

```
# 1.what is the difference between static and dynamic variables in python?
# static variables:
# -these are the variables which cannot change the they are in the complete class.they cannot change in
# when we make new object the of the class
class A:
    wheels=4
    print(wheels)
a=A()
a
# here wheels cannot change the so it is not static
# DYNAMIC VARIABLES:
# these variables become when we make the object of the class and they change for the differnet object for tl
class A:
    def __init__(self,color):
        self.color=color
        print(color)
b=A("red")
b
```

4  
red  
<\_\_main\_\_.A at 0x7b8a3bf807c0>

```
# explain the purpose of pop popitem clear() in a dictionary variables in pyhton ?
dict={"a":"ar","b":"br","c":"cr","d":"dr"}
# pop:-In this we give the key of the item and it delete the item
dict.pop('b')
print(dict)
# popitem:-In this its remove the last item of the dictionary
print(dict.popitem())
print(dict)
# clear:-it clear the dictionary or make the dictionary empty or remove all the item of the
dict.clear()
print(dict)
```

{'a': 'ar', 'c': 'cr', 'd': 'dr'}  
( 'd', 'dr')  
{'a': 'ar', 'c': 'cr'}  
{}

```
# frozen set with example
# these are immutable they cannot change once you made them
s=frozenset([1,2,3,4])
# s.add(5)
# add operation will not perfrom
print(s)
```

frozenset({1, 2, 3, 4})

```
# difference between immutalble and mutable datatype
# mutable datatype:-these are dynamic data type in which we can perform the add update delete and pop
# operation bascialy we change the length of the of the datatype
# eg-list ,array ,dictionary
l=[1,2,3,4]
l.append(5)
print(l)
# immutable:-these are dynamic in which we cannot perform add update delete and pop
# operation bascialy we not change the length of the of the datatype
# e.g=string,int,tuple
a=5
print()
```

```
→ [1, 2, 3, 4, 5]
5
```

```
# what is the __init__()?explain the example
# it is basically a constructor which call when we make object firstly it will execute
# it also setup initial and set the attribute of the object
class Car:
```

```
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
        print(f"Car created: {self.brand} {self.model}")
```

```
# OBJECT BANATE WAQT `__init__()` AUTOMATICALLY CALL HOGA
my_car = Car("Toyota", "Corolla")
```

```
→ Car created: Toyota Corolla
```

```
# docstring basically these are used to the meaning of the code these basically like a comment used for unde
def add_numbers(a, b):
```

```
    """
    YEH FUNCTION DO NUMBERS KO ADD KARTA HAI AUR RESULT RETURN KARTA HAI.

    PARAMETERS:
    a: PEHLA NUMBER
    b: DOOSRA NUMBER

    RETURN:
    DO NUMBERS KA SUM
    """
    return a + b
```

```
# DOCSTRING KO DEKHNE KE LIYE
print(add_numbers.__doc__)
```

```
→
```

```
YEH FUNCTION DO NUMBERS KO ADD KARTA HAI AUR RESULT RETURN KARTA HAI.

PARAMETERS:
a: PEHLA NUMBER
b: DOOSRA NUMBER

RETURN:
DO NUMBERS KA SUM
```

```
# unittest in python
# used for test the code in which we divide the code into small parts and test them and we ensure that our
import unittest
```

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

```
class TestAddNumbers(unittest.TestCase):

    def test_add_positive(self):
        self.assertEqual(add_numbers(1, 2), 3)

    def test_add_negative(self):
        self.assertEqual(add_numbers(-1, -2), -3)
```

```
# break continue and pass
# break:-break is basically is used for the the break the conditon or terminate the condition
for i in range(1,10):
    if i==5:
        break
    print(i,end=" ")
print()
# continue:-it is basically skip the condition and excute the complete code same but only skip that conditc
for i in range(1,10):
    if i==5:
        continue
    print(i,end=" ")
print()
# pass:-it basically used as a placeholder when we don't have the anything to write in the condtion then us
for i in range(1,10):
    if i==5:
        pass
    print(i,end=" ")
```

```
⇒ 1 2 3 4
   1 2 3 4 6 7 8 9
   1 2 3 4 5 6 7 8 9
```

```
# self :- it is basically used in the class when we make a object of the class and then we make funcation c
# class than we will pass the parameter in the class it shows the funcation is related with which class .it
# reference to the instance of the class
```

```
class A:
    def ss(self):
        print("rahul is good boy")
a=A()
a.ss()
```

```
⇒ rahul is good boy
```

```
# global:- they can excess these outside from the class
```

```
class car:
    def __init__(self,brand):
        self.brand=brand
```

```
a=car("Tata")
```

```
a.brand
```

```
# Protected:-only child can acess these own class
```

```
class college:
    def __init__(self):
        self._brand="Pw skills"
```

```
class st(college):
    def __init__(self,name):
        self.name=name
        college.__init__(self)
```

```
    def show(self):
        print(self.name)
        print(self._brand)
```

```
a=college()
```

```
a._brand
```

```
b=st('rao')
```

```
b.show()
```

```
# private :-they can access inside the class
```

```
class A:
    def __init__(self):
        self.__name="rao"
    def show(self):
        print(self.__name)
```

```
a=A()
```

```
a.show()
```

```

→ rao
Pw skills
rao

```

```

# package:- folder is called package in this it may consist many files and we make different files for dif
# my_package/
#   __init__.py
#   module1.py
#   module2.py
# def func1():
#     return "This is function 1 from module1"
# def func2():
#     return "This is function 2 from module2"
# from my_package import module1, module2

```

```

# print(module1.func1()) # Output: This is function 1 from module1
# print(module2.func2()) # Output: This is function 2 from module2
# module:-these are files in which we write the code or operations it is a single file which consist classes
def great():
    print("this is module")

```

```

# list:-it is data structure it is mutable and we update it
l=[1,2,3,4]
l.append(5)
# tuple:-these are immutable we cannot update it and fast
l=(1,2,3,4)

```

```

# INTERPRETED LANGUAGE:
# they used the interpreter
# they execute the code line by line
# they execute directly no compilation
# these are type checking language depend on the nature
# errors show on runtime and stop the execution in runtime.
# eg-python,js

```

```

# DYNAMICALLY TYPED LANGUAGE:
# decide the variable in runtime
# they do both compile and interpret
# type check on runtime
# more flexible
# error detect on runtime

```

```

# LIST COMPREHENSION:
# we can make the new list using the existing list we can perform the operation in the single line
l=[1,2,3,4,5]
s=[i*i for i in l]
print(s)
# Dictionary Comprehension:-
d={i:i*i for i in l}
print(d)

```

```

→ [1, 4, 9, 16, 25]
   {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

```

```
# Decorators:-
# they can modify the function behavior and it give reusable code .basically it used for reuse if we want
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

```
⇒ Something is happening before the function is called.
Hello!
Something is happening after the function is called.
```

# it allocate the memory dynamically when object create ,it handle the memory for small object memory alloc  
 # sometime memory allocate and deallocate when we make the copy of the any variable or other than it make ne  
 # copied variable so if we change in last one than no chnage in new. but in other case when assign the value  
 # to another variable than if we change the anything in the old one than will also in new variable also

```
# lambda function:- these are the anonymous function they are short function they used expression and ar
# they are high order function and inline functions
# they are single expression and no statements
a=lambda x:x**2
a(5)
```

```
⇒ 25
```

```
# split and join
# split function is used for the split the string based on some argument
l=['rao@gmail.com','ahir@gmail.com','rohan@gmail.com']
a=[]
for i in l:
    a.append(i.split('@')[0])
print(a)
```

```
⇒ ['rao', 'ahir', 'rohan']
```

```

# Iterable
# this is object in which we transerve,we can transverse by loop
# eg- list,tuple etc.
l=[1,2,3,4,5]
for i in l:
    print(i,end=" ")
print()
# Iterator
# these used object __iter__() and __next__() they implement the method these method.they provide the squer
l=[1,2,3]
l=iter(l)
print(next(l),end=" ")
print(next(l),end=" ")
print(next(l),end=" ")
# Generator
# they give the value one by one and they lazily produced they genrate the value when we need they used "yi
def gens(n):
    for i in range(n):
        yield i**2
a=gens(5)
print(next(a))
print(next(a))
print(next(a))

```

```

➡ 1 2 3 4 5
   1 2 3 0
   1
   4

```

Xrange is used in old version in the python and it used in python 2 it is memory efficiently in python 2 ar

```

for i in xrange(1,10,2):
    print(i,end=" ")

```

Range is used in python 3 and it moe memory efficiently and lazily used in the evaluted in python it is mor

```

python
for i in range(1,10,2):
    print(i,end=" ")

```

```

➡ 1 3 5 7 9

```

```

# four pillar of oops
# Encapsulation
# Encapsulation is the concept of bundling the data (attributes) and the methods (functions)
# that operate on the data into a single unit or class. It restricts direct access to some of an object's
# components, which is a means of preventing accidental interference and misuse
# of the data. Access to the data is typically controlled through public methods, known as getters and sett
class Student:
    def __init__(self, name, age):
        self.name = name          # Public attribute
        self.__age = age          # Private attribute

    def get_age(self):
        return self.__age        # Public method to access the private attribute

student = Student("Rahul", 21)
print(student.name)              # Accessing public attribute
print(student.get_age())         # Accessing private attribute via method
# Inheritance
# Definition: Inheritance allows a new class to inherit the properties and methods of an existing class.
# The new class is called the derived or child class, and the existing class is known as the base or parent
# class. This promotes code reuse and the creation of a hierarchical relationship between classes.
class Animal:
    def speak(self):
        print("Animal speaks")

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

dog = Dog()
dog.speak() # Inherited method
dog.bark()  # Own method
# Polymorphism
# Definition: Polymorphism means "many forms." It allows methods to do different things based on the
# object it is acting upon, even though they share the same name. This is often achieved through method ove
class Animal:
    def speak(self):
        print("Animal speaks")

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

class Cat(Animal):
    def speak(self):
        print("Cat meows")

def make_animal_speak(animal):
    animal.speak()

dog = Dog()
cat = Cat()

make_animal_speak(dog) # Output: Dog barks
make_animal_speak(cat) # Output: Cat meows
# Abstraction
# Definition: Abstraction is the concept of hiding the complex implementation details and showing only
# the essential features of the object. It helps in reducing programming complexity and effort by providir
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(Shape):

```

```

def __init__(self, width, height):
    self.width = width
    self.height = height

def area(self):
    return self.width * self.height

rectangle = Rectangle(5, 10)
print(rectangle.area()) # Output: 50

```

➞ Rahul  
21  
Animal speaks  
Dog barks  
Dog barks  
Cat meows  
50

```

# to check the class child of the another class
# we will the subclass function
# issubclass(child_class, parent_class)
class Animal:
    pass

class Dog(Animal):
    pass

class Cat:
    pass

# Check if Dog is a subclass of Animal
print(issubclass(Dog, Animal)) # Output: True

# Check if Cat is a subclass of Animal
print(issubclass(Cat, Animal)) # Output: False

# Check if Dog is a subclass of Cat
print(issubclass(Dog, Cat)) # Output: False

```

➞ True  
False  
False



```
# Inheritance in object-oriented programming (OOP) is a mechanism where a new class, known as a child class
# properties and behaviors (methods) from an existing class, known as a parent class or superclass. Inheri
# reusability and the creation of hierarchical relationships between classes.
```

```
# There are several types of inheritance in Python:
```

```
# 1. Single Inheritance
```

```
# Single inheritance occurs when a class inherits from just one parent class. This is the simplest form of
```

```
class Animal:
```

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

```
class Dog(Animal): # Dog inherits from Animal
```

```
    def bark(self):
        print("Dog barks")
```

```
dog = Dog()
```

```
dog.speak() # Inherited method
```

```
dog.bark() # Dog's own method
```

```
# Here, Dog is inheriting from a single class Animal.
```

```
# 2. Multiple Inheritance
```

```
# Multiple inheritance occurs when a class inherits from more than one parent class. The child class has ac
```

```
class Animal:
```

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

```
class Mammal:
```

```
    def has_fur(self):
        print("Mammal has fur")
```

```
class Dog(Animal, Mammal): # Dog inherits from both Animal and Mammal
```

```
    def bark(self):
        print("Dog barks")
```

```
dog = Dog()
```

```
dog.speak() # From Animal
```

```
dog.has_fur() # From Mammal
```

```
dog.bark() # Dog's own method
```

```
# In this example, Dog inherits from both Animal and Mammal.
```

```
# 3. Multilevel Inheritance
```

```
# Multilevel inheritance occurs when a class inherits from another class, which in turn inherits from anoth
```

```
class Animal:
```

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

```
class Mammal(Animal): # Mammal inherits from Animal
```

```
    def has_fur(self):
        print("Mammal has fur")
```

```
class Dog(Mammal): # Dog inherits from Mammal
```

```
    def bark(self):
        print("Dog barks")
```

```
dog = Dog()
```

```
dog.speak() # From Animal
```

```
dog.has_fur() # From Mammal
```

```
dog.bark() # Dog's own method
```

```
# Here, Dog inherits from Mammal, which in turn inherits from Animal.
```

```
# 4. Hierarchical Inheritance
```

# Hierarchical inheritance occurs when multiple classes inherit from the same parent class. Each child clas

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

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

class Cat(Animal): # Cat also inherits from Animal
    def meow(self):
        print("Cat meows")

dog = Dog()
cat = Cat()

dog.speak() # Inherited from Animal
dog.bark()  # Dog's own method

cat.speak() # Inherited from Animal
cat.meow()  # Cat's own method
# In this example, both Dog and Cat classes inherit from the Animal class.
```

