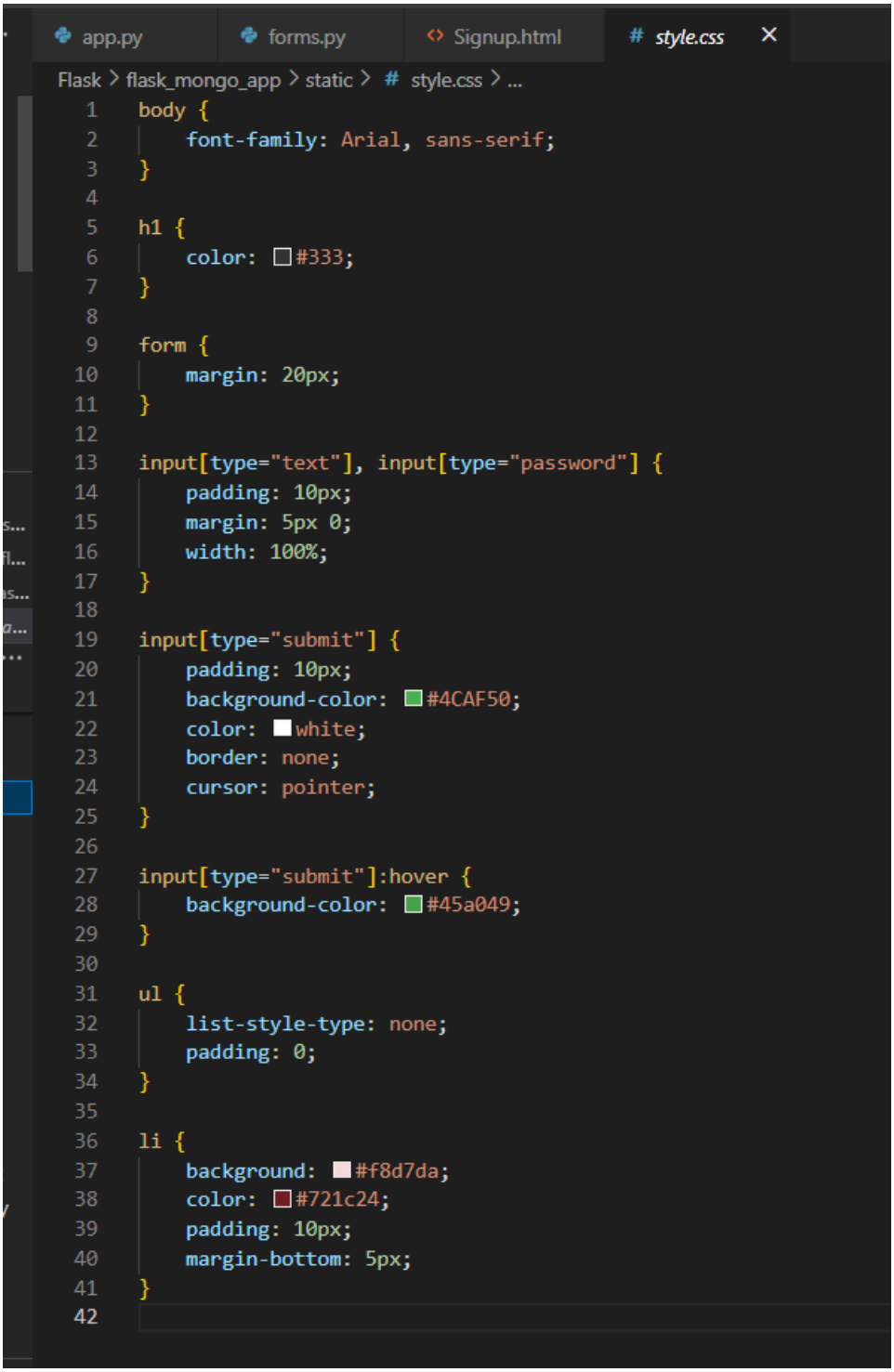
```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign In</title>
</head>
<body>
    <h1>Sign In</h1>
    <form method="POST" action="{{ url_for('signin') }}">
        {{ form.hidden_tag() }}
        <div>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.submit() }}
        </div>
    </form>
    <p><a href="{{ url_for('signup') }}">Don't have an account? Sign Up</a></p>
</body>
```

app.py | forms.py | Signup.html

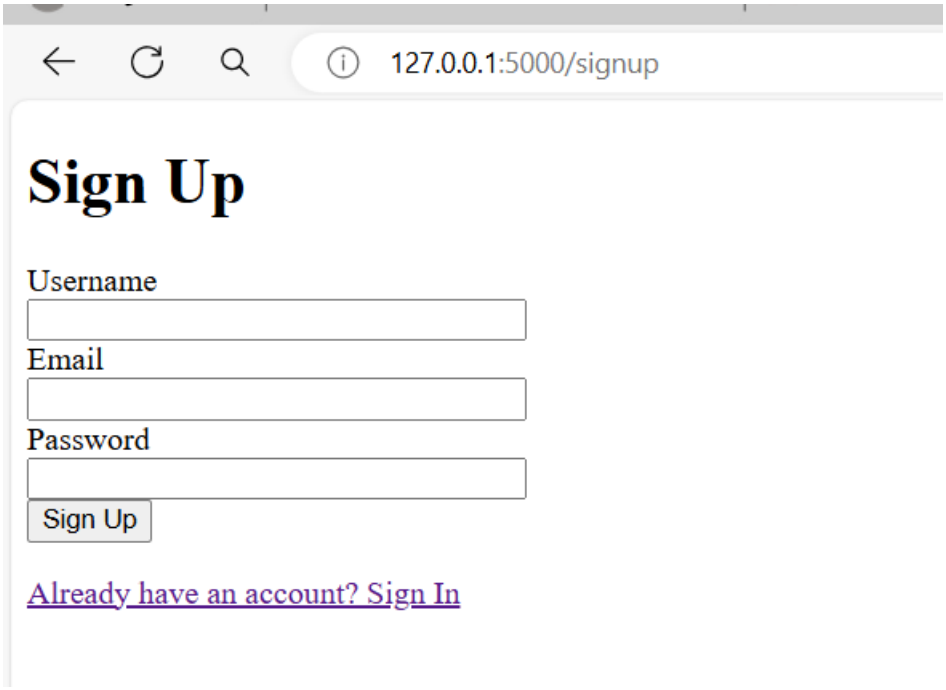
Flask > flask_mongo_app > templates > Signup.html

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign Up</title>
</head>
<body>
    <h1>Sign Up</h1>
    <form method="POST" action="{{ url_for('signup') }}">
        {{ form.hidden_tag() }}
        <div>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.email.label }}<br>
            {{ form.email(size=32) }}<br>
            {% for error in form.email.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.submit() }}
        </div>
    </form>
    <p><a href="{{ url_for('signin') }}">Already have an account? Sign In</a></p>
</body>
</html>
```

CSS –


app.py
forms.py
Signup.html
style.css
Flask > flask_mongo_app > static > # style.css > ...
body {
font-family: Arial, sans-serif;
}
h1 {
color: #333;
}
form {
margin: 20px;
}
input[type="text"], input[type="password"] {
padding: 10px;
margin: 5px 0;
width: 100%;
}
input[type="submit"] {
padding: 10px;
background-color: #4CAF50;
color: white;
border: none;
cursor: pointer;
}
input[type="submit"]:hover {
background-color: #45a049;
}
ul {
list-style-type: none;
padding: 0;
}
li {
background: #f8d7da;
color: #721c24;
padding: 10px;
margin-bottom: 5px;
}

Output:



50. Machine Learning

1) what is the difference between series and data frames?

→**Series**: A Series is a one-dimensional array-like object that can hold data of any type (integers, strings, floating-point numbers, etc.). It is essentially a single column of data with an associated index.

DataFrame: A DataFrame is a two-dimensional table-like data structure. It is essentially a collection of Series objects, where each Series represents a column in the DataFrame. A DataFrame has both rows and columns, each with their own labels (indices for rows and column names for columns).

