```
# Q.1 What is the  difference between static and dynamic variable in python?
#  Python doesn't have static variables in the same way as languages like C++ or Java.

# Understanding the Misconception
# The concept of static variables is often associated with languages that support compile-time memory allo

# Dynamic Variables in Python
# In Python, all variables are dynamic. This means:

# Memory allocation: Variables are created at runtime when they are assigned a value.
# Data type: The type of a variable can change during program execution.
# Scope: Variables have a scope determined by the code block where they are defined (global, local, or no
# Lifetime: Variables exist until they are no longer referenced and are automatically garbage collected.
# Example
# Python
def my_function():
    x = 10  # Local variable, created at runtime
    print(x)

x = 20  # Global variable, created at runtime

my_function()
print(x)
```

```
10
20
```

```
# Q.2 Explain the purpose of "pop","popitem","clear()" in a dictionary with suitable examples.
# Understanding pop, popitem, and clear() in Python Dictionaries

# POP()
# The pop() method removes and returns an element from a dictionary based on the specified key.
my_dict = {'apple': 3, 'banana': 2, 'orange': 1}

# Remove and return the value for 'apple'
removed_value = my_dict.pop('apple')
print(removed_value)  # Output: 3
print(my_dict)  # Output: {'banana': 2, 'orange': 1}

# Remove 'grape' and return a default value if not found
value = my_dict.pop('grape', 'Key not found')
print(value)

# POPITEMS()
# The popitem() method removes and returns an arbitrary (key, value) pair from the dictionary. It's usefu


my_dict = {'apple': 3, 'banana': 2, 'orange': 1}

# Remove and return the value for 'apple'
removed_value = my_dict.pop('apple')
print(removed_value)  # Output: 3
print(my_dict)  # Output: {'banana': 2, 'orange': 1}

# Remove 'grape' and return a default value if not found
value = my_dict.pop('grape', 'Key not found')
print(value)  # Output: Key not found

# CLEAR()
# The clear() method removes all elements from a dictionary, leaving it empty.

#Python
my_dict = {'apple': 3, 'banana': 2, 'orange': 1}

my_dict.clear()
print(my_dict)  # Output: {}
```

```
3
{'banana': 2, 'orange': 1}
Key not found
3
{'banana': 2, 'orange': 1}
Key not found
{}
```

```
# Q.3 What do you mean by frozenSet? Explain it with suitable examples.
# Frozenset in Python
# A frozenset is an immutable version of a Python set. This means that once a frozenset is created, its el

# Key Characteristics:
# Immutable: Unchangeable after creation.
# Unordered: Elements have no specific order.
# Unique: No duplicate elements.

my_list = [1, 2, 3, 2, 4]
my_frozenset = frozenset(my_list)
print(my_frozenset)
```