# 5. Hybrid Inheritance

# Hybrid inheritance is a combination of two or more types of inheritance. It is used to handle complex inh

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

class Mammal(Animal):
    def has_fur(self):
        print("Mammal has fur")

class Bird(Animal):
    def has_feathers(self):
        print("Bird has feathers")

class Bat(Mammal, Bird): # Bat inherits from both Mammal and Bird
    def fly(self):
        print("Bat flies")
```

```
bat = Bat()
bat.speak()      # From Animal
bat.has_fur()    # From Mammal
bat.has_feathers() # From Bird
bat.fly()        # Bat's own method
# In this example, Bat class inherits from both Mammal and Bird, creating a hybrid inheritance structure
```

```
➡ Animal speaks
   Dog barks
   Animal speaks
   Mammal has fur
   Dog barks
   Animal speaks
   Mammal has fur
   Dog barks
   Animal speaks
   Dog barks
   Animal speaks
   Cat meows
   Animal speaks
   Mammal has fur
   Bird has feathers
   Bat flies
```

```
# Encapsulation
# Definition: ENCAPSULATION DATA AUR METHODS KO EK SAATH BIND KARTA HAI, TAKE DATA KO DIRECTLY ACCESS
# NA KIYA JA SAKE. ISME ACCESS MODIFIERS (PUBLIC, PRIVATE, PROTECTED) USE KIYE JATE HAIN JINSE DATA KO PRC

class Student:
    def __init__(self, name, age):
        self.name = name        # Public attribute
        self.__age = age        # Private attribute

    def get_age(self):
        return self.__age      # Public method to access private attribute

student = Student("Rahul", 21)
print(student.name)           # Accessing public attribute
print(student.get_age())       # Accessing private attribute via method
```

```
⇒ Rahul
   21
```

```
# Polymorphism
# Definition: Polymorphism means "many forms." It allows methods to do different things based on the object
# acting upon, even though they share the same name. This is often achieved through method overriding and

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

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

class Cat(Animal):
    def speak(self):
        print("Cat meows")

def make_animal_speak(animal):
    animal.speak()

dog = Dog()
cat = Cat()

make_animal_speak(dog) # Output: Dog barks
make_animal_speak(cat) # Output: Cat meows
```

```
⇒ Dog barks
   Cat meows
```

```
# Question 1.2
# serial_no.
# Invalid: The period (.) is not allowed in variable names. Variable names can only contain letters, digits
# character), and underscores (_).

# 1st_Room
# Invalid: Variable names cannot start with a digit. They must begin with a letter or an underscore.

# Hundered$
# Invalid: The dollar sign ($) is not allowed in variable names. Only letters, digits, and underscores are

# total-Marks
# Invalid: The hyphen (-) is not allowed in variable names. Variable names can only include letters, digits

# Total Marks
# Invalid: Variable names cannot contain spaces. They must be a single continuous string of letters, digits

# True
# Invalid: True is a reserved keyword in Python. It is used to represent the boolean value true and cannot

# Question 1.3 (a)
```

```
name=['mohan','dash','karam','chandra','gandhi','bapu']
name.insert(0,"freedom_fighter")
name
```

```
→ ['freedom_fighter', 'mohan', 'dash', 'karam', 'chandra', 'gandhi', 'bapu']
```

```
# Question 1.3 (b)
# the output will be 8 because length1 is 3 and length2 5 so add length1 and length2 is 8 so output will be
```

```
# Question 1.3 (c)
name.append("Netaji")
name.append("Bose")
name
```

```
→ ['freedom_fighter',
    'mohan',
    'dash',
    'karam',
    'chandra',
    'gandhi',
    'bapu',
    'Netaji',
    'Bose']
```

```
# Question 1.3 (d)
name=["bapuji","dash","karam","chandra","gandhi","mohan"]
temp=name[-1]
# the temp value is mohan
```

```
# Question 1.4
# find the output of the following
animal=['hanuman','cat','mat','cat','rat','hanuman','lion']
print(animal.count('hanuman'))
# it will give output 2 because it is two times
print(animal.index('rat'))
# it will give you output 4 because rat is come at index 4
print(len(animal))
# the animal length is 7
```

```
# Question 1.5
tuple1=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22),[{"roll_no":1}, {"name": "Navneet"}])
print(len(tuple1))
# the length is 8

print(tuple1[-1][-1]['name'])
# the value of this is Navneet

print(tuple1[-1][-2]['roll_no'])
# the value of the roll no. is 1

print(tuple1[-3][1])
# the output is the "ji"

# fetch the 22 element from the tuple
print(tuple1[6][2])
```

```
⇒ 8
   Navneet
   1
   ji
   22
```

```
# Question 1.6
# write a program to display the appropriate message as per the color of signal(Red-stop/yellow-stay/green-
signal_color = input("Enter the color of the signal (Red/Yellow/Green):")
if signal_color.lower()=='red':
    print("STOP")
elif signal_color.lower() == 'yellow':
    print("STAY")
elif signal_color.lower() == 'green':
    print("GO")
else:
    print("INVALID COLOR")
```

```
⇒ Enter the color of the signal (Red/Yellow/Green):white
   INVALID COLOR
```

```
a=input("enter the first value")
b=input("enter the second value")
try:
    num1=int(a)
    num2=int(b)
    operation = input("Enter the operation (+, -, *, /): ")
    if operation == '+':
        print(num1 + num2)
    elif operation == '-':
        print(num1 - num2)
    elif operation == '*':
        print(num1 * num2)
    elif operation == '/':
        print(num1/num2)
    else:
        print("invalid operator")
except Exception as e:
    print(e)
```

```
⇒ enter the first value1
   enter the second value2
   Enter the operation (+, -, *, /): w
   invalid operator
```

```
# Question 1.8
# Pre-specified numbers
a = 10
b = 20
c = 15

# Find the largest using a nested ternary operator
largest = a if (a > b and a > c) else (b if b > c else c)

# Output the result
print(f"The largest number is: {largest}")
```

➞ The largest number is: 20

```
# factor using loop
# Function to find the common factors of three numbers
def find_common_factors(a, b, c):
    # Determine the minimum of the three numbers (as a factor can't be larger than the smallest number)
    min_num = min(a, b, c)

    # List to store common factors
    common_factors = []

    # Loop to find common factors
    for i in range(1, min_num + 1):
        if a % i == 0 and b % i == 0 and c % i == 0:
            common_factors.append(i)

    return common_factors

# Input: Three numbers
a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
c = int(input("Enter the third number: "))

# Find and print the common factors
factors = find_common_factors(a, b, c)
print(f"The common factors of {a}, {b}, and {c} are: {factors}")
```

➞ Enter the first number: 5  
Enter the second number: 10  
Enter the third number: 3  
The common factors of 5, 10, and 3 are: [1]

```
# Input: User enters a whole number
number = int(input("Enter a whole number: "))
a=[]
for i in range(2,number+1):
    if number%i==0:
        a.append(i)
print(f"the factors are the {a}")
```

➞ Enter a whole number: 10  
the factors are the [2, 5, 10]

```
# Question 1.10
n=int(input("enter the number"))
s=0
for i in range(n):
    a=int(input("enter the number"))
    if(a<0):
        break
    s=s+a
print(s)
```

```
↩ enter the number4
  enter the number1
  enter the number2
  enter the number3
  enter the number4
  10
```

```
# Question 1.11
for i in range(1,101):
    prime=True
    for j in range(2,int(i**0.5) + 1):
        if i%j==0:
            prime=False
            break
    if prime:
        print(i,end=" ")
```

```
↩ 1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

```
# Question 1.12
# Function to calculate the grade based on percentage
def calculate_grade(percentage):
    if percentage > 85:
        return 'A'
    elif 75 <= percentage <= 85:
        return 'B'
    elif 50 <= percentage < 75:
        return 'C'
    elif 30 <= percentage <= 50:
        return 'D'
    else:
        return 'Reappear'
```

```
# Input: Accept marks for five major subjects
marks = []
for i in range(1, 6):
    mark = int(input(f"Enter marks for subject {i}: "))
    marks.append(mark)
```

```
# Calculate total and percentage
total_marks = sum(marks)
percentage = (total_marks / 500) * 100
```

```
# Determine the grade
grade = calculate_grade(percentage)
```

```
# Display the results
print(f"\nTotal Marks: {total_marks}/500")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")
```

```
↩ Enter marks for subject 1: 100
  Enter marks for subject 2: 80
```

```
Enter marks for subject 3: 60
Enter marks for subject 4: 60
Enter marks for subject 5: 60
```

```
Total Marks: 360/500
Percentage: 72.00%
Grade: C
```

```
# Input: Accept marks for five major subjects
marks = []
for i in range(1, 6):
    mark = int(input(f"Enter marks for subject {i}: "))
    marks.append(mark)

# Calculate the total marks
total_marks = sum(marks)

# Calculate the percentage
percentage = total_marks / 5

# Display the total marks and percentage
print(f"\nTotal Marks: {total_marks}/500")
print(f"Percentage: {percentage:.2f}%")
```

```
➞ Enter marks for subject 1: 10
Enter marks for subject 2: 20
Enter marks for subject 3: 30
Enter marks for subject 4: 40
Enter marks for subject 5: 50
```

```
Total Marks: 150/500
Percentage: 30.00%
```



```
# Function to calculate the grade based on percentage
def calculate_grade(percentage):
    if percentage > 85:
        return 'A'
    elif 75 <= percentage <= 85:
        return 'B'
    elif 50 <= percentage < 75:
        return 'C'
    elif 30 <= percentage <= 50:
        return 'D'
    else:
        return 'Reappear'

# Input: Accept marks for five major subjects
marks = []
for i in range(1, 6):
    mark = int(input(f"Enter marks for subject {i}: "))
    marks.append(mark)

# Calculate the total marks
total_marks = sum(marks)

# Calculate the percentage
percentage = total_marks / 5

# Determine the grade
grade = calculate_grade(percentage)

# Display the total marks, percentage, and grade
print(f"\nTotal Marks: {total_marks}/500")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")
```

➡ Enter marks for subject 1: 60  
Enter marks for subject 2: 60  
Enter marks for subject 3: 60  
Enter marks for subject 4: 70  
Enter marks for subject 5: 90

Total Marks: 340/500  
Percentage: 68.00%  
Grade: C

```
# Function to determine color based on wavelength
def determine_color(wavelength):
    if 400 <= wavelength < 440:
        return "Violet"
    elif 440 <= wavelength < 460:
        return "Indigo"
    elif 460 <= wavelength < 500:
        return "Blue"
    elif 500 <= wavelength < 570:
        return "Green"
    elif 570 <= wavelength < 590:
        return "Yellow"
    elif 590 <= wavelength < 620:
        return "Orange"
    elif 620 <= wavelength <= 720:
        return "Red"
    else:
        return "Wavelength out of visible range"

# Input: Wavelength
wavelength = float(input("Enter the wavelength (in nm): "))

# Determine the color
color = determine_color(wavelength)

# Output: Color
print(f"The color corresponding to the wavelength {wavelength} nm is {color}.")
```

Enter the wavelength (in nm): 200  
The color corresponding to the wavelength 200.0 nm is Wavelength out of visible range.

```
# 1.14
# Constants
G = 6.674e-11 # Gravitational constant in N(m^2)/(kg^2)

# Given masses and distances
mass_earth = 5.972e24 # Mass of Earth in kg
mass_sun = 1.989e30 # Mass of Sun in kg
distance_earth_sun = 1.496e11 # Distance between Earth and Sun in meters

# Calculate the gravitational force between Earth and Sun
force_earth_sun = (G * mass_earth * mass_sun) / (distance_earth_sun ** 2)

# Output the result
print(f"The gravitational force between the Earth and the Sun is: {force_earth_sun:.2e} N")
force_earth_moon = (G * mass_earth * mass_moon) / (distance_moon_earth ** 2)

# Output the result
print(f"The gravitational force between the Earth and the Moon is: {force_earth_moon:.2e} N")
if force_earth_moon > force_earth_sun:
    print("moon gravitonal force is more")
else:
    print("earth gravitonal force is more")
# the earth is more attracted because it has a more gravitonal force than as compare to moon

The gravitational force between the Earth and the Sun is: 3.54e+22 N
The gravitational force between the Earth and the Moon is: 1.98e+20 N
earth gravitonal force is more
```

```

# Question 2
# design and implement a python program for managing the student information using object-oriented program
# create a class called student with encapsulated attributes for name age and roll no.implemented getter and setter
# these attributes. additionally ,provide methods to display student information and update student details
# Tasks
# define student class with encapsulated attributes.
# implement getter and setter methods for attribute
# write a method to display student information and update details
# create instance of the student class and set test implemented functionality
class student:
    def __init__(self,name,age,rollno):
        self.__name=name
        self.__age=age
        self.__roll_no=rollno
    def get_name(self):
        return self.__name
    def set_name(self,name):
        self.__name=name
    def get_age(self):
        return self.__age
    def set_age(self,age):
        self.__age=age
    def set_roll_no(self,roll_no):
        self.__roll_no=roll_no
    def get_roll_no(self):
        return self.__roll_no
    def display_info(self):
        return f"name { self.__name} class {self.__age} roll number {self.__roll_no}"
    def update_info(self,name,age,roll_no):
        self.__name=name
        self.__age=age
        self.__roll_no=roll_no

st=student("rohan",22,1223)
print("Initial Student Information:")
print(st.display_info())
st.update_info("Jane Doe", 21,2012)

# Display updated student information
print("\nUpdated Student Information:")
st.display_info()