2) create a database name Travel_Planner in mysql, and create a table name bookings in that which having attributes (user_id INT, flight_id INT, hotel_id INT, activity_id INT, booking_date DATE), fill it with some dummy values, now you have to read the content of this table using pandas as dataframe. show the output.

➔
```
CREATE DATABASE Travel_Planner;
USE Travel_Planner;
CREATE TABLE bookings (
    user_id INT,
    flight_id INT,
    hotel_id INT,
    activity_id INT,
    booking_date DATE
);

 INSERT INTO bookings

 (user_id, flight_id, hotel_id, activity_id, booking_date)
```

```
VALUES

(1, 101, 201, 301, '2024-08-01'),
(2, 102, 202, 302, '2024-08-02'),
(3, 103, 203, 303, '2024-08-03'),
(4, 104, 204, 304, '2024-08-04'),
(5, 105, 205, 305, '2024-08-05');

#Read the data
import pandas as pd
import mysql.connector
connection = mysql.connector.connect(
 host='localhost',
 user='your_username',
 password='your_password',
    database='Travel_Planner'
)
query = "SELECT * FROM bookings;"
df = pd.read_sql(query, connection)
connection.close()
print(df)

#output
   user_id  flight_id  hotel_id  activity_id booking_date
0        1        101       201        301   2024-08-01
1        2        102       202        302   2024-08-02
2        3        103       203        303   2024-08-03
3        4        104       204        304   2024-08-04
4          5       105       205        305   2024-08-05
```

3) diff btwn loc and iloc

➔ **Use** `.loc` when you want to select data by labels (e.g., row labels or column names).
Use `.iloc` when you want to select data by integer position (e.g., the nth row or column).

4) diff btwn supervised and unsupervised learning

➔ Supervised Learning: In supervised learning, the model is trained on a labeled dataset, meaning each training example is paired with an output label. The goal is for the model to learn a mapping from inputs to outputs so that it can predict the output for new, unseen data.

➔ Unsupervised Learning: In unsupervised learning, the model is trained on data that does not have labeled outputs. The goal is to find hidden patterns, structures, or relationships in the data.