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

```
# Q.4 Difference between mutable and immutable data types in python and give examples of mutable and immut
# Mutable vs Immutable Data Types in Python
# Mutable Data Types
# Mutable data types are those whose values can be changed after they are created. In other words, you can
# Examples of mutable data types:

# Lists:
my_list = [1, 2, 3]
my_list.append(4)  # Modifying the list
print(my_list)



# Immutable Data Types
# Immutable data types are those whose values cannot be changed after they are created. Any operation that

# Examples of immutable data types:

# Numbers (int, float, complex):

x = 10
x = x + 1  # Creates a new integer object
print(x)
```

```
[1, 2, 3, 4]
11
```

```
# Q.5 What is __init__? Explain with an examples.
# init in Python
# init is a special method in Python, often referred to as a constructor. It's automatically called when y

class Dog:
    def __init__(self, name, breed, age):
        self.name = name
        self.breed = breed
        self.age = age

    def bark(self):
        print(f"{self.name} barks!")

# Create objects
dog1 = Dog("Buddy", "Golden Retriever", 3)
dog2 = Dog("Max", "Labrador", 5)

# Access attributes and call methods
print(dog1.name)
dog2.bark()
```

```
Buddy
Max barks!
```

```
# Q.6 What is docstring in python ? Explain with an examples?
# Docstrings in Python
# Docstrings are strings that document a Python module, class, function, or method. They are placed as th

def add(x, y):
  """Adds two numbers and returns the sum."""
  return x + y

print(add.__doc__)
```

```
Adds two numbers and returns the sum.
```

```
# Q.8 What is break,continue and pass in python?

# The break :-
# statement is used to terminate the loop prematurely.
# When the break statement is encountered, the loop is immediately exited,
# and the control flow moves to the next statement after the loop.Break

for i in range(10):
    if i == 5:
        break
    print(i)

# Continue:-
# The continue statement is used to skip the rest of the current iteration of a loop
#  and move to the next iteration. The loop does not terminate,
#  but the code after the continue statement is not executed for the current iteration.

for i in range(5):
    if i == 2:
        continue
    print(i)

# Pass :-
# The pass statement is a null operation.
# It does nothing. It is often used as a placeholder when a statement
# is syntactically required but no code needs to be executed.

def function():
    pass
```

```
0
1
2
3
4
0
1
3
4
```

```
#. Q.10 What are global,protected and private attributes in python?

# Global, Protected, and Private Attributes in Python

# Unlike languages like Java or C++, Python doesn't have strict access modifiers like public, protected, a

# Global Attributes:-
# Defined outside any class or function.
# Accessible from anywhere in the code.

global_var = "I am a global variable"

class MyClass:
    def __init__(self):
        print(global_var)

# Protected Attributes
# Indicated by a single underscore prefix (_).
# Intended for internal use within the class and its subclasses.
# Conventionally not accessed from outside the class.

class MyClass:
    def __init__(self):
        self._protected_attr = "I am protected"

    def get_protected_attr(self):
        return self._protected_attr

# Private Attributes
# Indicated by a double underscore prefix (__).
# Intended for exclusive use within the class.
# Python uses name mangling to prevent direct access from outside the class.

class MyClass:
    def __init__(self):
        self.__private_attr = "I am private"

    def get_private_attr(self):
        return self.__private_attr
```

```
# Q.15 What are decorators in python? Explain it with an example.Write down its use cases.
# Decorators in Python
# A decorator is a function that takes another function as an argument, adds some functionality, and retu

def decorator_function(original_function):
    def wrapper_function(*args, **kwargs):
        print("This is the decorator function before the original function")
        result = original_function(*args, **kwargs)
        print("This is the decorator function after the original function")
        return result
    return wrapper_function

@decorator_function
def my_function(x, y):
    print(f"x is {x} and y is {y}")
    return x + y

result = my_function(3, 4)
print(result)
```

```
This is the decorator function before the original function
x is 3 and y is 4
This is the decorator function after the original function
7
```

```
# Q.17 What is lambda in python? why is it used?
# Lambda Functions in Python
# Lambda functions are anonymous functions, meaning they don't have a specific name. They are defined usi

add = lambda x, y: x + y
result = add(3, 4)
print(result)
```

```
7
```

```
# Q.18 Explain split() and join() function in python?
# split() and join() in Python
# split():-
# The split() method in Python is used to break a string into a list of substrings based on a specified de

text = "This is a sample string"
words = text.split()  # Split by whitespace
print(words)  # Output: ['This', 'is', 'a', 'sample', 'string']

numbers = "1,2,3,4,5"
number_list = numbers.split(",")  # Split by comma
print(number_list)

# join():-
# The join() method is used to join the elements of an iterable (like a list or tuple) into a single stri

words = ["hello", "world"]
joined_string = " ".join(words)  # Join with space
print(joined_string)  # Output: hello world

number_list = ["1", "2", "3"]
joined_numbers = "-".join(number_list)  # Join with hyphen
print(joined_numbers)
```

```
['This', 'is', 'a', 'sample', 'string']
['1', '2', '3', '4', '5']
hello world
1-2-3
```

```
# Q.19 What are iterators,iterable & generators in python?

#(1)  Iterables
# An iterable is any object that can be iterated over. This means you can use a for loop to go through its
my_list = [1, 2, 3]  # A list is iterable
for num in my_list:
    print(num)

#(2)Iterators
# An iterator is an object that implements the iterator protocol. It has two methods: __iter__() and __ne

my_list = [1, 2, 3]
my_iterator = iter(my_list)

print(next(my_iterator))
print(next(my_iterator))
print(next(my_iterator))

# Generators
# Generators are a special type of iterator created using the yield keyword. They provide a convenient way

def my_generator():
    yield 1
    yield 2
    yield 3

for num in my_generator():
    print(num)
```

```
1
2
3
1
2
3
1
2
3
```

```
#. Q.20 What is the difference between xrange and range in python?

# Feature           range()         xrange()
# Return type           List            Iterator-like object
# Memory usage        High            Low
# Performance          Slower             Faster
# Availability     Python 2 and 3       Python 2 only
```

```
#. Q.21 pillars of Oops.

# Pillars of Object-Oriented Programming (OOP)
# Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of "objects

# 1. Encapsulation
# Encapsulation is the bundling of data (attributes) and methods (functions) that operate on the data wit

# 2. Abstraction
# Abstraction focuses on the essential features of an object, hiding unnecessary implementation details.

# 3. Inheritance
# Inheritance allows you to create new classes (derived classes or subclasses) based on existing classes

# 4. Polymorphism
# Polymorphism means "many forms". It allows objects of different types to be treated as if they were of
```

```
# Q.22 How will you check if a class is child  of another class?
# Checking if a Class is a Child of Another Class in Python
#Python provides the issubclass() function to determine if a class is a subclass of another class.
class Animal:
    pass

class Dog(Animal):
    pass

class Cat(Animal):
    pass

class GoldenRetriever(Dog):
    pass

print(issubclass(Dog, Animal))  # Output: True
print(issubclass(Cat, Dog))  # Output: False
print(issubclass(GoldenRetriever, Animal))
```

```
True
False
True
```

```
#. Q.23 How does inheritance work in python ? Explain all types of inheritance with an example.

# Inheritance in Python:-

# Inheritance is a fundamental concept in object-oriented programming (OOP) that allows you to create new

# How it Works:
# A derived class inherits the attributes and methods of the base class.
# The derived class can add new attributes and methods or override existing ones.
# The super() function is used to access methods of the parent class.

# Types of Inheritance:

# 1. Single Inheritance:-
# A child class inherits from only one parent class.

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

    def speak(self):
        print("Animal speaking")

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def speak(self):
        print("Woof!")

# 2. Multiple Inheritance:
# A child class inherits from more than one parent class.

class Flyer:
    def fly(self):
        print("Flying")

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

class FlyingFish(Flyer, Swimmer):
  pass


# 3. Multilevel Inheritance:
# A child class inherits from a parent class, which in turn inherits from another parent class.

class Grandfather:
    pass

class Father(Grandfather):
    pass

class Son(Father):
    pass

# 4. Hierarchical Inheritance:
# Multiple child classes inherit from a single parent class.

class Animal:
    pass

class Dog(Animal):
    pass

class Cat(Animal):
    pass

#5. Hybrid Inheritance:
# A combination of two or more types of inheritance.

class Animal:
    pass

class Mammal(Animal):
    pass

class Fish(Animal):
    pass

class Bat(Mammal, Flyer):
    pass


# Q.25 What is polymorphism? Explain its with an example.

# Understanding Polymorphism
# Imagine you have a function that expects an animal object. You want to call a make_sound() method on th

class Animal:
    def make_sound(self):
        print("Generic animal sound")

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

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


def animal_sound(animal):
    animal.make_sound()


# Create objects
dog = Dog()
cat = Cat()

# Call the function with different animal objects
animal_sound(dog)  # Output: Woof!
animal_sound(cat)  # Output: Meow!
```

```
Woof!
Meow!
```

```
# Q.27 name=["mohan","dash","karam","chandra","gandhi","bapu"] do the following operations in the list;
# a) add an  element "freedom_fighter" in the list at the 0 index

name = ["mohan", "dash", "karam", "chandra", "gandhi", "bapu"]

# Add "freedom_fighter" at index 0
name.insert(0, "freedom_fighter")

print(name)
```