```

Initial Student Information:  
name rohan class 22 roll number 1223

Updated Student Information:  
'name Jane Doe class 21 roll number 2012'

```

# question 3
# develop the python program for managing library resources efficiently.design a class method name 'library
# author, and availability status.implement methods for borrowing and returning books while ensuring prope
# tasks.
# create the librarybook class with encapsulated attributes.
# implement methods for borrowing and returning books.
# ensure proper encapsulation to protect book details
# test the borrowing and returning functionlity with sample data
class LibraryBook:
    def __init__(self, book_name, author, is_available=True):
        self.__book_name = book_name
        self.__author = author
        self.__is_available = is_available

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

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

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

    # Method to borrow a book
    def borrow_book(self):
        if self.__is_available:
            self.__is_available = False
            print(f"The book '{self.__book_name}' by {self.__author} has been borrowed.")
        else:
            print(f"Sorry, the book '{self.__book_name}' by {self.__author} is currently not available.")

    # Method to return a book
    def return_book(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"The book '{self.__book_name}' by {self.__author} has been returned.")
        else:
            print(f"The book '{self.__book_name}' by {self.__author} was not borrowed, so it can't be retur

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

# Create instances of the LibraryBook class
book1 = LibraryBook("The Great Gatsby", "F. Scott Fitzgerald")
book2 = LibraryBook("1984", "George Orwell")

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

# Borrow a book
print("\nAttempting to Borrow 'The Great Gatsby':")
book1.borrow_book()

# Try to borrow the same book again
print("\nAttempting to Borrow 'The Great Gatsby' Again:")
book1.borrow_book()

```

```
# Return the book
print("\nReturning 'The Great Gatsby':")
book1.return_book()

# Try to return the same book again
print("\nAttempting to Return 'The Great Gatsby' Again:")
book1.return_book()

# Display final book information
print("\nFinal Book Information:")
book1.display_info()
book2.display_info()
```



Initial Book Information:  
Book Name: The Great Gatsby  
Author: F. Scott Fitzgerald  
Availability: Available  
Book Name: 1984  
Author: George Orwell  
Availability: Available

Attempting to Borrow 'The Great Gatsby':  
The book 'The Great Gatsby' by F. Scott Fitzgerald has been borrowed.

Attempting to Borrow 'The Great Gatsby' Again:  
Sorry, the book 'The Great Gatsby' by F. Scott Fitzgerald is currently not available.

Returning 'The Great Gatsby':  
The book 'The Great Gatsby' by F. Scott Fitzgerald has been returned.

Attempting to Return 'The Great Gatsby' Again:  
The book 'The Great Gatsby' by F. Scott Fitzgerald was not borrowed, so it can't be returned.

Final Book Information:  
Book Name: The Great Gatsby  
Author: F. Scott Fitzgerald  
Availability: Available  
Book Name: 1984  
Author: George Orwell  
Availability: Available

```
# Question 5
# write a python program that models are different animals and their sounds. design a base class called animal
# create subclass like dog and cat that override the make_sound() method to produce appropriate sounds.
# Task
# define Animal class with a method make_sound()
# create a subclass Dog and Cat that override make_sound() method
# implement the sound generation logic for each subclass
# test the program by creating instances of dog and cat and calling the make_sound() method.
# Base class: Animal
class Animal:
    def make_sound(self):
        return "Some generic animal sound"

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

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

# Test the program by creating instances of Dog and Cat
dog = Dog()
cat = Cat()

# Calling the make_sound() method for each instance
print("Dog makes sound:", dog.make_sound())
print("Cat makes sound:", cat.make_sound())
```

➡ Dog makes sound: Woof Woof!  
Cat makes sound: Meow Meow!

```
# Question 6.
# Write a code for restaurant management system using oops
# create a menuitem class that has attribute such as name description price and category. implement method:
# and remove a menu item from the menu .use encapsulation to hide the menu item's unique identification number
# menuitem class to create a fooditem class and a beverageitem class each with their own specific attributes
# Base Class: MenuItem
class MenuItem:
    __item_id_counter = 0 # Class variable to keep track of unique item IDs

    def __init__(self, name, description, price, category):
        self.__item_id = MenuItem.__item_id_counter + 1 # Unique ID for each item
        MenuItem.__item_id_counter += 1
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def get_item_id(self):
        return self.__item_id

    def display_info(self):
        return f"ID: {self.__item_id}, Name: {self.name}, Description: {self.description}, Price: ${self.price}"

# Subclass: FoodItem
class FoodItem(MenuItem):
    def __init__(self, name, description, price, cuisine_type):
        super().__init__(name, description, price, "Food")
        self.cuisine_type = cuisine_type

    def display_info(self):
        return f"{super().display_info()}, Cuisine Type: {self.cuisine_type}"
```

```

# Subclass: BeverageItem
class BeverageItem(MenuItem):
    def __init__(self, name, description, price, size):
        super().__init__(name, description, price, "Beverage")
        self.size = size

    def display_info(self):
        return f"{super().display_info()}, Size: {self.size}"

# Restaurant Menu Management
class RestaurantMenu:
    def __init__(self):
        self.menu_items = []

    def add_item(self, item):
        self.menu_items.append(item)
        print(f"Item '{item.name}' added to the menu.")

    def remove_item(self, item_id):
        for item in self.menu_items:
            if item.get_item_id() == item_id:
                self.menu_items.remove(item)
                print(f"Item '{item.name}' removed from the menu.")
                return
        print(f"No item found with ID: {item_id}")

    def display_menu(self):
        if not self.menu_items:
            print("The menu is empty.")
        else:
            print("Menu Items:")
            for item in self.menu_items:
                print(item.display_info())

# Testing the Restaurant Management System
menu = RestaurantMenu()

# Adding food items
food1 = FoodItem("Pizza", "Cheese Pizza with tomato sauce", 8.99, "Italian")
food2 = FoodItem("Sushi", "Fresh salmon sushi", 12.99, "Japanese")
menu.add_item(food1)
menu.add_item(food2)

# Adding beverage items
bev1 = BeverageItem("Coke", "Chilled Coca-Cola", 1.99, "Medium")
bev2 = BeverageItem("Coffee", "Hot brewed coffee", 2.49, "Large")
menu.add_item(bev1)
menu.add_item(bev2)

# Display the menu
menu.display_menu()

# Remove an item from the menu
menu.remove_item(2)

# Display the menu after removal
menu.display_menu()

```



```

Item 'Pizza' added to the menu.
Item 'Sushi' added to the menu.
Item 'Coke' added to the menu.
Item 'Coffee' added to the menu.
Menu Items:
ID: 1, Name: Pizza, Description: Cheese Pizza with tomato sauce, Price: $8.99, Category: Food, Cuisine

```

```
ID: 2, Name: Sushi, Description: Fresh salmon sushi, Price: $12.99, Category: Food, Cuisine Type: Japan
ID: 3, Name: Coke, Description: Chilled Coca-Cola, Price: $1.99, Category: Beverage, Size: Medium
ID: 4, Name: Coffee, Description: Hot brewed coffee, Price: $2.49, Category: Beverage, Size: Large
Item 'Sushi' removed from the menu.
Menu Items:
ID: 1, Name: Pizza, Description: Cheese Pizza with tomato sauce, Price: $8.99, Category: Food, Cuisine
ID: 3, Name: Coke, Description: Chilled Coca-Cola, Price: $1.99, Category: Beverage, Size: Medium
ID: 4, Name: Coffee, Description: Hot brewed coffee, Price: $2.49, Category: Beverage, Size: Large
```



```

# Question 7
# write a hotel mangement system using oops
# create a room class that has attribute such as room number,room,type,rate,and availablility(private).
# implement methods to book a room ,check in a guest,and checkout a guest
# use encapsulation to hide the room' unique identification number .inherit from the room class  to create
# and a standardroom class,each with thier own specific attributes and methods
# Base Class: Room
class Room:
    __room_id_counter = 0 # Class variable to generate unique room IDs

    def __init__(self, room_number, room_type, rate, availability=True):
        self.__room_id = Room.__room_id_counter + 1 # Unique ID for each room
        Room.__room_id_counter += 1
        self.room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = availability

    def get_room_id(self):
        return self.__room_id

    def is_available(self):
        return self.__availability

    def book_room(self):
        if self.__availability:
            self.__availability = False
            print(f"Room {self.room_number} has been booked.")
        else:
            print(f"Room {self.room_number} is already booked.")

    def check_in(self):
        if not self.__availability:
            print(f"Guest checked in to room {self.room_number}.")
        else:
            print(f"Room {self.room_number} is not booked yet. Please book it first.")

    def check_out(self):
        if not self.__availability:
            self.__availability = True
            print(f"Room {self.room_number} has been checked out.")
        else:
            print(f"Room {self.room_number} is already available. No need to check out.")

    def display_info(self):
        availability = "Available" if self.__availability else "Booked"
        return f"Room ID: {self.__room_id}, Room Number: {self.room_number}, Type: {self.room_type}, Rate:

# Subclass: SuiteRoom
class SuiteRoom(Room):
    def __init__(self, room_number, rate, amenities):
        super().__init__(room_number, "Suite", rate)
        self.amenities = amenities

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Amenities: {'', '.join(self.amenities)}"

# Subclass: StandardRoom
class StandardRoom(Room):
    def __init__(self, room_number, rate, bed_type):
        super().__init__(room_number, "Standard", rate)
        self.bed_type = bed_type

    def display_info(self):
        base_info = super().display_info()

```

```

        return f"{base_info}, Bed Type: {self.bed_type}"

# Testing the Room Management System
def test_room_management():
    # Creating rooms
    suite = SuiteRoom(101, 250.0, ["WiFi", "TV", "Mini Bar"])
    standard = StandardRoom(102, 100.0, "Queen Bed")

    # Display room information
    print(suite.display_info())
    print(standard.display_info())

    # Book the suite room
    suite.book_room()
    # Attempt to check in
    suite.check_in()

    # Attempt to book the same room again
    suite.book_room()

    # Check out from the suite room
    suite.check_out()

    # Display room information after check-out
    print("\nAfter Check-Out:")
    print(suite.display_info())


    # Book and check in to the standard room
    standard.book_room()
    standard.check_in()

    # Check out from the standard room
    standard.check_out()

    # Display room information after check-out
    print("\nAfter Check-Out:")
    print(standard.display_info())

# Run the test
test_room_management()

```

 Room ID: 1, Room Number: 101, Type: Suite, Rate: \$250.0, Status: Available, Amenities: WiFi, TV, Mini Bar  
 Room ID: 2, Room Number: 102, Type: Standard, Rate: \$100.0, Status: Available, Bed Type: Queen Bed  
 Room 101 has been booked.  
 Guest checked in to room 101.  
 Room 101 is already booked.  
 Room 101 has been checked out.

After Check-Out:  
 Room ID: 1, Room Number: 101, Type: Suite, Rate: \$250.0, Status: Available, Amenities: WiFi, TV, Mini Bar  
 Room 102 has been booked.  
 Guest checked in to room 102.  
 Room 102 has been checked out.

After Check-Out:  
 Room ID: 2, Room Number: 102, Type: Standard, Rate: \$100.0, Status: Available, Bed Type: Queen Bed

```

# Question 8
# write a code for fitness club management system using oops:-
# create a member class that has a attribute 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 family member class and an individualmember class each with
# own specific attributes and method
# Base Class: Member
class Member:
    __member_id_counter = 0 # Class variable to generate unique member IDs

    def __init__(self, name, age, membership_type):
        self.__member_id = Member.__member_id_counter + 1 # Unique ID for each member
        Member.__member_id_counter += 1
        self.name = name
        self.age = age
        self.membership_type = membership_type
        self.__membership_status = "Active"

    def get_member_id(self):
        return self.__member_id

    def get_membership_status(self):
        return self.__membership_status

    def register(self):
        print(f"Member {self.name} registered with ID {self.__member_id} and {self.membership_type} members")

    def renew_membership(self):
        if self.__membership_status == "Cancelled":
            self.__membership_status = "Active"
            print(f"Membership for {self.name} has been renewed.")
        else:
            print(f"Membership for {self.name} is already active.")

    def cancel_membership(self):
        if self.__membership_status == "Active":
            self.__membership_status = "Cancelled"
            print(f"Membership for {self.name} has been cancelled.")
        else:
            print(f"Membership for {self.name} is already cancelled.")

    def display_info(self):
        return f"ID: {self.__member_id}, Name: {self.name}, Age: {self.age}, Membership Type: {self.membership_type}"

# Subclass: FamilyMember
class FamilyMember(Member):
    def __init__(self, name, age, membership_type, family_name):
        super().__init__(name, age, membership_type)
        self.family_name = family_name

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Family Name: {self.family_name}"

# Subclass: IndividualMember
class IndividualMember(Member):
    def __init__(self, name, age, membership_type, trainer_assigned):
        super().__init__(name, age, membership_type)
        self.trainer_assigned = trainer_assigned

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Trainer Assigned: {self.trainer_assigned}"

# Testing the Fitness Club Management System

```

```
def test_fitness_club_management():
    # Creating members
    family_member = FamilyMember("John Doe", 40, "Family", "Doe Family")
    individual_member = IndividualMember("Jane Smith", 28, "Individual", "Trainer Mike")

    # Register members
    family_member.register()
    individual_member.register()

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

    # Renew and cancel membership
    family_member.cancel_membership()
    family_member.renew_membership()
    individual_member.cancel_membership()

    # Display member information after changes
    print("\nAfter Membership Changes:")
    print(family_member.display_info())
    print(individual_member.display_info())

# Run the test
test_fitness_club_management()
```



Member John Doe registered with ID 1 and Family membership.  
 Member Jane Smith registered with ID 2 and Individual membership.  
 ID: 1, Name: John Doe, Age: 40, Membership Type: Family, Status: Active, Family Name: Doe Family  
 ID: 2, Name: Jane Smith, Age: 28, Membership Type: Individual, Status: Active, Trainer Assigned: Trainer Mike  
 Membership for John Doe has been cancelled.  
 Membership for John Doe has been renewed.  
 Membership for Jane Smith has been cancelled.