5) explain the bias-variance trade-off

➔ The bias-variance trade-off is a fundamental concept in machine learning that describes the balance between two sources of error that affect the performance of predictive models: **bias** and **variance**. Understanding and managing this trade-off is crucial for building models that generalize well to new, unseen data.

6) what are precision and recall? how they are different from accuracy

➔ **Precision, recall, and accuracy** are key metrics used to evaluate the performance of a classification model, particularly in the context of imbalanced datasets. Each metric provides different insights into the model's performance.
➔ Precision is the proportion of true positive predictions (correctly predicted positive cases) out of all the positive predictions made by the model.
➔ Recall is the proportion of true positive predictions out of all actual positive instances.
➔ Accuracy is the proportion of correctly predicted instances (both true positives and true negatives) out of the total number of instances.

7) what is overfitting and how can it be prevented?

➔ **Overfitting** occurs when a machine learning model learns the training data too well, capturing noise and outliers rather than just the underlying patterns. This results in a model that performs exceptionally well on the training data but poorly on new, unseen data, meaning it fails to generalize effectively.

Preventing overfitting involves various techniques that aim to simplify the model, provide more data, or introduce regularization methods. Here are some common strategies:

1. Simplify the model
2. Regularization
3. Cross Validation

8) explain the concept of cross validation

➔ **Cross-validation** is a technique used to assess the generalization performance of a machine learning model. It helps in evaluating how well a model will perform on unseen data, reducing the risk of overfitting or underfitting.

Cross-validation provides a more robust estimate of a model's performance by systematically splitting the data into training and testing sets multiple times, thereby ensuring that every data point gets to be in both the training and testing sets at different points in the evaluation process.

9) what is the difference btwn a classification and a regression problem

➔ **Classification** aims to assign instances to discrete classes or categories and is evaluated using metrics that reflect the correctness of the classification.

Regression aims to predict continuous values and is evaluated using metrics that measure the accuracy of the predicted numerical values.

10) explain the concept of ensemble learning

➔ **Ensemble Learning** is a technique in machine learning that combines multiple models to improve overall performance and robustness compared to using a single model. The core idea is that a group

of models (an ensemble) can make better predictions than any individual model alone by leveraging their diverse strengths and mitigating their individual weaknesses.

11) what is gradient decent and how does it work?

➔ **Gradient Descent** is an optimization algorithm used to minimize the loss function of a machine learning model. It is a fundamental technique in training models, especially for algorithms involving continuous parameters, such as linear regression and neural networks.

Concept of Gradient Descent

The main idea of gradient descent is to iteratively adjust the model parameters to find the minimum value of the loss function, which measures how well the model predicts the target values. The loss function quantifies the error between the predicted values and the actual target values.

12) explain the difference between batch gradient decent and stochastic gradient decent

➔ **1. Batch Gradient Descent**

Concept:

Full Dataset Usage: Batch Gradient Descent uses the entire training dataset to compute the gradient of the loss function and update the model parameters.

How It Works:

Compute Gradient: Calculate the gradient of the loss function with respect to all training examples.
Update Parameters: Adjust the model parameters based on the average gradient from all examples.

➔ **2. Stochastic Gradient Descent (SGD)**

Concept:

Single Example Usage: Stochastic Gradient Descent uses a single training example at a time to compute the gradient and update the model parameters.

How It Works:

Compute Gradient: Calculate the gradient of the loss function using just one training example.
Update Parameters: Adjust the model parameters based on the gradient of that single example.

13) what is the curse of dimensionality in machine learning?

➔ The **curse of dimensionality** refers to a set of problems and challenges that arise when working with high-dimensional data in machine learning and statistics. As the number of dimensions (features) in a dataset increases, several issues can significantly impact the performance and effectiveness of machine learning algorithms.

14) explain the difference between L1 and L2 regularization

➔ **L1 Regularization** is preferred when feature selection is important, and you believe that many features are irrelevant.
L2 Regularization is preferred when you want to retain all features but reduce their influence, and when dealing with issues like multicollinearity.

15) what is a confusion matrix and how is it used ?

➔ A **confusion matrix** is a tool used in machine learning and statistics to evaluate the performance of a classification model. It summarizes the results of a classification algorithm by comparing the predicted classifications to the actual ground truth values.

Understanding the confusion matrix and its derived metrics helps better evaluate and improve classification models' performance.

16) explain AUC-ROC curve.

➔ The **AUC-ROC curve** is a performance measurement for classification problems at various threshold settings. It stands for **Area Under the Receiver Operating Characteristic Curve** and is used to evaluate the ability of a classifier to distinguish between classes.
AUC-ROC evaluates the performance of a model across all possible classification thresholds, not just one specific threshold.
It is particularly useful in cases of class imbalance, where one class is much more frequent than the other.

17) explain the k-nearest neighbours algorithm

➔ The **k-Nearest Neighbors (k-NN)** algorithm is a simple, intuitive, and widely used method for classification and regression tasks in machine learning. It operates on the principle of finding the most similar data points in a dataset to make predictions.