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

```
# Q.28 name=["mohan","dash","karam","chandra","gandhi","bapu"] do the following operations in the list;

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

name=["freedomfighter","Bapuji","mohan","dash","karam","chandra","gandhi"]
length1=len((name[-len(name)+1:-1:2]))
length2=len((name[-len(name)+1:-1]))
print(length1+length2)
```

```
8
```

```
# Q.28 name=["mohan","dash","karam","chandra","gandhi","bapu"] do the following operations in the list;
# c) add two more elements in the name["Netaji","bose"] at the end of the list.

name = ["mohan", "dash", "karam", "chandra", "gandhi", "bapu"]

# Add two more elements to the end of the list
name.extend(["Netaji", "bose"])

print(name)
```

```
['mohan', 'dash', 'karam', 'chandra', 'gandhi', 'bapu', 'Netaji', 'bose']
```

```
# Q.28 name = ["mohan", "dash", "karam", "chandra", "gandhi", "bapu"]

# d) What will be the value of temp:

name=["bapuji","dash","karam","chandra","gandhi","mohan"]
temp=name[-1]
name[-1]=name[0]
name[0]=temp
print(name)
```

```
['mohan', 'dash', 'karam', 'chandra', 'gandhi', 'bapuji']
```

```
# Q.29 Find the output of the following
animal = ['human','cat','mat','cat','rat','human','lion']
print(animal.count("human"))
print(animal.index("rat"))
print(len(animal))
```

```
2
4
7
```

```
# Q.30 Tuple=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22)[{"roll_no":1},{"name":"navneet"}]
# a) print(len(Tuple))

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "na

print(len(Tuple))
```

```
8
```

```
# Q.30 Tuple=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22)[{"roll_no":1},{"name":"navneet"}]

# b) print(Tuple[-1][-1]["name"])

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "na

print(Tuple[-1][-1]["name"])
```

```
navneet
```

```
# Q.30 Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"nam

# C) fetch the values of roll_no from this tuple.

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "na

# Access the last element of the tuple, which is a list of dictionaries
list_of_dicts = Tuple[-1]

# Extract the roll_no value from the first dictionary
roll_no = list_of_dicts[0]["roll_no"]

print(roll_no)
```

```
1
```

```
# Q.30 Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"nam

# d) print(Tuple[-3][1])
Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "na

print(Tuple[-3][1])
```

```
ji
```

```
# Q.30 Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"nam

#  e) fetch the element '22' from this tuple.

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "na

result = Tuple[6][2]
print(result)  # This will output: 22
```

```
22
```

```
# Q.31 Write a program to display the appropriate message as per the colour of signal(red-stop/yellow-stay

def display_signal_message(signal_color):
    # Convert the input to lowercase to handle different cases
    signal_color = signal_color.lower()

    # Determine the message based on the signal color
    if signal_color == 'red':
        return "Stop"
    elif signal_color == 'yellow':
        return "Stay"
    elif signal_color == 'green':
        return "Go"
    else:
        return "Invalid color! Please enter 'red', 'yellow', or 'green'."

def main():
    # Prompt the user to enter the color of the signal
    signal_color = input("Enter the color of the traffic signal (red, yellow, green): ")

    # Display the appropriate message
    message = display_signal_message(signal_color)
    print(message)

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

```
Enter the color of the traffic signal (red, yellow, green): RED
Stop
```

```
# Q.32 Write a program to create a simple calculator performing only four basic operations(+,-,/,*).
def calculator():
  """Performs basic arithmetic operations."""