After Membership Changes:  
 ID: 1, Name: John Doe, Age: 40, Membership Type: Family, Status: Active, Family Name: Doe Family  
 ID: 2, Name: Jane Smith, Age: 28, Membership Type: Individual, Status: Cancelled, Trainer Assigned: Trainer Mike



```

# question 9
# write a code event mangement system using oops
# create an event that has attribute 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 numbers of attendees.
# use encapsulation to hide the event's unique identification number
# inherit from the Event class to create a PrivateEvent class and a public class each with their
# own specific attributes and method
# Base Class: Event
class Event:
    __event_id_counter = 0 # Class variable to generate unique event IDs

    def __init__(self, name, date, time, location):
        self.__event_id = Event.__event_id_counter + 1 # Unique ID for each event
        Event.__event_id_counter += 1
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = [] # Private list to store attendees

    def get_event_id(self):
        return self.__event_id

    def create_event(self):
        print(f"Event '{self.name}' created with ID {self.__event_id} on {self.date} at {self.time} in {self.location}")

    def add_attendee(self, attendee_name):
        self.__attendees.append(attendee_name)
        print(f"Attendee '{attendee_name}' added to the event '{self.name}'.")

    def remove_attendee(self, attendee_name):
        if attendee_name in self.__attendees:
            self.__attendees.remove(attendee_name)
            print(f"Attendee '{attendee_name}' removed from the event '{self.name}'.")
        else:
            print(f"Attendee '{attendee_name}' is not in the event '{self.name}'.")

    def get_total_attendees(self):
        return len(self.__attendees)

    def display_info(self):
        return f"Event ID: {self.__event_id}, Name: {self.name}, Date: {self.date}, Time: {self.time}, Location: {self.location}"

# Subclass: PrivateEvent
class PrivateEvent(Event):
    def __init__(self, name, date, time, location, invited_guests):
        super().__init__(name, date, time, location)
        self.invited_guests = invited_guests

    def add_attendee(self, attendee_name):
        if attendee_name in self.invited_guests:
            super().add_attendee(attendee_name)
        else:
            print(f"Attendee '{attendee_name}' is not on the invited guests list for the private event '{self.name}'.")

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Invited Guests: {', '.join(self.invited_guests)}"

# Subclass: PublicEvent
class PublicEvent(Event):
    def __init__(self, name, date, time, location, max_capacity):
        super().__init__(name, date, time, location)
        self.max_capacity = max_capacity

    def add_attendee(self, attendee_name):

```

```

        if self.get_total_attendees() < self.max_capacity:
            super().add_attendee(attendee_name)
        else:
            print(f"Cannot add attendee '{attendee_name}' to the public event '{self.name}'. The event is a

def display_info(self):
    base_info = super().display_info()
    return f"{base_info}, Max Capacity: {self.max_capacity}"

# Testing the Event Management System
def test_event_management():
    # Creating events
    private_event = PrivateEvent("Board Meeting", "2024-08-25", "10:00 AM", "Conference Room A", ["Alice",
    public_event = PublicEvent("Music Concert", "2024-08-30", "07:00 PM", "City Park", 100)

    # Creating events
    private_event.create_event()
    public_event.create_event()

    # Adding attendees
    private_event.add_attendee("Alice")
    private_event.add_attendee("Dave") # Should not be allowed as Dave is not invited

    public_event.add_attendee("Eve")
    public_event.add_attendee("Frank")

    # Display event information
    print("\nPrivate Event Information:")
    print(private_event.display_info())

    print("\nPublic Event Information:")
    print(public_event.display_info())

    # Remove an attendee from the private event
    private_event.remove_attendee("Alice")
    private_event.remove_attendee("Eve") # Eve is not an attendee, so it should show an error

    # Display event information after changes
    print("\nPrivate Event Information After Changes:")
    print(private_event.display_info())

    # Checking total attendees for public event
    print(f"Total Attendees for Public Event: {public_event.get_total_attendees()}")

# Run the test
test_event_management()

```



Event 'Board Meeting' created with ID 1 on 2024-08-25 at 10:00 AM in Conference Room A.  
 Event 'Music Concert' created with ID 2 on 2024-08-30 at 07:00 PM in City Park.  
 Attendee 'Alice' added to the event 'Board Meeting'.  
 Attendee 'Dave' is not on the invited guests list for the private event 'Board Meeting'.  
 Attendee 'Eve' added to the event 'Music Concert'.  
 Attendee 'Frank' added to the event 'Music Concert'.

Private Event Information:

Event ID: 1, Name: Board Meeting, Date: 2024-08-25, Time: 10:00 AM, Location: Conference Room A, Total

Public Event Information:

Event ID: 2, Name: Music Concert, Date: 2024-08-30, Time: 07:00 PM, Location: City Park, Total Attendee  
 Attendee 'Alice' removed from the event 'Board Meeting'.

Attendee 'Eve' is not in the event 'Board Meeting'.

Private Event Information After Changes:

Event ID: 1, Name: Board Meeting, Date: 2024-08-25, Time: 10:00 AM, Location: Conference Room A, Total  
 Total Attendees for Public Event: 2

```

# write a code for airline reservation sysem using oops
# create an flight class that has attribute such as flight number ,deprature ,and arrival airports,deperatur
# and avaialbe seats(private).
# implement methods to book a seat, cancel a reservation, and get available remaing 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
# own specific attributes and method
# Base Class: Flight
class Flight:
    __flight_id_counter = 0 # Class variable to generate unique flight IDs

    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time, total_seats):
        self.__flight_id = Flight.__flight_id_counter + 1 # Unique ID for each flight
        Flight.__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 = total_seats # Private attribute to store available seats

    def get_flight_id(self):
        return self.__flight_id

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            print(f"Seat booked successfully on flight {self.flight_number}.")
        else:
            print(f"No available seats on flight {self.flight_number}.")

    def cancel_reservation(self):
        self.__available_seats += 1
        print(f"Reservation cancelled successfully on flight {self.flight_number}.")

    def get_available_seats(self):
        return self.__available_seats

    def display_info(self):
        return (f"Flight ID: {self.__flight_id}, Flight Number: {self.flight_number}, "
                f"Departure: {self.departure_airport} at {self.departure_time}, "
                f"Arrival: {self.arrival_airport} at {self.arrival_time}, "
                f"Available Seats: {self.__available_seats}")

# Subclass: DomesticFlight
class DomesticFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time, total_seats, region):
        super().__init__(flight_number, departure_airport, arrival_airport, departure_time, arrival_time, total_seats)
        self.region = region

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Region: {self.region}"

# Subclass: InternationalFlight
class InternationalFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time, total_seats, passport_required):
        super().__init__(flight_number, departure_airport, arrival_airport, departure_time, arrival_time, total_seats)
        self.passport_required = passport_required

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Passport Required: {'Yes' if self.passport_required else 'No'}"

```

```

# Testing the Airline Reservation System
def test_airline_reservation():
    # Creating flights
    domestic_flight = DomesticFlight("AI101", "DEL", "MUM", "10:00 AM", "12:00 PM", 180, "North")
    international_flight = InternationalFlight("AI201", "DEL", "LHR", "02:00 PM", "06:00 PM", 300, True)

    # Displaying flight information
    print("Domestic Flight Information:")
    print(domestic_flight.display_info())

    print("\nInternational Flight Information:")
    print(international_flight.display_info())

    # Booking seats
    domestic_flight.book_seat()
    international_flight.book_seat()

    # Display available seats after booking
    print("\nAvailable Seats After Booking:")
    print(f"Domestic Flight: {domestic_flight.get_available_seats()}")
    print(f"International Flight: {international_flight.get_available_seats()}")

    # Cancelling reservation
    domestic_flight.cancel_reservation()
    international_flight.cancel_reservation()

    # Display available seats after cancellation
    print("\nAvailable Seats After Cancellation:")
    print(f"Domestic Flight: {domestic_flight.get_available_seats()}")
    print(f"International Flight: {international_flight.get_available_seats()}")

# Run the test
test_airline_reservation()

```



```

Domestic Flight Information:
Flight ID: 1, Flight Number: AI101, Departure: DEL at 10:00 AM, Arrival: MUM at 12:00 PM, Available Seats: 179

International Flight Information:
Flight ID: 2, Flight Number: AI201, Departure: DEL at 02:00 PM, Arrival: LHR at 06:00 PM, Available Seats: 299
Seat booked successfully on flight AI101.
Seat booked successfully on flight AI201.

Available Seats After Booking:
Domestic Flight: 179
International Flight: 299
Reservation cancelled successfully on flight AI101.
Reservation cancelled successfully on flight AI201.

Available Seats After Cancellation:
Domestic Flight: 180
International Flight: 300

```

```

# constants.py

# Mathematical constant Pi
PI = 3.141592653589793

# Speed of light in vacuum (meters per second)
SPEED_OF_LIGHT = 299792458 # m/s

```



```
# calculator.py

def add(a, b):
    """Returns the sum of two numbers."""
    return a + b

def subtract(a, b):
    """Returns the difference between two numbers."""
    return a - b

def multiply(a, b):
    """Returns the product of two numbers."""
    return a * b

def divide(a, b):
    """Returns the quotient of two numbers. Raises an error if divided by zero."""
    if b == 0:
        raise ValueError("Cannot divide by zero!")
    return a / b
```

```
# Question 13.
# products.py

# def add_product(product_id, name, price):
#     """Add a new product to the catalog."""
#     # Implementation to add product
#     print(f"Product {name} added with ID {product_id} and price {price}.")

# def remove_product(product_id):
#     """Remove a product from the catalog."""
#     # Implementation to remove product
#     print(f"Product with ID {product_id} removed.")
# inventory.py

# def check_stock(product_id):
#     """Check the stock level of a product."""
#     # Implementation to check stock
#     print(f"Checking stock for product ID {product_id}.")

# def update_stock(product_id, quantity):
#     """Update the stock level of a product."""
#     # Implementation to update stock
#     print(f"Updated stock for product ID {product_id} by {quantity} units.")
# orders.py

# def create_order(order_id, product_id, quantity):
#     """Create a new order."""
#     # Implementation to create order
#     print(f"Order {order_id} created for product {product_id} with quantity {quantity}.")

# def cancel_order(order_id):
#     """Cancel an existing order."""
#     # Implementation to cancel order
#     print(f"Order {order_id} cancelled.")
# payments.py

# def process_payment(order_id, amount):
#     """Process payment for an order."""
#     # Implementation to process payment
#     print(f"Payment of {amount} processed for order {order_id}.")

# def refund_payment(order_id, amount):
#     """Process a refund for an order."""
#     # Implementation to process refund
#     print(f"Refund of {amount} processed for order {order_id}.")
# main.py

# from ecommerce.product_management.products import add_product, remove_product
# from ecommerce.product_management.inventory import check_stock, update_stock
# from ecommerce.order_processing.orders import create_order, cancel_order
# from ecommerce.order_processing.payments import process_payment, refund_payment

# Product management
# add_product(1, "Laptop", 1500)
# remove_product(1)

# Inventory management
# check_stock(1)
# update_stock(1, 10)

# Order processing
# create_order(101, 1, 2)
# cancel_order(101)

# Payment processing
```

```

# process_payment(101, 3000)
# refund_payment(101, 3000)

# Question 14.

# implement a python module named_file operations.py with
# functions for reading,writing,appending,data to a file
# # file_operations.py

# def read_file(file_path):
#     """Reads the contents of a file and returns it as a string."""
#     try:
#         with open(file_path, 'r') as file:
#             content = file.read()
#             return content
#     except FileNotFoundError:
#         return "File not found."

# def write_file(file_path, data):
#     """Writes data to a file. Overwrites the file if it already exists."""
#     with open(file_path, 'w') as file:
#         file.write(data)
#     return "Data written successfully."

# def append_file(file_path, data):
#     """Appends data to the end of a file. Creates the file if it does not exist."""
#     with open(file_path, 'a') as file:
#         file.write(data)
#     return "Data appended successfully."

# Question 15.
# write a python module named file_operations.py with functions for reading,writing,
# and appending data to a file
# file_operations.py

def read_file(file_path):
    """Reads the contents of a file and returns it as a string."""
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError:
        return "File not found."

def write_file(file_path, data):
    """Writes data to a file. Overwrites the file if it already exists."""
    with open(file_path, 'w') as file:
        file.write(data)
    return "Data written successfully."

def append_file(file_path, data):
    """Appends data to the end of a file. Creates the file if it does not exist."""
    with open(file_path, 'a') as file:
        file.write(data)
    return "Data appended successfully."

# question 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
# Define employee details
employees = [
    {"name": "Alice", "age": 30, "salary": 70000},
    {"name": "Bob", "age": 35, "salary": 50000}
]