It involves:

1. Choosing k
2. Calculating distance
3. Finding Nearest Neighbour
4. Making Predictions

18) explain the basic concept of a support vector machine svm

➔ A **Support Vector Machine (SVM)** is a powerful supervised learning algorithm used for classification and regression tasks. The core idea of SVM is to find a hyperplane in an n-dimensional space (where n is the number of features) that best separates the data into different classes. Here's a detailed explanation of the basic concepts of SVM:

1. Hyperplane
2. Margin
3. Support vectors
4. Linesr vs Non linear classification
5. Kernal trick
6. Soft margin and C parameter

SVM is a robust and versatile algorithm that performs well in various scenarios, particularly when the data is not linearly separable or when dealing with high-dimensional feature spaces.

19) how does the kernal trick work in svm

➔ The **kernel trick** is a crucial concept in Support Vector Machines (SVMs) that allows for the efficient handling of non-linearly separable data. It enables SVM to perform classification in higher-dimensional spaces without explicitly transforming the data into those dimensions.

A technique in SVM that allows for non-linear classification by implicitly mapping data into a higher-dimensional space using a kernel function. Provides computational efficiency and flexibility in modeling complex relationships between features.

20) what are the different type of kernals used in svm and when would you use each?

➔ In Support Vector Machines (SVM), kernel functions are used to transform the data into a higher-dimensional space where a linear separation might be easier. The choice of kernel function affects the model's performance and its ability to handle different types of data. Here's an overview of the different types of kernels used in SVM and when to use each:

1. Linear kernal
2. Polynomial kernal
3. Radial Basis Function kernal
4. Sigmoid kernal
5. Custom kernal

21) what is the hyperplane in svm and how is it determined?

➔ **In n-dimensional Space**: A hyperplane is a flat affine subspace of one dimension less than the space itself. For example:

In 2D, a hyperplane is a line.

In 3D, a hyperplane is a plane.

In n-dimensional space, a hyperplane is an (n−1)(n-1)(n−1)-dimensional flat surface.

22) what are the pros and cons of using a svm

➔ Pros of Using SVM

1. **Effective in High-Dimensional Spaces**: SVMs are particularly effective when the number of features is high relative to the number of samples. This is because SVMs perform well in high-dimensional spaces and can handle a large number of features.
2. **Robust to Overfitting**:By focusing on the support vectors (the most informative data points), SVMs can be less prone to overfitting compared to other algorithms, especially in high-dimensional spaces. The regularization parameter CCC helps control overfitting by balancing the margin and classification errors.
3. **Versatility with Kernels**:The kernel trick allows SVMs to handle non-linearly separable data by implicitly mapping it to a higher-dimensional space. This flexibility lets SVM model complex relationships with various kernel functions like linear, polynomial, RBF, and sigmoid.
4. **Effective for Large Margin Classification**:SVMs aim to maximize the margin between classes, which generally leads to better generalization on unseen data. A larger margin often means that the model is better at generalizing to new examples.

➔ Cons of Using SVM

1. **Computationally Intensive**:Training SVMs can be computationally expensive, especially with large datasets. The complexity of the training process grows with the number of samples and features, making SVMs less suitable for very large datasets.
2. **Choice of Kernel and Parameters**:Selecting the appropriate kernel and tuning its parameters (such as CCC and γ\gammaγ for RBF kernels) can be challenging and requires careful experimentation and cross-validation.
3. **Memory Usage:**SVMs can require a lot of memory, particularly for large datasets, because they involve storing and processing a large number of support vectors.

23) explain the diff btwn a hard margin and a soft margin svm

➔ **Hard Margin SVM** aims to find a hyperplane that perfectly separates the data into different classes with no misclassification. It requires that all training data points be correctly classified and lie outside or on the margin.

➔ **Soft Margin SVM** introduces some flexibility by allowing for misclassification of data points. This approach is designed to handle cases where the data is not perfectly linearly separable or when there is noise in the data.

24) describe the process of constructing a decision tree.

➔ Constructing a decision tree involves creating a model that makes decisions based on the input features by recursively splitting the data into subsets. The goal is to build a tree-like structure where each internal node represents a decision based on a feature, each branch represents the outcome of that decision, and each leaf node represents a class label or a predicted value.
Process:

1. Start with the entire Dataset
2. Choose the best feature to split
3. Split the data
4. Recursively apply the process
5. Stopping criteria
6. Assign class labels or values
7. Prune the tree

25) describe the working principle of a decision tree

➔ The working principle of a decision tree involves creating a hierarchical, tree-like model that makes decisions by recursively splitting the dataset based on feature values. The tree structure consists of nodes, branches, and leaves, and each part plays a specific role in making predictions. Here's a detailed breakdown of how a decision tree works:

1. Tree Structure: Root node, Internal node, Branches and Leaf nodes.
2. Decision making process:

    a) Start with the entire Dataset
    b) Evaluate Features
    c) Choosing the best split
    d) Split the data
    e) Recursively apply the process
    f) Stopping criteria
    g) Assign class labels or values
    h) Pruning

26) what is information gain and how is it used in decision tree?