  while True:
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Divide")
    print("4. Multiply")
    print("5. Exit")

    choice = input("Enter choice (1/2/3/4/5): ")

    if choice in ('1', '2', '3', '4'):
      num1 = float(input("Enter first number: "))
      num2 = float(input("Enter second number: "))

      if choice == '1':
        print(num1, "+", num2,"=", num1 + num2)
      elif choice == '2':
        print(num1, "-", num2, "=", num1 - num2)
      elif choice == '3':
        if num2 == 0:
          print("Error: Division by zero")
      else:
          print(num1, "/", num2, "=", num1 / num2)
    elif choice == '4':
      print(num1, "*", num2, "=", num1 * num2)

    elif choice == '5':
      break

    else:
      print("Invalid input")

calculator()
```

```
Select operation:
1. Add
2. Subtract
3. Divide
4. Multiply
5. Exit
25.0 + 25.0 = 50.0
Select operation:
1. Add
2. Subtract
3. Divide
4. Multiply
5. Exit
```

```
# Q.33 Write a program to find the larger of the three pre-specified numbers using ternary operators.

num1 = 10
num2 = 25
num3 = 15

largest = num1 if (num1 >= num2 and num1 >= num3) else (num2 if num2 >= num3 else num3)

print("The largest number is:", largest)
```

```
The largest number is: 25
```

```
# Q.34 Write a program to find the factor of whole number using a while loop.
def find_factors(number):
  """Finds the factors of a given number using a while loop.

  Args:
    number: The number to find factors for.

  Returns:
    A list of factors of the number.
  """

  factors = []
  divisor = 1

  while divisor <= number:
    if number % divisor == 0:
      factors.append(divisor)
    divisor += 1

  return factors

# Get input from the user
num = int(input("Enter a whole number: "))

# Find and print the factors
factors = find_factors(num)
print("Factors of", num, "are:", factors)
```

```
Enter a whole number: 5
Factors of 5 are: [1, 5]
```

```
# Q.35 Write a program to find the sum of all the positive numbers entered by the user. As soon as the us

def sum_positive_numbers():
  """Calculates the sum of positive numbers entered by the user."""

  total = 0
  number = 0

  while number >= 0:
    number = int(input("Enter a positive number (or a negative number to stop): "))
    if number >= 0:
      total += number

  print("The sum of the positive numbers is:", total)

sum_positive_numbers()
```

```
Enter a positive number (or a negative number to stop): 5
Enter a positive number (or a negative number to stop): -2
The sum of the positive numbers is: 5
```

```
# Q.36 Write a program to find prime numbers between 2 to 100 using nested for loops.

def sum_positive_numbers():
  """Calculates the sum of positive numbers entered by the user."""

  total = 0
  number = 0

  while number >= 0:
    number = int(input("Enter a positive number (or a negative number to stop): "))
    if number >= 0:
      total += number

  print("The sum of the positive numbers is:", total)

sum_positive_numbers()
```

```
Enter a positive number (or a negative number to stop): 2
Enter a positive number (or a negative number to stop): 4
Enter a positive number (or a negative number to stop): 6
Enter a positive number (or a negative number to stop): -5
The sum of the positive numbers is: 12
```

```
# Q.36 Write a program for the following.
# a) accept the marks of the student in five major subject and display the same.
#  Criteria                                       Grade
  # percentage > 85                                   A
  # percentage <85 & percentage >=75                  B
  # percentage <75 & percentage >=50                  C
  # percentage >30 & percentage <=50                  D
  # percentage <30                                    E

def calculate_grade(marks):
  """Calculates and displays the grade based on the given marks.

  Args:
    marks: A list of marks in five subjects.
  """

  total_marks = sum(marks)
  percentage = (total_marks / 500) * 100

  print("Total Marks:", total_marks)
  print("Percentage:", percentage)

  if percentage > 85:
    grade = "A"
  elif percentage >= 75:
    grade = "B"
  elif percentage >= 50:
    grade = "C"
  elif percentage >= 30:
    grade = "D"
  else:
    grade = "E"

  print("Grade:", grade)

# Get marks from the user
subjects = ["English", "Maths", "Science", "Social Science", "Hindi"]
marks = []
for subject in subjects:
  mark = int(input(f"Enter marks for {subject}: "))
  marks.append(mark)

# Calculate and display grade
calculate_grade(marks)
```

```
Enter marks for English: 5
Enter marks for Maths: 5
Enter marks for Science: 5
Enter marks for Social Science: 5
Enter marks for Hindi: 8
Total Marks: 28
Percentage: 5.6000000000000005
Grade: E
```

```
# Q.36 Write a program for the following.
 # b) Calculate the sum of the marks of all subject .Divide the total marks of the numbers  of subject (i
# Criteria                                      Grade
  # percentage > 85                                   A
  # percentage <85 & percentage >=75                  B
  # percentage <75 & percentage >=50                  C
  # percentage >30 & percentage <=50                  D
  # percentage <30                                    E

def calculate_grade():
  """Calculates and displays the percentage and grade based on given marks."""