```

```

    { "name": "Bob", "age": 25, "salary": 50000},
    {"name": "Charlie", "age": 35, "salary": 80000}
]

# File path
file_path = "employees.txt"

# Write employee details to the file
def write_employee_details(file_path, employees):
    with open(file_path, 'w') as file:
        for employee in employees:
            file.write(f"Name: {employee['name']}, Age: {employee['age']}, Salary: {employee['salary']}\n")
        print(f"Employee details have been written to {file_path}")

# Call the function to write details
write_employee_details(file_path, employees)

```

➞ Employee details have been written to employees.txt

```

employees = [
    {"name": "Alice", "age": 30, "salary": 70000},
    {"name": "Bob", "age": 25, "salary": 50000},
    {"name": "Charlie", "age": 35, "salary": 80000}
]

# File path
file_path = "inventory.txt"

# Write employee details to the file
def write_employee_details(file_path, employees):
    with open(file_path, 'w') as file:
        for employee in employees:
            file.write(f"Salary: {employee['salary']}\n")
        print(f"Employee details have been written to {file_path}")

# Call the function to write details
write_employee_details(file_path, employees)

```

➞ Employee details have been written to inventory.txt

```

# 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
# Define the file path
file_path = "inventory.txt"

# Function to read and display the contents of the file line by line
def display_file_contents(file_path):
    try:
        # Open the file in read mode
        with open(file_path, 'r') as file:
            # Read and print each line
            for line in file:
                print(line.strip())
    except FileNotFoundError:
        print(f"The file {file_path} does not exist.")

# Call the function to display the contents
display_file_contents(file_path)

```

➞ Salary: 70000  
Salary: 50000  
Salary: 80000

Start coding or [generate](#) with AI.

```
# create a python program that reads a text file named 'paragraph.txt' and counts the occurrences of the
# each word in the paragraph ,displaying the results in alphabetical order
from collections import Counter
import string

# Function to read the file and count word occurrences
def count_word_occurrences(file_path):
    # Dictionary to hold word counts
    word_count = Counter()

    try:
        # Open the file in read mode
        with open(file_path, 'r') as file:
            # Read the file content
            content = file.read()

            # Convert to lowercase to make counting case-insensitive
            content = content.lower()

            # Remove punctuation from the content
            content = content.translate(str.maketrans('', '', string.punctuation))

            # Split the content into words
            words = content.split()

            # Count occurrences of each word
            word_count.update(words)

            # Sort the words alphabetically and display the results
            for word in sorted(word_count):
                print(f"{word}: {word_count[word]}")

    except FileNotFoundError:
        print(f"The file {file_path} does not exist.")

# Define the file path
file_path = "paragraph.txt"

# Call the function to count and display word occurrences
count_word_occurrences(file_path)
```

```
➞ 50000: 1
   70000: 1
   80000: 1
   salary: 3
```

```
# Question 20
# .what do you mean by measure of the central tendency and measure of
# # dispersion .how it can be calculated
# # Measure of Central Tendency refers to statistical metrics that describe the center or typical value of

# # Key Measures:
# # Mean (Arithmetic Average):


# # Explanation: Sum of all values divided by the number of values.
# # Calculation: Add all data points and divide by the number of data points.
# # Median:

# # Explanation: The middle value when data points are arranged in ascending order. If there is
# # an even number of observations, it is the average of the two middle numbers.
# Measure of Dispersion (or Spread) describes the extent to which data points in a dataset are spread out f

# Key Measures:
# Range:

# Formula:
# Range
# =
# Maximum
# -
# Minimum
# Range=Maximum-Minimum
# Explanation: Difference between the highest and lowest values in the dataset.
# Calculation: Subtract the minimum value from the maximum value.
# Variance:
# for sample
# Explanation: Average of the squared differences from the mean.
# Standard Deviation:
# Formula:
# Standard Deviation
# Variance
# Standard Deviation=
# Variance
# Explanation: Square root of the variance, providing a measure of spread in the same units as the data.
# Calculation: Take the square root of the variance.
# Interquartile Range (IQR):

# Formula:
#  $IQR = Q3 - Q1$ 
#  $IQR = Q3 - Q1$ 
# Explanation: The range between the first quartile (25th percentile) and the third quartile (75th percenti
# Calculation:
# Find the first quartile (Q1) and third quartile (Q3).
# Subtract Q1 from Q3.
```

 Employee details have been written to paragraph.txt

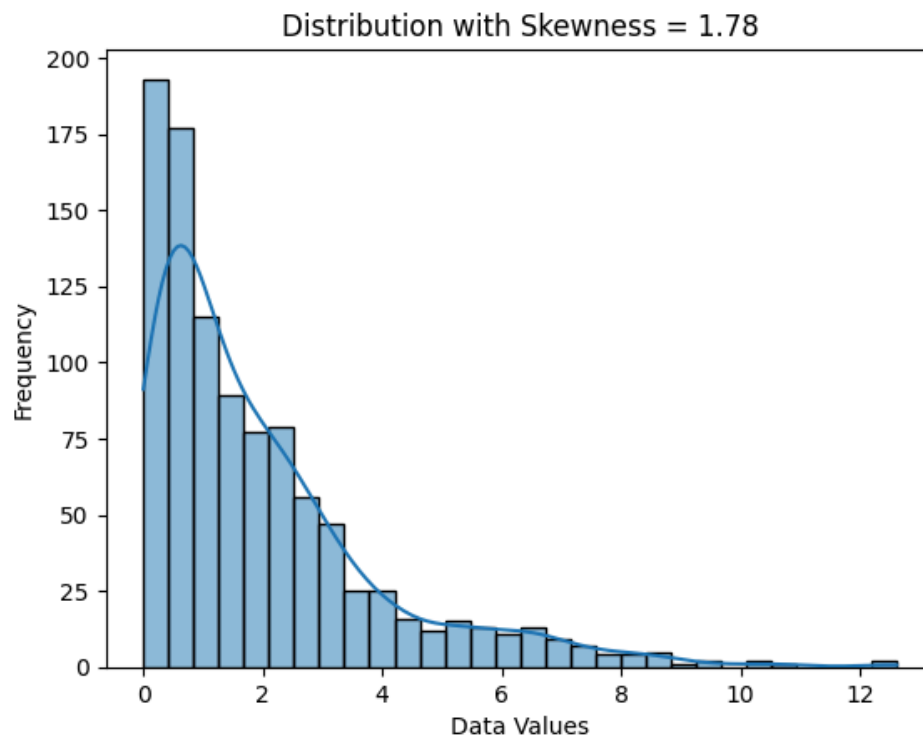
```
# Question 21
# what do you mean by skewness by graph
# Skewness refers to the asymmetry of a distribution. If a distribution is perfectly symmetrical, its skewness is 0.

# Positive Skewness: The tail is longer on the right side.
# Negative Skewness: The tail is longer on the left side.
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import skew

# Generate data with positive skewness
data = np.random.exponential(scale=2, size=1000)

# Calculate skewness
skewness_value = skew(data)

# Plot the distribution
sns.histplot(data, kde=True)
plt.title(f'Distribution with Skewness = {skewness_value:.2f}')
plt.xlabel('Data Values')
plt.ylabel('Frequency')
plt.show()
```



```

# # Question 22
# # explain pmf and pdf
# PMF (Probability Mass Function)
# Definition: PMF is used with discrete random variables. It gives the probability that a discrete random v
# Formula:
#  $p(X=x)$ 
#  $P(X=x)$ , where  $X$ 
#  $X$  is the random variable and  $x$ 
#  $x$  is a specific value.
# Characteristics:
# The PMF sums up to 1 across all possible values of the random variable.
# It is used when the random variable can take on specific, distinct values (e.g., the roll of a dice).
# . PDF (Probability Density Function)
# Definition: PDF is used with continuous random variables. It represents the likelihood of a random
# variable taking on a specific value, but since the variable is continuous, the probability at any single
# point is 0. Instead, the PDF is used to determine the probability that the variable falls within a certa
# range.
#  $f(x)$  is the PDF.
# Characteristics:
# The area under the PDF curve over the entire space equals 1.
# It is used when the random variable can take on any value within a continuous range.

```

```

# # Question 24
# orrelation is a statistical measure that describes the strength and direction of the relationship between

```

```

# Key Points about Correlation:
# Direction of Correlation:

```

```

# Positive Correlation: When one variable increases, the other variable also increases. For example, height
# Negative Correlation: When one variable increases, the other variable decreases. For example, the number
# No Correlation: There is no consistent relationship between the variables. Changes in one variable do not
# Strength of Correlation:

```

```

# Correlation Coefficient ( $r$ ): The strength of correlation is quantified by the correlation coefficient, wh
#  $r$ 
# =
# 1
#  $r=1$ : Perfect positive correlation.
#  $r$ 
# =
# -
# 1
#  $r=-1$ : Perfect negative correlation.
#  $r$ 
# =
# 0
#  $r=0$ : No correlation.
# Values closer to 1 or -1 indicate a stronger relationship, while values closer to 0 indicate a weaker rel
# Types of Correlation:

```

```

# Pearson Correlation: Measures the linear relationship between two continuous variables. It's the most com
# Spearman's Rank Correlation: Measures the strength and direction of the association between two ranked va
# Kendall's Tau: Another measure for ranked variables, it's similar to Spearman's but less sensitive to err
# Example:
# If we have data on students' study hours and their test scores, we might calculate the Pearson correlatic
#  $r$ 
# =
# 0.8
#  $r=0.8$ , it indicates a strong positive correlation, meaning that generally, more study hours are associate

```

```

# Important Considerations:
# Correlation does not imply causation: Just because two variables are correlated doesn't mean that one cau
# Linear Relationship: Pearson correlation only measures linear relationships. If the relationship is non-l

```



```
import pandas as pd

# DATA DICTIONARY
data = {
    'Student': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Accountancy': [45, 70, 65, 30, 90, 40, 50, 75, 85, 60],
    'Statistics': [35, 90, 70, 40, 95, 40, 60, 80, 80, 50]
}

# CREATING DATAFRAME
df = pd.DataFrame(data)

# DISPLAY DATAFRAME
print(df)

correlation = df[['Accountancy', 'Statistics']].corr().iloc[0, 1]

# DISPLAY CORRELATION
print("Correlation between Accountancy and Statistics:", correlation)
```



	Student	Accountancy	Statistics
0	1	45	35
1	2	70	90
2	3	65	70
3	4	30	40
4	5	90	95
5	6	40	40
6	7	50	60
7	8	75	80
8	9	85	80
9	10	60	50

Correlation between Accountancy and Statistics: 0.9031178882610627

# Purpose:

# Correlation: Measures the strength and direction of the relationship between two variables. It indicates  
 # Regression: Aims to predict the value of a dependent variable based on the value(s) of one or more indepe  
 # Measurement:

# Correlation: Provides a correlation coefficient that ranges from -1 to +1. This coefficient shows how str  
 # Regression: Provides a regression equation (e.g.,  $Y = a + bX$ ) that describes the relationship between var  
 # Output:

# Correlation: Outputs a single value called the correlation coefficient, which indicates the degree of lir  
 # Regression: Outputs a regression equation or model that can be used to estimate or predict the dependent  
 # Causality:

# Correlation: Does not imply causation. It only indicates whether and how strongly two variables are relat  
 # Regression: Attempts to establish a causal relationship between variables. It helps determine whether cha

```

# # # find the most likely price at delhi corresponding to the price of rs.70 at agra from the following
# # # coefficient of correlation between the prices of two places +0.8
# # TO ESTIMATE THE MOST LIKELY PRICE IN DELHI CORRESPONDING TO A PRICE OF RS. 70 IN AGRA,
# # YOU CAN USE THE LINEAR RELATIONSHIP BETWEEN THE TWO PRICES. GIVEN THAT THE CORRELATION
# # COEFFICIENT BETWEEN THE PRICES IN AGRA AND DELHI IS +0.8, WE CAN USE
# # THIS INFORMATION TO MAKE AN ESTIMATE.THE RELATIONSHIP CAN BE SIMPLIFIED USING THE CORRELATION COEFFICIENT
# WHERE SD IS THE STANDARD DEVIATION AND Mean IS THE AVERAGE.
# WITHOUT SPECIFIC VALUES FOR MEAN AND STANDARD DEVIATION, A SIMPLER APPROACH IS TO USE THE CORRELATION COEFFICIENT
# EXAMPLE:
# IF WE ASSUME THAT THE AVERAGE PRICE IN DELHI IS SIMILAR TO THE AVERAGE PRICE IN AGRA, THEN THE ESTIMATION
# Estimated Price in Delhi
# =
# Mean Price in Delhi
# +0.8·(
# Price in Agra -Mean Price in Agra)
# Estimated Price in Delhi=Mean Price in Delhi+0.8·(Price in Agra-Mean Price in Agra)
# IF MEAN PRICES ARE UNKNOWN, ASSUME THEY ARE THE SAME AND SIMPLY APPLY:
# Estimated Price in Delhi
# ≈
# Correlation·(Price in Agra)
# Estimated Price in Delhi=Correlation·(Price in Agra)
# Estimated Price in Delhi=0.8·70=56
# Estimated Price in Delhi=0.8·70=56
# SO, THE MOST LIKELY PRICE IN DELHI CORRESPONDING TO A PRICE OF RS. 70 IN AGRA CAN BE APPROXIMATED TO RS.

# 8x-10y=-66
# 40x-18y=214
# Mean of x = 13
# Mean of y= 17
# Variance of
# x = 9
# Variance of
# y = 11.25
# Standard deviation of
# x = 3
# Standard deviation of
# y = 3.35

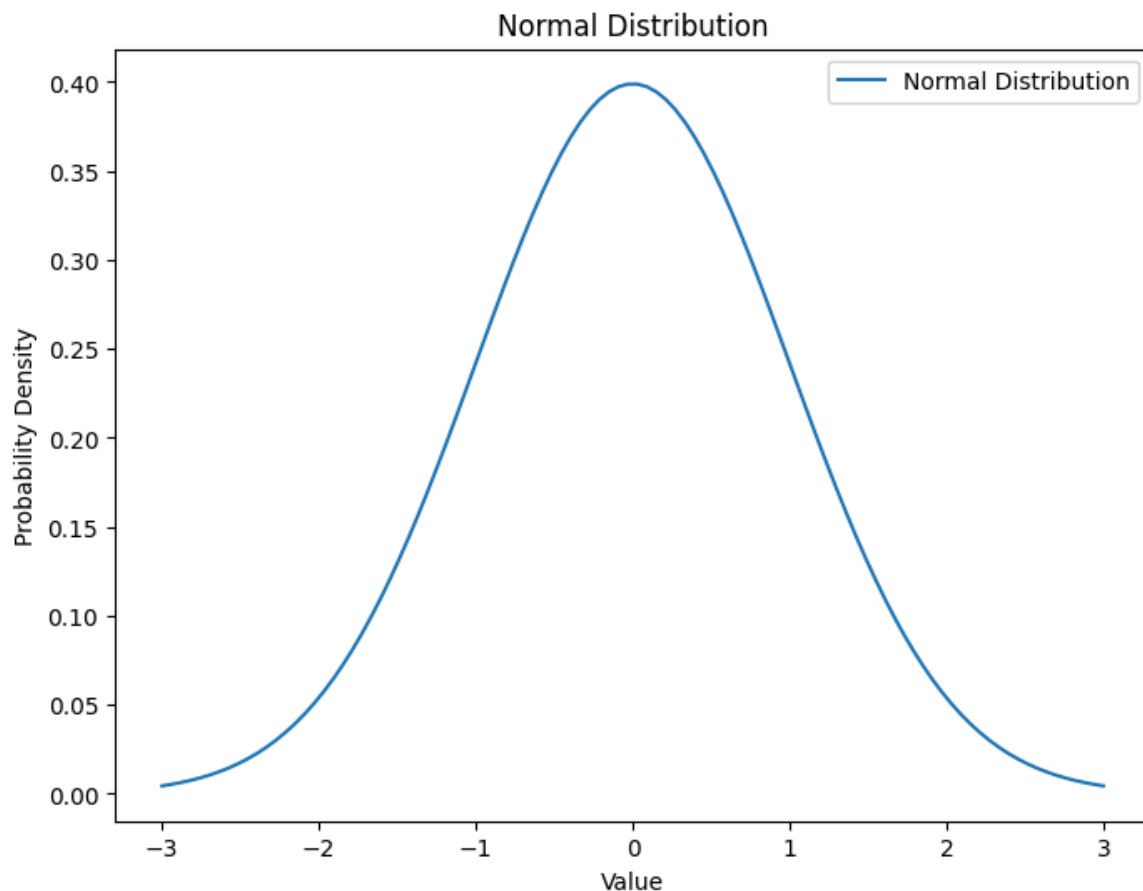
```

```
# Symmetry: The distribution is symmetric around its mean. The left and right sides of the curve are mirror
# images of each other.
# Mean, Median, and Mode: In a normal distribution, the mean, median, and mode are all equal and located
# at the center of the distribution.
# Bell-Shaped Curve: The shape of the normal distribution curve is bell-shaped. It is highest at the
# mean and decreases as you move away from the mean in either direction.
# Defined by Two Parameters: The normal distribution is completely defined by two parameters: the mean
# ( $\mu$ ) and the standard deviation ( $\sigma$ ). The mean determines the center of the distribution, while the
# standard deviation determines the spread or width of the distribution.
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Parameters
mean = 0
std_dev = 1

# Generate data
x = np.linspace(mean - 3*std_dev, mean + 3*std_dev, 100)
y = (1/(std_dev * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - mean) / std_dev) ** 2)

# Plot
plt.figure(figsize=(8, 6))
sns.lineplot(x=x, y=y, label='Normal Distribution')
plt.title('Normal Distribution')
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.legend()
plt.show()
```



# Question 29

# THE NORMAL DISTRIBUTION HAS SEVERAL DISTINCT CHARACTERISTICS THAT MAKE IT A FUNDAMENTAL CONCEPT IN STATISTICS

# \*\*1. Symmetry:

# Bell-Shaped Curve: THE CURVE IS SYMMETRIC AROUND THE MEAN. THIS MEANS THE LEFT SIDE OF THE CURVE IS A MIRROR IMAGE OF THE RIGHT SIDE.

# Mean, Median, and Mode: IN A NORMAL DISTRIBUTION, THE MEAN, MEDIAN, AND MODE ARE ALL EQUAL AND LOCATED AT THE CENTER OF THE DISTRIBUTION.

# \*\*2. Mean, Median, and Mode:

# Central Location: THE MEAN ( $\mu$ ) IS LOCATED AT THE CENTER OF THE DISTRIBUTION AND IS THE POINT OF SYMMETRY.

# \*\*3. Spread:

# Standard Deviation: THE STANDARD DEVIATION ( $\sigma$ ) MEASURES THE SPREAD OF THE DISTRIBUTION. A SMALLER STANDARD DEVIATION INDICATES A MORE TIGHTLY CLUSTERED DISTRIBUTION.

# Variance: THE VARIANCE ( $\sigma^2$ ) IS THE SQUARE OF THE STANDARD DEVIATION AND PROVIDES A MEASURE OF THE DISPERSION.

# Question 30.

# Summary of Correct Options:

# a. Correct

# b. Approximately correct

# c. Approximately correct

# d. Correct

# e. Correct

# OPTIONS a, d, AND e ARE DEFINITELY CORRECT, WHILE b AND c ARE APPROXIMATELY CORRECT BUT HAVE SLIGHTLY DIFFERENT VALUES.

# question 31.

# Percentage of items between 60 and 72: 38.49%

# Percentage of items between 50 and 60: 34.13%

# Percentage beyond 72: 11.51%

# Percentage between 70 and 80: 13.59%

# Question 32

# Proportion of students scoring more than 55 marks: 15.87% (approximately 238 students)

# Proportion of students scoring more than 70 marks: 0.02% (approximately 1 student)

# Question 33

# YES, THAT'S CORRECT!

# PROPORTION OF STUDENTS SCORING MORE THAN 55 MARKS: 15.87% WHICH CORRESPONDS TO APPROXIMATELY 238 STUDENTS

# PROPORTION OF STUDENTS SCORING MORE THAN 70 MARKS: 0.02% WHICH CORRESPONDS TO APPROXIMATELY 1 STUDENT OUT OF 50

# Question 34

# 1. NULL HYPOTHESIS ( $H_0$ ):

# Definition: THE NULL HYPOTHESIS IS A STATEMENT THAT THERE IS NO EFFECT OR NO DIFFERENCE, AND IT REPRESENTS THE STATUS QUO.

# Example: IF YOU ARE TESTING WHETHER A NEW DRUG HAS AN EFFECT, THE NULL HYPOTHESIS MIGHT BE THAT THE DRUG HAS NO EFFECT.

# 2. ALTERNATIVE HYPOTHESIS ( $H_1$  OR  $H_A$ ):

# Definition: THE ALTERNATIVE HYPOTHESIS IS A STATEMENT THAT CONTRADICTS THE NULL HYPOTHESIS. IT SUGGESTS THAT THERE IS AN EFFECT OR A DIFFERENCE.

# Example: USING THE SAME DRUG STUDY, THE ALTERNATIVE HYPOTHESIS MIGHT BE THAT THE DRUG DOES HAVE AN EFFECT.

# TYPES OF ALTERNATIVE HYPOTHESES:

# ONE-TAILED (DIRECTIONAL) ALTERNATIVE HYPOTHESIS:

# SUGGESTS THAT THE EFFECT OR DIFFERENCE IS IN A SPECIFIC DIRECTION (E.G., GREATER THAN OR LESS THAN).

# EXAMPLE: THE MEAN OF GROUP A IS GREATER THAN THE MEAN OF GROUP B.

# TWO-TAILED (NON-DIRECTIONAL) ALTERNATIVE HYPOTHESIS:

# SUGGESTS THAT THERE IS A DIFFERENCE, BUT DOES NOT SPECIFY THE DIRECTION OF THE DIFFERENCE.

# EXAMPLE: THE MEAN OF GROUP A IS NOT EQUAL TO THE MEAN OF GROUP B.

```
# Question 35
import scipy.stats as stats

# Given data
n = 25 # Sample size
s = 9.0 # Sample standard deviation
sigma_0 = 10.5 # Hypothesized population standard deviation

# Calculate the chi-square test statistic
chi_square_stat = (n - 1) * (s ** 2) / (sigma_0 ** 2)

# Degrees of freedom
df = n - 1

# Significance level (alpha)
alpha = 0.05

# Critical values for the chi-square distribution (two-tailed test)
chi_square_critical_low = stats.chi2.ppf(alpha / 2, df)
chi_square_critical_high = stats.chi2.ppf(1 - alpha / 2, df)

# Print results
print(f"Chi-Square Test Statistic: {chi_square_stat:.2f}")
print(f"Critical Value (Lower): {chi_square_critical_low:.2f}")
print(f"Critical Value (Upper): {chi_square_critical_high:.2f}")

# Decision rule
if chi_square_critical_low < chi_square_stat < chi_square_critical_high:
    print("Fail to reject the null hypothesis (H0).")
else:
    print("Reject the null hypothesis (H0).")
# SINCE 17.63 IS BETWEEN THE CRITICAL VALUES 12.401 AND 39.364, WE FAIL TO REJECT THE NULL HYPOTHESIS.
# THERE IS NOT ENOUGH EVIDENCE TO REJECT THE NULL HYPOTHESIS. THEREFORE, WE CONCLUDE THAT THE POPULATION ST
```

```

# Question 36
import scipy.stats as stats

# Given data
observed_frequencies = [15, 17, 30, 22, 16] # Observed frequencies for grades A, B, C, D, E
total_students = sum(observed_frequencies) # Total number of students

# Expected frequency for a uniform distribution
expected_frequency = [total_students / len(observed_frequencies)] * len(observed_frequencies)

# Perform the chi-square test
chi_square_stat, p_value = stats.chisquare(f_obs=observed_frequencies, f_exp=expected_frequency)

# Degrees of freedom (df) is number of categories - 1
df = len(observed_frequencies) - 1

# Critical value for chi-square distribution at alpha = 0.05
alpha = 0.05
chi_square_critical = stats.chi2.ppf(1 - alpha, df)

# Print results
print(f"Chi-Square Test Statistic: {chi_square_stat:.2f}")
print(f"P-Value: {p_value:.4f}")
print(f"Critical Value (Chi-Square Distribution): {chi_square_critical:.2f}")

# Decision rule
if chi_square_stat > chi_square_critical:
    print("Reject the null hypothesis (H0): The distribution of grades is not uniform.")
else:
    print("Fail to reject the null hypothesis (H0): The distribution of grades is uniform.")

```

```

→ Chi-Square Test Statistic: 7.70
P-Value: 0.1032
Critical Value (Chi-Square Distribution): 9.49
Fail to reject the null hypothesis (H0): The distribution of grades is uniform.