➔ Information Gain: Information Gain quantifies the reduction in entropy or uncertainty about the dataset after splitting it based on a particular feature. It helps determine which feature to use for splitting at each node in the decision tree.

➔ How Information Gain is used in Decision tree:
Information Gain is used to choose the feature that provides the most significant reduction in entropy, leading to more effective splits in the decision tree.
Decision trees use Information Gain to build a model that is easy to interpret and understand, making decisions based on the most informative features at each step.

27) explain gini impurity and its role in decision tree.

➔ **Gini Impurity** is a criterion used in decision tree algorithms to measure the impurity or disorder of a dataset. It helps in deciding the best feature to split the data at each node of the tree.
Gini impurity quantifies how often a randomly chosen element from the dataset would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the dataset.

28) what are the advantages and disadvantages of decision tree?

➔ **Advantages**:
1.Intuitive and interpretable
2. No need for feature scaling
3. Handles both numerical and categorical data
4. Models non-linear relationships
5. Provides feature importance
6. Handles missing values

➔ **Disadvantages**:
1. Prone to overfitting
2. Instability with small data changes
3. Complexity with large trees
4. Bias toward features with more levels
5. Limited expressiveness in capturing interactions
6. Bias in imbalanced datasets

29) How do random forest improve upon decision tree?

➔ **Random Forest** is an ensemble learning technique that improves upon decision trees by combining multiple trees to create a more robust and accurate model.
A Random Forest consists of a collection (or "forest") of decision trees, typically trained using a method called **bagging** (Bootstrap Aggregating). Each tree in the forest is built using a different subset of the training data and a random subset of features. The final prediction is made by aggregating the predictions of all individual trees, either by majority voting (for classification) or averaging (for regression).
1. Reduce Overfitting
2. Increase Stability
3. Handles Large Dataset and Features well
4. Improve Prediction Accuracy
5. Handles missing value
6. Robust to noise

30) how does a random forest algorithm works?

➔ A Random Forest consists of a large number of decision trees (hence, "forest"). Each tree is trained using a different subset of the training data and a random subset of features. The final prediction is made by aggregating the predictions from all the individual trees.

Random Feature Selection: For each decision tree, during the construction of each split, select a random subset of features (a feature subset) rather than using all features. This process helps to ensure that each tree is different and reduces the correlation between trees.

Tree Construction: Build each decision tree using the corresponding bootstrap sample and each node's randomly selected feature subset. Trees are usually grown to their maximum depth or until a stopping criterion is met, such as a minimum number of samples in a leaf node.

31) what is bootstrapping in the context of random forests?

➔ **Bootstrapping** is a statistical technique used in Random Forests to create multiple subsets of the training data by sampling with replacement. In the context of Random Forests, bootstrapping plays a crucial role in enhancing the diversity and robustness of the ensemble of decision trees. Here's a detailed explanation of bootstrapping and its role in Random Forests:

32) Explain the concept of feature importance in random forests

➔ Feature importance in Random Forests measures how much each feature contributes to the model's predictions. It is typically calculated using:

1. **Mean Decrease in Impurity (MDI):** Measures how much a feature reduces impurity (like Gini index) across all trees. Higher reductions indicate higher importance.
2. **Mean Decrease in Accuracy (MDA) or Permutation Importance:** Measures the decrease in model accuracy when a feature's values are randomly shuffled. A larger decrease in accuracy indicates higher importance.

   Feature importance helps with model interpretation, feature selection, and gaining insights into the data.

33) What are the key hyperparameters of a random forest and how do they affect the model?

➔ The key hyperparameters of a Random Forest and their effects on the model are as follows:

1. **n_estimators (Number of Trees):**
   - **Effect:** Determines the number of decision trees in the forest. More trees generally improve performance and reduce variance, but with diminishing returns. More trees also increase computational cost and training time.
2. **max_depth (Maximum Depth of Trees):**
   - **Effect:** Limits the depth of each tree. Shallow trees (low max_depth) prevent overfitting but may underfit the data. Deep trees (high max_depth) allow for more complex relationships but can overfit.
3. **min_samples_split (Minimum Samples to Split a Node):**
   - **Effect:** Sets the minimum number of samples required to split a node. Higher values prevent the model from learning overly specific patterns (reducing overfitting), while lower values make the model more flexible but risk overfitting.
4. **min_samples_leaf (Minimum Samples in a Leaf Node):**
   - **Effect:** Sets the minimum number of samples that must be present in a leaf node. Higher values create more general trees, reducing the risk of overfitting. Lower values allow for more specific splits.
5. **max_features (Maximum Features Considered for Splitting):**
   - **Effect:** Limits the number of features used to split at each node. Lower values reduce overfitting by introducing randomness but might underfit if too few features are used. Common settings are "sqrt" or "log2", which work well for most cases.