  subjects = ["English", "Maths", "Science", "Social Science", "Hindi"]
  marks = []

  for subject in subjects:
    mark = int(input(f"Enter marks for {subject}: "))
    marks.append(mark)

  total_marks = sum(marks)
  percentage = (total_marks / 500) * 100

  print("Total Marks:", total_marks)
  print("Percentage:", percentage)

  if percentage > 85:
    grade = "A"
  elif percentage >= 75:
    grade = "B"
  elif percentage >= 50:
    grade = "C"
  elif percentage >= 30:
    grade = "D"
  else:
    grade = "E"

  print("Grade:", grade)

calculate_grade()
```

```
Enter marks for English: 15
Enter marks for Maths: 25
Enter marks for Science: 69
Enter marks for Social Science: 85
Enter marks for Hindi: 35
Total Marks: 229
Percentage: 45.800000000000004
Grade: D
```

```
# Q.36 Write a program for the following.
 # c) find the grade of the student as per the following criteria. Hint use match & case for this.
# Criteria                                      Grade
  # percentage > 85                                   A
  # percentage <85 & percentage >=75                  B
  # percentage <75 & percentage >=50                  C
  # percentage >30 & percentage <=50                  D
  # percentage <30                                    E

def calculate_grade(percentage):
  """Calculates the grade based on the given percentage using match-case.

  Args:
    percentage: The student's percentage.

  Returns:
    The grade corresponding to the percentage.
  """

  match percentage:
    case percentage if percentage > 85:
      return "A"
    case percentage if percentage >= 75:
      return "B"
    case percentage if percentage >= 50:
      return "C"
    case percentage if percentage >= 30:
      return "D"
    case _:
      return "E"

def main():
  subjects = ["English", "Maths", "Science", "Social Science", "Hindi"]
  marks = []

  for subject in subjects:
    mark = int(input(f"Enter marks for {subject}: "))
    marks.append(mark)

  total_marks = sum(marks)
  percentage = (total_marks / 500) * 100

  print("Total Marks:", total_marks)
  print("Percentage:", percentage)

  grade = calculate_grade(percentage)
  print("Grade:", grade)