```

```

pip install Flask
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "Hello, World!"

if __name__ == "__main__":
    app.run(debug=True)

```

```
# Question 40
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        # Get form data
        name = request.form.get("name")
        return f"Hello, {name}!"
    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Form Submission</title>
</head>
<body>
    <h1>Enter Your Name</h1>
    <form method="POST" action="/">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

```
# question 41
from flask import Flask

app = Flask(__name__)

# Route that accepts a parameter in the URL
@app.route("/greet/<name>")
def greet(name):
    return f"Hello, {name}!"

if __name__ == "__main__":
    app.run(debug=True)
```

```

# Question 42
from flask import Flask, render_template, redirect, url_for, request
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired

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'

# Dummy user store
users = {
    'user1': {'password': 'password1'}
}

class User(UserMixin):
    def __init__(self, username):
        self.id = username

@login_manager.user_loader
def load_user(user_id):
    return User(user_id) if user_id in users else None

# Form classes
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        username = form.username.data
        password = form.password.data
        user = users.get(username)
        if user and user['password'] == password:
            login_user(User(username))
            return redirect(url_for('protected'))
    return render_template('login.html', form=form)

@app.route('/protected')
@login_required
def protected():
    return f'Hello, {current_user.id}! You are logged in.'

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)

```



```

# Question 43
from flask import Flask, render_template, redirect, url_for, request
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired

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'

# Dummy user store
users = {
    'user1': {'password': 'password1'}
}

class User(UserMixin):
    def __init__(self, username):
        self.id = username

@login_manager.user_loader
def load_user(user_id):
    return User(user_id) if user_id in users else None

# Form classes
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        username = form.username.data
        password = form.password.data
        user = users.get(username)
        if user and user['password'] == password:
            login_user(User(username))
            return redirect(url_for('protected'))
    return render_template('login.html', form=form)

@app.route('/protected')
@login_required
def protected():
    return f'Hello, {current_user.id}! You are logged in.'

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>

```

```
<body>
  <h1>Login</h1>
  <form method="POST" action="">
    {{ form.hidden_tag() }}
    <p>
      {{ form.username.label }}<br>
      {{ form.username() }}
    </p>
    <p>
      {{ form.password.label }}<br>
      {{ form.password() }}
    </p>
    <p>
      {{ form.submit() }}
    </p>
  </form>
</body>
</html>
```

```
# Question 43
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# Configure SQLite database
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}>'

# Route to create a new user
@app.route('/add_user', methods=['POST'])
def add_user():
    username = request.json.get('username')
    email = request.json.get('email')
    if username and email:
        new_user = User(username=username, email=email)
        db.session.add(new_user)
        db.session.commit()
        return jsonify({"message": "User added!"}), 201
    return jsonify({"message": "Invalid input!"}), 400

# Route to get all users
@app.route('/users', methods=['GET'])
def get_users():
    users = User.query.all()
    return jsonify([{'id': user.id, 'username': user.username, 'email': user.email} for user in users])

if __name__ == '__main__':
    # Create tables
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

```
# Question 44
from flask import Flask, jsonify

app = Flask(__name__)

# Define a RESTful API endpoint
@app.route('/api/data', methods=['GET'])
def get_data():
    # Example data to return
    data = {
        "id": 1,
        "name": "Alice",
        "email": "alice@example.com",
        "age": 30
    }
    return jsonify(data)

if __name__ == '__main__':
    app.run(debug=True)
```

```

# Question 45
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, Email, Length

app = Flask(__name__)
app.secret_key = 'your_secret_key' # REQUIRED FOR FORM SECURITY

# Define a form class
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():
        # Access form data
        username = form.username.data
        email = form.email.data
        password = form.password.data
        # Process the data (e.g., check credentials)
        return redirect(url_for('success'))
    return render_template('login.html', form=form)

@app.route('/success')
def success():
    return 'Login Successful!'

if __name__ == '__main__':
    app.run(debug=True)
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form method="POST" action="">
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}
        </p>
        <p>
            {{ form.email.label }}<br>
            {{ form.email(size=32) }}
        </p>
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}
        </p>
        <p>
            {{ form.submit() }}
        </p>
    </form>
</body>
</html>

```

```
# Question 46
from flask import Flask, render_template, request, redirect, url_for
import os

app = Flask(__name__)

# Set the upload folder and allowed file extensions
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'jpg', 'jpeg', 'png', 'gif'}

# Ensure the upload folder exists
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

def allowed_file(filename):
    """Check if the file has an allowed extension."""
    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':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(filename)
            return redirect(url_for('uploaded_file', filename=file.filename))
    return render_template('upload.html')

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return f'File uploaded successfully: {filename}'

if __name__ == '__main__':
    app.run(debug=True)
```

```
# Question 47
from flask import Flask

app = Flask(__name__)

# Import the blueprint
from my_blueprint import my_blueprint
app.register_blueprint(my_blueprint)

@app.route('/')
def home():
    return 'Welcome to the Home Page'

if __name__ == '__main__':
    app.run(debug=True)
from flask import Blueprint

# Create a Blueprint instance
my_blueprint = Blueprint('my_blueprint', __name__)

@my_blueprint.route('/blueprint')
def blueprint_home():
    return 'Welcome to the Blueprint Page'
<!DOCTYPE html>
<html>
<head>
    <title>Blueprint Page</title>
</head>
<body>
    <h1>Welcome to the Blueprint Page</h1>
</body>
</html>
```

# Questio 48

```
Deploying a Flask Application Using Gunicorn and Nginx
Gunicorn AND Nginx ARE COMMONLY USED FOR DEPLOYING FLASK APPLICATIONS IN A PRODUCTION ENVIRONMENT
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form method="POST">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>
        <br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
        <br>
        <input type="submit" value="Login">
    </form>
    <a href="{{ url_for('signup') }}">Sign Up</a>
</body>
</html>
```

```
from flask import Flask, render_template, redirect, url_for, request, session
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash
```

```
app = Flask(__name__)
app.secret_key = 'your_secret_key'
app.config['MONGO_URI'] = 'mongodb://localhost:27017/yourdatabase'
```

```

mongo = pymongo(app)

@app.route('/')
def home():
    if 'username' in session:
        return f'Hello {session["username"]}!'
    return redirect(url_for('login'))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = mongo.db.users.find_one({'username': username})
        if user and check_password_hash(user['password'], password):
            session['username'] = username
            return redirect(url_for('home'))
    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']
        hashed_password = generate_password_hash(password, method='sha256')
        mongo.db.users.insert_one({'username': username, 'password': hashed_password})
        return redirect(url_for('login'))
    return render_template('signup.html')

if __name__ == '__main__':
    app.run(debug=True)

```

# Question 49

```

sudo apt update
sudo apt install nginx
gunicorn -w 4 -b 0.0.0.0:8000 app:app
sudo apt update
sudo apt install nginx
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /static {
        alias /path/to/your/static;
    }
}
sudo ln -s /etc/nginx/sites-available/yourapp /etc/nginx/sites-enabled
sudo nginx -t
sudo systemctl restart nginx
sudo apt update
sudo apt install mongodb
sudo systemctl start mongodb

```



```
# Question 50
# 1. Difference Between Series and DataFrames
# Series AND DataFrames ARE BOTH DATA STRUCTURES IN PANDAS. HERE'S A COMPARISON:

# Series:

# ONE-DIMENSIONAL
# STORES DATA IN A SINGLE COLUMN
# HAS AN INDEX
# USAGE: REPRESENTS A SINGLE VARIABLE
# DataFrame:

# TWO-DIMENSIONAL
# STORES DATA IN A TABULAR FORMAT WITH MULTIPLE COLUMNS
# HAS ROW AND COLUMN INDEXES
# USAGE: REPRESENTS MULTIPLE VARIABLES OR FEATURES
# 2. Creating and Reading a MySQL Table with Pandas
# Create the Database and Table:

# Create the Database and Table:

# sql
# Copy code
# 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');
# Read the Table Using Pandas:

# python
# Copy code
# import pandas as pd
# import mysql.connector

# # Connect to the MySQL database
# conn = mysql.connector.connect(
#     host='localhost',
#     user='your_username',
#     password='your_password',
#     database='Travel_Planner'
# )

# # Query the table and read it into a DataFrame
# query = 'SELECT * FROM bookings'
# df = pd.read_sql(query, conn)

# # Close the connection
# conn.close()

# # Show the DataFrame
# print(df)
# 3. Difference Between loc and iloc
# loc:
```

```
# LABEL-BASED INDEXING
# USES ROW AND COLUMN LABELS
# EXAMPLE: df.loc['row_label', 'column_label']
# iloc:

# INTEGER-BASED INDEXING
# USES INTEGER POSITION INDEXES
# EXAMPLE: df.iloc[0, 1] (FIRST ROW, SECOND COLUMN)
# 4. Supervised vs. Unsupervised Learning
# Supervised Learning:

# LEARNING FROM LABELED DATA
# MODEL IS TRAINED ON INPUT-OUTPUT PAIRS
# EXAMPLES: REGRESSION, CLASSIFICATION
# Unsupervised Learning:

# LEARNING FROM UNLABELED DATA
# MODEL IDENTIFIES PATTERNS OR STRUCTURE IN THE DATA
# EXAMPLES: CLUSTERING, DIMENSIONALITY REDUCTION
# 5. Bias-Variance Tradeoff
# Bias:

# ERROR DUE TO ASSUMPTIONS IN THE MODEL
# HIGH BIAS CAN LEAD TO UNDERFITTING
# Variance:

# ERROR DUE TO FLUCTUATIONS IN THE TRAINING DATA
# HIGH VARIANCE CAN LEAD TO OVERFITTING
# Tradeoff:

# A MODEL WITH LOW BIAS AND HIGH VARIANCE MAY OVERFIT
# A MODEL WITH HIGH BIAS AND LOW VARIANCE MAY UNDERFIT
# THE GOAL IS TO FIND A BALANCE THAT MINIMIZES BOTH BIAS AND VARIANCE.
# 6. Precision vs. Recall vs. Accuracy
# Accuracy:

# PROPORTION OF TRUE RESULTS (BOTH TRUE POSITIVES AND TRUE NEGATIVES) OUT OF TOTAL RESULTS.
# Accuracy = (TP + TN) / (TP + TN + FP + FN)
# Precision:

# PROPORTION OF TRUE POSITIVES OUT OF ALL POSITIVE PREDICTIONS.
# Precision = TP / (TP + FP)
# Recall:

# PROPORTION OF TRUE POSITIVES OUT OF ALL ACTUAL POSITIVES.
# Recall = TP / (TP + FN)
# 7. Overfitting and How to Prevent It
# Overfitting:

# WHEN A MODEL LEARNS NOISE IN THE TRAINING DATA INSTEAD OF THE UNDERLYING PATTERNS.
# RESULTS IN POOR PERFORMANCE ON NEW DATA (TEST SET).
# Prevention Techniques:

# Cross-Validation: USE TO VALIDATE THE MODEL ON DIFFERENT SUBSETS OF DATA.
# Regularization: ADD PENALTIES TO THE MODEL FOR COMPLEXITY (L1, L2 REGULARIZATION).
# Pruning: REMOVE SOME OF THE MODEL'S COMPLEXITY (FOR DECISION TREES).
# Early Stopping: STOP TRAINING ONCE THE MODEL PERFORMANCE ON A VALIDATION SET STOPS IMPROVING.
# Ensemble Methods: USE TECHNIQUES LIKE BAGGING OR BOOSTING TO IMPROVE GENERALIZATION.
# THIS COMBINATION OF UNDERSTANDING DIFFERENT DATA STRUCTURES, DEPLOYMENT TECHNIQUES, MACHINE LEARNING CONC
```

```
# Question 50
# 1. Concept of Cross-Validation
# Cross-validation IS A TECHNIQUE USED TO ASSESS HOW THE RESULTS OF A STATISTICAL ANALYSIS WILL GENERALIZE

# K-Fold Cross-Validation:

# SPLIT THE DATA INTO K EQUAL PARTS (FOLDS).
# TRAIN THE MODEL ON K-1 FOLDS AND TEST IT ON THE REMAINING FOLD.
# REPEAT THIS K TIMES, EACH TIME WITH A DIFFERENT TEST FOLD.
# AVERAGE THE PERFORMANCE METRICS TO GET A MORE RELIABLE ESTIMATE OF THE MODEL'S PERFORMANCE.
# Leave-One-Out Cross-Validation (LOOCV):

# A SPECIAL CASE OF K-FOLD WHERE K IS EQUAL TO THE NUMBER OF OBSERVATIONS.
# USE ONE OBSERVATION AS TEST DATA AND THE REST AS TRAINING DATA.
# REPEAT FOR EACH OBSERVATION.
# 2. Classification vs. Regression Problem
# Classification:

# PREDICTS CATEGORICAL LABELS OR CLASSES.
# EXAMPLE: EMAIL SPAM DETECTION (SPAM OR NOT SPAM).
# Regression:

# PREDICTS CONTINUOUS NUMERICAL VALUES.
# EXAMPLE: PREDICTING HOUSE PRICES BASED ON FEATURES LIKE SIZE AND LOCATION.
# 3. Concept of Ensemble Learning
# Ensemble Learning INVOLVES COMBINING MULTIPLE MODELS TO IMPROVE PERFORMANCE. THE IDEA IS THAT A GROUP OF

# Bagging:

# TRAIN MULTIPLE MODELS IN PARALLEL ON DIFFERENT SUBSETS OF DATA.
# EXAMPLE: RANDOM FOREST.
# Boosting:

# TRAIN MODELS SEQUENTIALLY, WHERE EACH MODEL CORRECTS THE ERRORS OF THE PREVIOUS ONE.
# EXAMPLE: ADABOOST, GRADIENT BOOSTING.
# Stacking:

# TRAIN MULTIPLE MODELS AND THEN TRAIN A META-MODEL ON THEIR PREDICTIONS TO MAKE FINAL PREDICTIONS.
# 4. Gradient Descent
# Gradient Descent IS AN OPTIMIZATION ALGORITHM USED TO MINIMIZE THE LOSS FUNCTION OF A MODEL BY UPDATING M

# HOW IT WORKS:
# Initialize: START WITH RANDOM PARAMETERS.
# Compute Gradient: CALCULATE THE GRADIENT OF THE LOSS FUNCTION WITH RESPECT TO EACH PARAMETER.
# Update Parameters: ADJUST PARAMETERS BY SUBTRACTING A PROPORTION OF THE GRADIENT (LEARNING RATE).
# Repeat: ITERATIVELY UPDATE PARAMETERS UNTIL CONVERGENCE.
# 5. Batch Gradient Descent vs. Stochastic Gradient Descent
# Batch Gradient Descent:

# UPDATES PARAMETERS AFTER COMPUTING THE GRADIENT USING THE ENTIRE DATA SET.
# Pros: MORE STABLE UPDATES.
# Cons: CAN BE SLOW FOR LARGE DATASETS.
# Stochastic Gradient Descent (SGD):

# UPDATES PARAMETERS AFTER COMPUTING THE GRADIENT FOR A SINGLE SAMPLE OR SMALL BATCH.
# Pros: FASTER AND CAN ESCAPE LOCAL MINIMA.
# Cons: NOISY UPDATES, WHICH CAN MAKE CONVERGENCE LESS STABLE.
# 6. Curse of Dimensionality
# Curse of Dimensionality REFERS TO THE PROBLEMS THAT ARISE WHEN ANALYZING AND ORGANIZING DATA IN HIGH-DIME

# ISSUES INCLUDE:
# Increased Sparsity: DATA BECOMES SPARSE AS DIMENSIONS INCREASE, MAKING IT HARD TO GENERALIZE.
# Distance Metrics: DISTANCES BETWEEN DATA POINTS BECOME LESS MEANINGFUL IN HIGH DIMENSIONS.
# Overfitting: MODELS MAY OVERFIT BECAUSE OF THE HIGH NUMBER OF FEATURES.
# 7. L1 vs. L2 Regularization
# L1 Regularization (LASSO):
```



```

# Penalty: SUM OF ABSOLUTE VALUES OF PARAMETERS.
# Effect: CAN DRIVE SOME PARAMETERS TO ZERO, THUS PERFORMING FEATURE SELECTION.
# Regularization Term:  $\lambda * \sum |w_i|$ 
# L2 Regularization (Ridge):

# Penalty: SUM OF SQUARES OF PARAMETERS.
# Effect: TENDS TO SHRINK PARAMETERS TOWARDS ZERO BUT DOES NOT DRIVE THEM TO ZERO.
# Regularization Term:  $\lambda * \sum w_i^2$ 
# 8. Confusion Matrix
# Confusion Matrix IS A TABULAR REPRESENTATION OF A CLASSIFIER'S PERFORMANCE, SHOWING TRUE POSITIVES (TP),

# USED FOR:
# Evaluating: PERFORMANCE OF A CLASSIFICATION MODEL.
# Calculating Metrics: PRECISION, RECALL, F1 SCORE.
# 9. AUC-ROC Curve
# AUC-ROC Curve IS A GRAPHICAL REPRESENTATION OF A CLASSIFIER'S PERFORMANCE ACROSS ALL THRESHOLDS.

# ROC Curve: PLOTTING TRUE POSITIVE RATE (RECALL) AGAINST FALSE POSITIVE RATE.
# AUC (Area Under the Curve): MEASURES THE OVERALL PERFORMANCE OF THE CLASSIFIER. HIGHER AUC INDICATES BETT
# 10. k-Nearest Neighbors (k-NN) Algorithm
# k-NN IS A SIMPLE, INSTANCE-BASED LEARNING ALGORITHM.

# HOW IT WORKS:
# Classify a Point: BASED ON THE MAJORITY CLASS OF THE k NEAREST NEIGHBORS.
# Distance Metrics: EUCLIDEAN DISTANCE OR OTHER DISTANCE MEASURES.
# Parameter k: THE NUMBER OF NEAREST NEIGHBORS TO CONSIDER.
# 11. Support Vector Machine (SVM)
# SVM IS A CLASSIFICATION METHOD THAT SEEKS TO FIND THE HYPERPLANE THAT BEST SEPARATES CLASSES.

# HOW IT WORKS:
# Find the Hyperplane: THAT MAXIMIZES THE MARGIN BETWEEN CLASSES.
# Support Vectors: DATA POINTS NEAREST TO THE HYPERPLANE.
# 12. Kernel Trick in SVM
# Kernel Trick TRANSFORMS DATA INTO HIGHER DIMENSIONS TO MAKE IT LINEARLY SEPARABLE.

# HOW IT WORKS:
# Apply a Kernel Function: TO PROJECT DATA INTO A HIGHER DIMENSION.
# Example Kernels: POLYNOMIAL, RADIAL BASIS FUNCTION (RBF).
# 13. Types of Kernels in SVM
# Linear Kernel: USES A LINEAR SEPARATION.

# Polynomial Kernel: USES POLYNOMIAL FUNCTION FOR TRANSFORMATION.

# Radial Basis Function (RBF) Kernel: USES EXPONENTIAL FUNCTION BASED ON DISTANCE.

# Sigmoid Kernel: USES TANGENT HYPERBOLIC FUNCTION.

# WHEN TO USE:

# Linear: WHEN DATA IS LINEARLY SEPARABLE.
# Polynomial/RBF: FOR NON-LINEAR SEPARATION.
# 14. Hyperplane in SVM
# Hyperplane IS A DECISION BOUNDARY THAT SEPARATES DIFFERENT CLASSES IN FEATURE SPACE.

# Determined By:
# Support Vectors: NEAREST POINTS TO THE HYPERPLANE.
# Optimization: MAXIMIZES MARGIN BETWEEN CLASSES.
# 15. Pros and Cons of SVM
# Pros:

# EFFECTIVE IN HIGH-DIMENSIONAL SPACES.
# ROBUST TO OVERFITTING WITH A GOOD CHOICE OF PARAMETERS.
# Cons:

# MEMORY INTENSIVE.