6. **bootstrap:**

- **Effect:** Determines whether bootstrap samples are used when building trees. When True, each tree is trained on a random subset of the data with replacement, increasing diversity among trees and reducing overfitting.

7. **criterion (Splitting Criterion):**
   - **Effect:** Defines the function to measure the quality of a split (e.g., Gini impurity or entropy for classification). The choice of criterion can affect the splits and ultimately the performance, but it usually has a smaller impact compared to other hyperparameters.
8. **max_samples (Maximum Samples for Bootstrapping):**
   - **Effect:** Limits the number of samples used for building each tree when bootstrap=True. This introduces additional randomness, potentially reducing overfitting, especially in large datasets.

34) Describe the logistic regression model and its assumptions.

➔ **Logistic Regression Model**

Logistic Regression is a model used for binary classification, predicting the probability that an input belongs to a specific class (e.g., 0 or 1). It uses the logistic (sigmoid) function:

σ(z)=11+e−z\sigma(z) = \frac{1}{1 + e^{-z}}σ(z)=1+e−z1

where zzz is a linear combination of the input features. The output is a probability between 0 and 1, with a threshold (often 0.5) to determine the class.

Assumptions of Logistic Regression

1. **Linearity of Logits:** The log-odds of the outcome are a linear combination of the input features.
2. **Independence of Errors:** The observations are independent of each other.
3. **No Multicollinearity:** Independent variables are not highly correlated.
4. **Large Sample Size:** Requires a large sample size for reliable estimates.
5. **Binary Outcome:** The dependent variable has two categories.
6. **Absence of Outliers:** Outliers can distort the model's predictions.

35) How does logistic regression handle binary classification problems?

➔ Logistic regression handles binary classification by estimating the probability that a given input belongs to one of two classes .

 Linear Combination of Features: Logistic regression first calculates a linear combination of the input features (i.e., z=β0+β1x1+β2x2+⋯+βnxnz = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_nx_nz=β0+β1x1+β2x2+⋯+βnxn), where β0\beta_0β0 is the intercept, and β1,β2,…,βn\beta_1, \beta_2, \dots, \beta_nβ1,β2,…,βn are the coefficients of the input features x1,x2,…,xnx_1, x_2, \dots, x_nx1,x2,…,xn.

 Sigmoid Function: The model applies the sigmoid function to this linear combination to map the result into a probability between 0 and 1:

P(Y=1)=σ(z)=11+e−zP(Y=1) = \sigma(z) = \frac{1}{1 + e^{-z}}P(Y=1)=σ(z)=1+e−z1

This probability represents the likelihood that the input belongs to the positive class (e.g., class 1).

 Threshold Decision: A threshold (commonly 0.5) is applied to the probability. If the probability is greater than or equal to the threshold, the model predicts the positive class (1); otherwise, it predicts the negative class (0).

 Model Training: The model is trained using maximum likelihood estimation, which adjusts the coefficients to maximize the probability of correctly classifying the training data.

36) What is the sigmoid function and how is it used in logistic regression

➔ **Sigmoid Function**

The sigmoid function, also known as the logistic function, is a mathematical function that maps any real-valued input to a value between 0 and 1. The formula for the sigmoid function is:

σ(z)=11+e−z\sigma(z) = \frac{1}{1 + e^{-z}}σ(z)=1+e−z1

where:

- zzz is the input value (often a linear combination of features in logistic regression).
- eee is the base of the natural logarithm (approximately 2.718).

Usage in Logistic Regression

In logistic regression, the sigmoid function is used to model the probability that a given input belongs to the positive class (e.g., class 1). Here's how it works:

1. **Linear Combination of Features:** The input features are combined linearly using the model's coefficients:

z=β0+β1x1+β2x2+⋯+βnxnz = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_nx_nz=β0+β1x1+β2x2+⋯+βnxn

where β0\beta_0β0 is the intercept, and β1,β2,…,βn\beta_1, \beta_2, \dots, \beta_nβ1,β2,…,βn are the coefficients.

2. **Sigmoid Function Application:** The linear combination zzz is passed through the sigmoid function:

P(Y=1)=σ(z)=11+e−zP(Y=1) = \sigma(z) = \frac{1}{1 + e^{-z}}P(Y=1)=σ(z)=1+e−z1

This output represents the probability that the input belongs to the positive class (1).

3. **Classification Decision:** A threshold (commonly 0.5) is applied to this probability. If the probability P(Y=1)P(Y=1)P(Y=1) is greater than or equal to 0.5, the model predicts the positive class (1); otherwise, it predicts the negative class (0).

37) Explain the concept of the cost function in logistic regression ?

➔ **Cost Function in Logistic Regression**

The cost function in logistic regression, also known as the loss function, measures how well the model's predictions match the actual outcomes. It is a critical component used during the training process to optimize the model's parameters (coefficients).