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

```
Enter marks for English: 88
Enter marks for Maths: 85
Enter marks for Science: 75
Enter marks for Social Science: 69
Enter marks for Hindi: 84
Total Marks: 401
Percentage: 80.2
Grade: B
```

```
# Q.37 Write a program for VIBGYOR Spectrum based on their Wavelength using wavelength range:
#  COLOUR                            Wavelength(nm)
#  violet                            400 to 440
#  indigo                            440 to 460
#  blue                              460 to 500
#  green                             500 to 570
#  yellow                            570 to 590
#  orange                            590 to 620
#  red                               620 to 720

def get_color_from_wavelength(wavelength):
  """Determines the color based on the given wavelength.

  Args:
    wavelength: The wavelength of the light in nanometers.

  Returns:
    The color corresponding to the wavelength, or 'Unknown' if not in range.
  """

  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 'Unknown'

# Example usage:
wavelength = float(input("Enter the wavelength in nanometers: "))
color = get_color_from_wavelength(wavelength)
print("The color is:", color)
```

```
Enter the wavelength in nanometers: 570
The color is: yellow
```

```
# Q.38 Consider the gravitational interactions between the earth,moon,and sun in our solar system.given:-
# mass_earth=5.972e24 # mass of earth in kilograms
# mass_moon=7.34767309e22 # mass of moon in kilograms
# mass_sun=1.989e30 # mass of sun in kilograms

# distance_earth_sun=1.496e11 # distance between earth and sun in meters
# distance_moon_earth=3.844e8 # distance between moon and earth in meters

#  a) Calculate the gravitational force between the earth and the sun.

import math

# Given values
G = 6.67430e-11  # Gravitational constant
mass_earth = 5.972e24
mass_sun = 1.989e30
distance_earth_sun = 1.496e11

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

print("Gravitational force between Earth and Sun:", force_earth_sun, "N")
```

```
Gravitational force between Earth and Sun: 3.5423960813684973e+22 N
```

```
# Q.38 Consider the gravitational interactions between the earth,moon,and sun in our solar system.given:-
# mass_earth=5.972e24 # mass of earth in kilograms
# mass_moon=7.34767309e22 # mass of moon in kilograms
# mass_sun=1.989e30 # mass of sun in kilograms

# distance_earth_sun=1.496e11 # distance between earth and sun in meters
# distance_moon_earth=3.844e8 # distance between moon and earth in meters

# b) Calculate the gravitational force between the moon and the earth.

import math

# Given values
G = 6.67430e-11  # Gravitational constant
mass_earth = 5.972e24
mass_moon = 7.34767309e22
distance_moon_earth = 3.844e8

# Calculate the gravitational force
force_moon_earth = G * (mass_earth * mass_moon) / (distance_moon_earth**2)

print("Gravitational force between Moon and Earth:", force_moon_earth, "N")
```

```
Gravitational force between Moon and Earth: 1.9820225456526813e+20 N
```

```
# Q.39 Design and implement a python program for managing student information using object-oriented princ
# a) Define the `student` class with encapsulated attributes.

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

    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):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def get_roll_number(self):
        return self.__roll_number

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
```

```
# Q.39 Design and implement a python program for managing student information using object-oriented princ
# b) Implement getter and setter methods for the attributes.

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

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

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

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
```

```
# Q.39 Design and implement a python program for managing student information using object-oriented princ
# c) Write methods to display student information and update details.

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

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

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

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number

    # Display student information
    def display_student_info(self):
        print("Name:", self.__name)
        print("Age:", self.__age)
        print("Roll Number:", self.__roll_number)

    # Update student details
    def update_student_details(self, new_name, new_age, new_roll_number):
        self.set_name(new_name)
        self.set_age(new_age)
        self.set_roll_number(new_roll_number)
```

```
#  Q.39 Design and implement a python program for managing student information using object-oriented prin
#  d) Create instance of the `student` class and test the implemented functionality

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

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

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

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number

    # Display student information
    def display_student_info(self):
        print("Name:", self.__name)
        print("Age:", self.__age)
        print("Roll Number:", self.__roll_number)

    # Update student details
    def update_student_details(self, new_name, new_age, new_roll_number):
        self.set_name(new_name)
        self.set_age(new_age)
        self.set_roll_number(new_roll_number)

# Create instances of the Student class
student1 = Student("Alice", 20, 12345)
student2 = Student("Bob", 22, 54321)

# Test the functionality
student1.display_student_info()

print("\nUpdating student1 details...")
student1.update_student_details("Alice Updated", 21, 12346)
student1.display_student_info()

print("\nAccessing student2 details using getters:")
print("Name:", student2.get_name())
print("Age:", student2.get_age())
print("Roll Number:", student2.get_roll_number())
```

```
Name: Alice
Age: 20
Roll Number: 12345

Updating student1 details...
Name: Alice Updated
Age: 21
Roll Number: 12346

Accessing student2 details using getters:
Name: Bob
```

```
Age: 22
Roll Number: 54321
```

```
# Q.40 Develop a python program for managing library resources efficiently. design a class named `LibraryI
# a) Create the `LibraryBook` class with encaptulated attributes.

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

    def get_book_name(self):
        return self.__book_name

    def get_author(self):
        return self.__author

    def is_available(self):
        return self.__is_available
```

```
# Q.40 Develop a python program for managing library resources efficiently. design a class named `LibraryI
# b) Implement methods for borrowing and returning books

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

    def get_book_name(self):
        return self.__book_name

    def get_author(self):
        return self.__author

    def is_available(self):
        return self.__is_available

    def borrow_book(self):
        if self.__is_available:
            self.__is_available = False
            print(f"{self.__book_name} by {self.__author} has been borrowed.")
        else:
            print(f"{self.__book_name} is currently unavailable.")

    def return_book(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"{self.__book_name} by {self.__author} has been returned.")
        else:
            print(f"{self.__book_name} is already available.")
```

```
# Q.40 Develop a python program for managing library resources efficiently. design a class named `Library
# c) Ensure proper encapsulation to protect book details.

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

    @property
    def book_name(self):
        return self.__book_name

    @property
    def author(self):
        return self.__author

    def is_available(self):
        return self.__is_available

    def borrow_book(self):
        if self.__is_available:
            self.__is_available = False
            print(f"{self.__book_name} by {self.__author} has been borrowed.")
        else:
            raise ValueError("Book is already borrowed")

    def return_book(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"{self.__book_name} by {self.__author} has been returned.")
        else:
            raise ValueError("Book is already available")
```

```
# Q.40 Develop a python program for managing library resources efficiently. design a class named `Library
# d) Test the borrowing and returning functionality with sample data.
class LibraryBook:
    def __init__(self, book_name, author):
        self.__book_name = book_name
        self.__author = author
        self.__availability_status = True  # True indicates the book is available