```

```
# LESS EFFECTIVE ON LARGE DATASETS AND NOISE.
# 16. Hard Margin vs. Soft Margin SVM
# Hard Margin SVM:

# Strictly Separates CLASSES WITH NO MISCLASSIFICATION.
# Use When: CLASSES ARE LINEARLY SEPARABLE.
# Soft Margin SVM:

# ALLOW SOME MISCLASSIFICATION TO DEAL WITH NON-LINEAR SEPARABILITY.
# Use When: CLASSES ARE NOT PERFECTLY SEPARABLE.
# 17. Constructing a Decision Tree
# Decision Tree Construction INVOLVES SPLITTING THE DATA BASED ON FEATURE VALUES TO MINIMIZE IMPURITY.

# Steps:
# Select Feature: THAT BEST SPLITS THE DATA.
# Create Nodes: BASED ON SPLIT.
# Repeat: UNTIL LEAF NODES ARE HOMOGENEOUS OR STOPPING CRITERIA ARE MET.
# 18. Working Principle of a Decision Tree
# Decision Tree MAKES DECISIONS BY SPLITTING DATA INTO SUBSETS BASED ON FEATURE VALUES.

# Principle:
# Split Data: BASED ON FEATURE THAT MAXIMIZES INFORMATION GAIN OR MINIMIZES GINI IMPURITY.
# Terminal Nodes: REPRESENT FINAL DECISIONS OR CLASSIFICATIONS.
# 19. Information Gain in Decision Trees
# Information Gain MEASURES THE REDUCTION IN ENTROPY AFTER A SPLIT.

# USED FOR:
# Selecting Splits: BASED ON FEATURE'S ABILITY TO REDUCE UNCERTAINTY.
# Formula:  $IG = Entropy(Parent) - [Weighted\ Average] * Entropy(Children)$ 
# 20. Gini Impurity in Decision Trees
# Gini Impurity MEASURES THE DEGREE OF IMPURITY OR MIXTURE IN A NODE.

# USED FOR:
# Selecting Splits: CHOOSING FEATURES THAT MINIMIZE IMPURITY.
# Formula:  $Gini = 1 - \sum(p_i^2)$ , where  $p_i$  is the probability of class  $i$ .
# 21. Advantages and Disadvantages of Decision Trees
# Advantages:

# EASY TO UNDERSTAND AND INTERPRET.
# NO NEED FOR FEATURE SCALING.
# Disadvantages:

# CAN OVERFIT TO TRAINING DATA.
# SENSITIVE TO NOISE.
```

```

# How Random Forests Improve Upon Decision Trees
# Random Forests IMPROVE UPON DECISION TREES BY MITIGATING SOME OF THE LIMITATIONS OF SINGLE DECISION TREES

# Reduction of Overfitting:

# INDIVIDUAL DECISION TREES CAN OVERFIT TRAINING DATA. RANDOM FORESTS COMBINE MULTIPLE TREES TO REDUCE OVER
# Increased Accuracy:

# AGGREGATING PREDICTIONS FROM MULTIPLE TREES USUALLY LEADS TO MORE ACCURATE AND STABLE PREDICTIONS THAN A
# Robustness:

# RANDOM FORESTS ARE LESS SENSITIVE TO NOISE IN THE DATA COMPARED TO INDIVIDUAL DECISION TREES.
# 2. How Random Forest Algorithm Works
# Random Forest IS AN ENSEMBLE LEARNING METHOD THAT COMBINES MULTIPLE DECISION TREES TO IMPROVE PERFORMANCE

# Training Phase:

# Bootstrapping: CREATE MULTIPLE DATA SUBSETS BY RANDOMLY SAMPLING WITH REPLACEMENT FROM THE TRAINING DATA.
# Tree Construction: TRAIN A DECISION TREE ON EACH DATA SUBSET. AT EACH SPLIT, USE A RANDOM SUBSET OF FEATL
# Prediction Phase:

# Aggregation: FOR REGRESSION, AVERAGE THE PREDICTIONS OF ALL TREES. FOR CLASSIFICATION, USE VOTING (MAJORI
# 3. Bootstrapping in Random Forests
# Bootstrapping INVOLVES CREATING MULTIPLE TRAINING SUBSETS BY RANDOMLY SAMPLING WITH REPLACEMENT FROM THE

# Purpose:
# Diversity: ENSURES THAT EACH DECISION TREE IN THE FOREST IS TRAINED ON A SLIGHTLY DIFFERENT SET OF DATA,
# 4. Feature Importance in Random Forests
# Feature Importance MEASURES THE IMPACT OF EACH FEATURE ON THE PREDICTION OF THE MODEL.

# Concept:
# Gini Importance (Mean Decrease in Impurity): CALCULATES HOW MUCH EACH FEATURE REDUCES THE GINI IMPURITY A
# Mean Decrease in Accuracy: MEASURES HOW MUCH THE MODEL ACCURACY DECREASES WHEN THE FEATURE IS RANDOMLY PE
# 5. Key Hyperparameters of Random Forests
# Number of Trees (n_estimators):

# Impact: MORE TREES USUALLY IMPROVE PERFORMANCE BUT INCREASE COMPUTATION TIME.
# Maximum Depth (max_depth):

# Impact: CONTROLS THE MAXIMUM DEPTH OF EACH TREE. DEEPER TREES CAN CAPTURE MORE COMPLEX PATTERNS BUT MAY C
# Minimum Samples Split (min_samples_split):

# Impact: MINIMUM NUMBER OF SAMPLES REQUIRED TO SPLIT AN INTERNAL NODE. HIGHER VALUES PREVENT OVERFITTING.
# Minimum Samples Leaf (min_samples_leaf):

# Impact: MINIMUM NUMBER OF SAMPLES REQUIRED TO BE AT A LEAF NODE. HELPS PREVENT OVERFITTING.
# Maximum Features (max_features):

# Impact: NUMBER OF FEATURES TO CONSIDER WHEN LOOKING FOR THE BEST SPLIT. REDUCING THIS CAN IMPROVE MODEL C
# 6. Logistic Regression Model and Assumptions
# Logistic Regression IS A STATISTICAL METHOD FOR BINARY CLASSIFICATION.

# Model:

# PREDICTS THE PROBABILITY OF A CLASS LABEL BY ESTIMATING THE LOGIT FUNCTION.
# Assumptions:

# Linearity: LOGIT OF THE RESPONSE VARIABLE IS LINEAR IN THE PREDICTORS.
# Independence: OBSERVATIONS ARE INDEPENDENT OF EACH OTHER.
# No Multicollinearity: PREDICTORS ARE NOT HIGHLY CORRELATED.
# 7. Logistic Regression for Binary Classification
# Logistic Regression HANDLES BINARY CLASSIFICATION BY ESTIMATING THE PROBABILITY THAT AN INSTANCE BELONGS

# Model Output:
# USES THE SIGMOID FUNCTION TO MAP LINEAR COMBINATION OF FEATURES TO A PROBABILITY BETWEEN 0 AND 1.
# 8. Sigmoid Function in Logistic Regression

```