Logistic Regression Cost Function

In logistic regression, the cost function is based on the concept of **log-likelihood** and is specifically designed to handle binary classification problems. The cost function for a single training example can be expressed as:

Cost(hθ(x),y)=−y·log⁡(hθ(x))−(1−y)·log⁡(1−hθ(x))\text{Cost}(h_\theta(x), y) = -y \cdot \log(h_\theta(x)) - (1 - y) \cdot \log(1 - h_\theta(x))Cost(hθ(x),y)=−y·log(hθ(x))−(1−y)·log(1−hθ(x))

Where:

- $h_\theta(x)$ is the predicted probability of the positive class (i.e., $h_\theta(x) = \sigma(z) = \frac{1}{1 + e^{-z}}$).
- y is the actual class label (0 or 1).

Explanation of the Formula

- **When $y = 1$:**

$$\text{Cost}(h_\theta(x), 1) = -\log(h_\theta(x))$$

If the model predicts a high probability for the positive class (close to 1), the cost is low. If the prediction is far from 1, the cost increases sharply.

- **When $y = 0$:**

$$\text{Cost}(h_\theta(x), 0) = -\log(1 - h_\theta(x))$$

If the model predicts a low probability for the positive class (close to 0), the cost is low. If the prediction is far from 0, the cost increases sharply.

Overall Cost Function

The overall cost function, also known as the **logistic loss** or **log-loss**, is the average cost over all training examples:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Where:

- m is the number of training examples.
- θ represents the model parameters (coefficients).

Purpose of the Cost Function

- **Optimization:** The cost function is used in optimization algorithms, such as gradient descent, to find the model parameters θ that minimize the cost. Minimizing the cost function means the model's predictions are as close as possible to the actual labels.
- **Model Evaluation:** It helps evaluate how well the model is performing during training. A lower cost indicates better model performance.

38) How can logistic regression be extended to handle multiclass classification?

➔ Logistic regression can be extended to handle multiclass classification through two main approaches: **One-vs-Rest (OvR)** and **Softmax Regression (also known as Multinomial Logistic Regression)**.

1. One-vs-Rest (OvR) Approach

- **Concept:** The One-vs-Rest approach involves breaking down a multiclass classification problem into multiple binary classification problems. For each class, a separate logistic regression model is trained to distinguish that class from all other classes.
- **How It Works:**
    - For a problem with K classes, K binary logistic regression models are trained.

- Each model predicts the probability of its respective class.
- For a new input, all KKK models output a probability, and the class with the highest probability is chosen as the final prediction.

- **Example:**
  - If you have three classes (A, B, C), you would train three models:
    1. Model 1: A vs. (B + C)
    2. Model 2: B vs. (A + C)
    3. Model 3: C vs. (A + B)
  - During prediction, the class with the highest probability from these three models is selected.

2. Softmax Regression (Multinomial Logistic Regression)

- **Concept:** Softmax regression is a generalization of logistic regression to multiple classes. Instead of modeling binary outcomes, softmax regression directly models the probability of each class.
- **How It Works:**
  - The model calculates a score (logit) for each class as a linear combination of input features.
  - These scores are then passed through the **softmax function** to convert them into probabilities that sum to 100%.
  - The softmax function for class jjj is given by: P(y=j|x)=ezj∑k=1KezkP(y=j | x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}P(y=j|x)=∑k=1Kezkezj where zjz_jzj is the score for class jjj, and KKK is the total number of classes.
- **Prediction:** The model predicts the class with the highest probability.

39) What is the difference between L1 and L2 regularization in logistic regression\

➔ **L1 Regularization (Lasso)**

- **Penalty Term:** L1 regularization adds the absolute values of the coefficients to the cost function:

Cost Function with L1=J(θ)+λ∑j=1n|θj|\text{Cost Function with L1} = J(\theta) + \lambda \sum_{j=1}^{n} |\theta_j|Cost Function with L1=J(θ)+λj=1∑n|θj|

where λ\lambdaλ is the regularization parameter controlling the strength of the penalty, and θj\theta_jθj are the model's coefficients.

- **Effect:** L1 regularization can lead to sparse models, meaning it can drive some coefficients to exactly zero, effectively performing feature selection. This makes L1 useful when you expect that only a few features are important.
- **Sparsity:** Because L1 regularization tends to zero out some coefficients, it can simplify the model and make it easier to interpret.

L2 Regularization (Ridge)

- **Penalty Term:** L2 regularization adds the squared values of the coefficients to the cost function:

Cost Function with L2=J(θ)+λ∑j=1nθj2\text{Cost Function with L2} = J(\theta) + \lambda \sum_{j=1}^{n} \theta_j^2Cost Function with L2=J(θ)+λj=1∑nθj2

- **Effect:** L2 regularization penalizes large coefficients more heavily than small ones but does not zero out coefficients. Instead, it shrinks them towards zero, reducing their impact but keeping all features in the model.
- **Smoothness:** L2 regularization tends to produce models with smaller coefficients, leading to smoother predictions and generally better performance when many features contribute to the outcome.

Summary of Differences

- **L1 Regularization:**
  - Adds absolute values of coefficients as a penalty.
  - Can result in sparse models (some coefficients exactly zero).
  - Useful for feature selection.
- **L2 Regularization:**
  - Adds squared values of coefficients as a penalty.