    def borrow_book(self):
        if self.__availability_status:
            self.__availability_status = False
            print(f"You have borrowed '{self.__book_name}' by {self.__author}.")
        else:
            print(f"Sorry, '{self.__book_name}' is currently unavailable.")

    def return_book(self):
        if not self.__availability_status:
            self.__availability_status = True
            print(f"Thank you for returning '{self.__book_name}'.")
        else:
            print(f"'{self.__book_name}' was not borrowed, so it cannot be returned.")

    def get_book_info(self):
        status = "Available" if self.__availability_status else "Unavailable"
        return f"Book: {self.__book_name}, Author: {self.__author}, Status: {status}"

# Testing the borrowing and returning functionality with sample data

book1 = LibraryBook("1984", "George Orwell")
book2 = LibraryBook("To Kill a Mockingbird", "Harper Lee")

# Display initial status
print(book1.get_book_info())
print(book2.get_book_info())

# Borrow books
book1.borrow_book()
book2.borrow_book()

# Try borrowing the same book again
book1.borrow_book()

# Display status after borrowing
print(book1.get_book_info())
print(book2.get_book_info())

# Return books
book1.return_book()
book2.return_book()

# Try returning the same book again
book1.return_book()

# Display status after returning
print(book1.get_book_info())
print(book2.get_book_info())
```

```
Python Crash Course by Eric Matthes has been borrowed.
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-30-2635146aa9fd> in <cell line: 41>()
     39 # Test borrowing and returning
     40 book1.borrow_book()
---> 41 book1.borrow_book()  # Should raise an error
     42 book1.return_book()
     43 book2.borrow_book()

<ipython-input-30-2635146aa9fd> in borrow_book(self)
     24             print(f"{self.__book_name} by {self.__author} has been borrowed.")
     25         else:
---> 26             raise ValueError("Book is already borrowed")
     27 
     28     def return_book(self):

ValueError: Book is already borrowed
```

```
# Q.41 Create a simple banking system using object-oriented concepts in python.Design classes representin
# a) Define base class (es) for bank accounts with common attributes and methods.

class BankAccount:
  def __init__(self, account_number, balance=0.0):
    self.account_number = account_number
    self.balance = balance

  def deposit(self, amount):
    if amount > 0:
      self.balance += amount
    print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
      print("Invalid deposit amount.
 Please enter a positive value.")

  def withdraw(self, amount):
    if amount > 0 and amount <= self.balance:
      self.balance -= amount
      print(f"Withdrew ${amount:.2f}.
 New balance: ${self.balance:.2f}")
    else:
          if amount <= 0:
          print("Invalid withdrawal amount. Please enter a positive value.")
    else:
        print(f"Insufficient funds. Available balance: ${self.balance:.2f}")

  def get_balance(self):
    print(f"Your current balance is: ${self.balance:.2f}")
```

```
  File "<ipython-input-37-a1bd77bfef2c>", line 14
    print("Invalid deposit amount.
          ^
SyntaxError: unterminated string literal (detected at line 14)
```

```
# Q.41 Create a simple banking system using object-oriented concepts in python.Design classes representing
# b) Implement subclasses for specific account types(e.g.,savingAccount,CheckingAccount).

class BankAccount:
  def __init__(self, account_number, balance=0.0):
    self.account_number = account_number
    self.balance = balance

  def deposit(self, amount):
    if amount > 0:
      self.balance += amount
      print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
      print("Invalid deposit amount.Please enter a positive value.")

  def withdraw(self, amount):
    if amount > 0 and amount <= self.balance:
      self.balance -= amount
      print(f"Withdrew ${amount:.2f}.New balance: ${self.balance:.2f}")
    else:

      if amount <= 0:
        print("Invalid withdrawal amount. Please enter a positive value.")
      else:
        print(f"Insufficient funds. Available balance: ${self.balance:.2f}")

  def get_balance(self):
    print(f"Your current balance is: ${self.balance:.2f}")


class SavingsAccount(BankAccount):
  def __init__(self, account_number, balance=0.0, interest_rate=0.01):
    super().__init__(account_number, balance)  # Call base class constructor
    self.interest_rate = interest_rate

  # Additional method specific to SavingsAccount
  def calculate_interest(self):
    interest = self.balance * self.interest_rate
    print(f"Interest earned: ${interest:.2f}")


class CheckingAccount(BankAccount):
  def __init__(self, account_number, balance=0.0, overdraft_limit=0.0):
    super().__init__(account_number, balance)  # Call base class constructor
    self.overdraft_limit = overdraft_limit

  # Override the withdraw method to consider overdraft limit
  def withdraw(self, amount):
    if amount > 0 and (amount <= self.balance + self.overdraft_limit):
      self.balance -= amount
      print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
      if amount <= 0:
        print("Invalid withdrawal amount. Please enter a positive value.")
      else:
        print(f"Insufficient funds. Available balance: ${self.balance:.2f} + Overdraft limit: ${self.overdr
# Q.41 Create a simple banking system using object-oriented concepts in python.Design classes representin
# d) Test the banking system by creating instance of different accounts types and performing transactions

class BankAccount:
  def __init__(self, account_number, balance=0.0):
    self.account_number = account_number
    self.balance = balance

  def deposit(self, amount):
    if amount > 0:
      self.balance += amount
      print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
      print("Invalid deposit amount.Please enter a positive value.")

  def withdraw(self, amount):
    if amount > 0 and amount <= self.balance:
      self.balance -= amount
      print(f"Withdrew ${amount:.2f}.New balance: ${self.balance:.2f}")
    else:

      if amount <= 0:
        print("Invalid withdrawal amount. Please enter a positive value.")
      else:
        print(f"Insufficient funds. Available balance: ${self.balance:.2f}")

  def get_balance(self):
    print(f"Your current balance is: ${self.balance:.2f}")


class SavingsAccount(BankAccount):
  def __init__(self, account_number, balance=0.0, interest_rate=0.01):
    super().__init__(account_number, balance)  # Call base class constructor
    self.interest_rate = interest_rate

  # Additional method specific to SavingsAccount
  def calculate_interest(self):
    interest = self.balance * self.interest_rate
    print(f"Interest earned: ${interest:.2f}")


class CheckingAccount(BankAccount):
  def __init__(self, account_number, balance=0.0, overdraft_limit=0.0):
    super().__init__(account_number, balance)  # Call base class constructor
    self.overdraft_limit = overdraft_limit

  # Override the withdraw method to consider overdraft limit
  def withdraw(self, amount):
    if amount > 0 and (amount <= self.balance + self.overdraft_limit):
      self.balance -= amount
      print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
      if amount <= 0:
        print("Invalid withdrawal amount. Please enter a positive value.")
      else:
        print(f"Insufficient funds. Available balance: ${self.balance:.2f} + Overdraft limit: ${self.over


# Create test accounts
savings = SavingsAccount("SAV1234", 1000, 0.05)  # 5% interest rate
checking = CheckingAccount("CHK5678", 500, 100)  # $100 overdraft limit

# Test transactions
print("\n**Savings Account Transactions**")
savings.deposit(200)
savings.withdraw(150)
savings.get_balance()
savings.calculate_interest()  # Specific to SavingsAccount
```

```
print("\n**Checking Account Transactions**")
checking.deposit(100)
checking.withdraw(700)  # Test overdraft limit
checking.withdraw(50)
checking.get_balance()
```

```
**Savings Account Transactions**
Deposited $200.00. New balance: $1200.00
Withdrew $150.00.New balance: $1050.00
Your current balance is: $1050.00
Interest earned: $52.50

**Checking Account Transactions**
Deposited $100.00. New balance: $600.00
Withdrew $700.00. New balance: $-100.00
Insufficient funds. Available balance: $-100.00 + Overdraft limit: $100.00
Your current balance is: $-100.00
```

```
# Q.42 Write a python program that models different animals and their sounds. design a base class called
# b) Create a subclasses `dog` and `cat` that override the `make_sound()`method.

class Animal:
    def make_sound(self):
        print("Generic animal sound")

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

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