  - Does not produce sparse models (coefficients are shrunk but not zeroed).
  - Leads to smoother and more evenly distributed coefficients.

40) What is XGBoost and how does it differ from other boosting algorithms?

➔ XGBoost (e**X**treme **G**radient **Boost**ing) is an advanced machine learning library for gradient boosting, optimized for speed and performance, especially with structured/tabular data.

How XGBoost Differs from Other Boosting Algorithms:

1. **Speed and Efficiency:** XGBoost is highly optimized, faster, and more efficient than traditional boosting algorithms.
2. **Regularization:** XGBoost includes L1 and L2 regularization to prevent overfitting, unlike standard gradient boosting methods.
3. **Handling Missing Data:** XGBoost can automatically handle missing values during training, while other methods require more preprocessing.
4. **Parallel Processing:** XGBoost supports parallel processing, making it scalable to large datasets. Other boosting algorithms typically lack this level of optimization.

41) Explain the concept of boosting in the context of ensemble learning

➔ Boosting is an ensemble learning technique that combines the predictions of multiple weak learners (usually decision trees) to create a strong predictive model. It works by training each weak learner sequentially, with each one focusing on the mistakes made by the previous learners. The key idea is to "boost" the performance of the model by giving more weight to the misclassified instances in each iteration, thereby improving the overall accuracy.In summary, boosting converts weak models into a strong model by focusing on the errors of previous models in the sequence.

42) How does XGBoost handle missing values.

➔ XGBoost handles missing values automatically by learning the best direction to take when it encounters them during the training process. Specifically:

Sparsity-Aware Split Finding: During training, XGBoost considers missing values as a separate category and determines the optimal way to handle them for each decision tree split. It does this by evaluating whether assigning missing values to the left or right branch of a split leads to a better reduction in the loss function.Default Direction: For each split, XGBoost assigns missing values to the side (left or right) that leads to the most significant improvement in model performance. This direction is learned from the data and applied consistently during prediction.

43) What are the key hyperparameters in XGBoost and how do they affect model performance.

➔ **Key Hyperparameters in XGBoost:**

1. **n_estimators:** Number of trees in the model. More trees can improve performance but increase the risk of overfitting.
2. **learning_rate:** Step size for updating weights. Lower values slow learning and require more trees but can lead to better generalization.
3. **max_depth:** Maximum depth of each tree. Deeper trees can capture more complexity but may overfit.
4. **subsample:** Fraction of the training data used for each tree. Reducing this can prevent overfitting but might decrease model accuracy.
5. **colsample_bytree:** Fraction of features used for each tree. Lower values can reduce overfitting and improve generalization.
6. **lambda (L2 regularization) & alpha (L1 regularization):** Regularization terms to control model complexity and prevent overfitting.
7. **gamma:** Minimum loss reduction required to make a split. Higher values make the model more conservative.

These hyperparameters control the balance between model complexity, training speed, and the risk of overfitting, directly affecting the model's performance.

44) Describe the process of gradient boosting in XGBoost.

➔ **Gradient Boosting Process in XGBoost:**

1. **Initialize the Model:** Start with an initial prediction, often the mean of the target values.
2. **Calculate Residuals:** Compute the difference (residuals) between the actual target values and the current predictions. These residuals represent the errors the model needs to correct.
3. **Train Weak Learners:** Train a decision tree (weak learner) to predict the residuals. The tree focuses on the patterns in the errors, helping to improve the model's accuracy.
4. **Update Predictions:** The predictions from the tree are scaled by a learning rate and added to the current predictions, updating the model's overall prediction.
5. **Iterate:** Repeat the process by calculating new residuals, training another tree on these residuals, and updating the predictions. Continue this process for a specified number of iterations (trees).
6. **Final Model:** The final model is the sum of all the trees, each correcting the errors of the previous ones, resulting in a strong predictive model.

45) What are the advantages and disadvantages of using XGBoost?

➔ **Advantages of XGBoost:**

1. **High Performance:** XGBoost is known for its speed and efficiency in both training and prediction, thanks to optimizations like parallel processing and tree pruning.
2. **Regularization:** It includes L1 (Lasso) and L2 (Ridge) regularization, which helps prevent overfitting and improves model generalization.
3. **Handling Missing Data:** XGBoost can automatically handle missing values during training, reducing the need for extensive data preprocessing.
4. **Flexibility:** Supports various objective functions and evaluation metrics, making it suitable for a wide range of tasks beyond classification, such as regression and ranking.
5. **Feature Importance:** Provides insights into feature importance, which can be useful for model interpretation and feature selection.

Disadvantages of XGBoost:

1. **Complexity:** The large number of hyperparameters can make tuning the model challenging and time-consuming.
2. **Memory Usage:** XGBoost can be memory-intensive, particularly with large datasets and many trees.
3. **Training Time:** Although faster than some methods, training a large number of trees or a very deep model can still be time-consuming.
4. **Overfitting Risk:** Despite regularization, XGBoost models can still overfit, especially if not properly tuned or if too many trees are used.