```
# Q.42 Write a python program that models different animals and their sounds. design a base class called
# c) Implement the  sound  generation logic for each subclass.

class Animal:
    def make_sound(self):
        print("Generic animal sound")

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

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

```
# Q.42 Write a python program that models different animals and their sounds. design a base class called
# d) Test the program  by creating instance of  `Dog` and `cat` and  calling the `make_sound()`method.

class Animal:
    def make_sound(self):
        print("Generic animal sound")

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

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


# Create instances of Dog and Cat
dog = Dog()
cat = Cat()

# Call make_sound() method for each instance
dog.make_sound()  # Output: Woof!
cat.make_sound()  # Output: Meow!
```

```
Woof!
Meow!
```

```
# Q.43 Write a code for restaurent management system using Oops.
# a) Create a menu_item class that has attributes such as name,description,price and category.

class MenuItem:
    def __init__(self, name, description, price, category):
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - ${self.price:.2f}"
```

```
# Q.43 Write a code for restaurent management system using Oops.
# b) Implement methods to add a new menu_item,update menu item information, and remove a menu item from tl

class MenuItem:
    def __init__(self, name, description, price, category):
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - ${self.price:.2f}"

class Menu:
    def __init__(self):
        self.menu_items = []

    def add_item(self, menu_item):
        self.menu_items.append(menu_item)

    def update_item(self, item_name, new_description, new_price, new_category):
        for item in self.menu_items:
            if item.name == item_name:
                item.description = new_description
                item.price = new_price
                item.category = new_category
                break

    def remove_item(self, item_name):
        for item in self.menu_items:
            if item.name == item_name:
                self.menu_items.remove(item)
                break
```

```
# Q.43 Write a code for restaurent management system using Oops.
# c) use encapsulation to hide the menu item's unique identification number.

class MenuItem:
    def __init__(self, name, description, price, category, item_id):
        self.name = name
        self.description = description
        self.price = price
        self.category = category
        self.__item_id = item_id  # Encapsulated attribute

    def get_item_id(self):
        return self.__item_id

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - ${self.price:.2f}"

class Menu:
    def __init__(self):
        self.menu_items = []
        self.next_item_id = 1

    def add_item(self, name, description, price, category):
        item_id = self.next_item_id
        self.next_item_id += 1
        menu_item = MenuItem(name, description, price, category, item_id)
        self.menu_items.append(menu_item)
        return menu_item

    def update_item(self, item_id, new_description, new_price, new_category):
        for item in self.menu_items:
            if item.get_item_id() == item_id:
                item.description = new_description
                item.price = new_price
                item.category = new_category
                break

    def remove_item(self, item_id):
        for item in self.menu_items:
            if item.get_item_id() == item_id:
                self.menu_items.remove(item)
                break

    def get_menu_items(self):
        return self.menu_items
```

```
# Q.43 Write a code for restaurent management system using Oops.
# d) inherit  from the menu_item class to create a fooditem class and a Beaverageitem class,each with the

class MenuItem:
    def __init__(self, name, description, price, category, item_id):
        self.name = name
        self.description = description
        self.price = price
        self.category = category
        self.__item_id = item_id  # Encapsulated attribute

    def get_item_id(self):
        return self.__item_id

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - ${self.price:.2f}"

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

class BeverageItem(MenuItem):
    def __init__(self, name, description, price, category, item_id, size):
        super().__init__(name, description, price, category, item_id)
        self.size = size
```

```
# Q.44 Write a code for hotel management system using Oops.
# a) Create a room class that has attributes such as room number,room type,rate, and availability(private)

class Room:
    def __init__(self, room_number, room_type, rate):
        self.room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True  # Private attribute

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability
```

```
# Q.44 Write a code for hotel management system using Oops.
# b) Implement methods to book  a room,check in a guest,and check out a guest.

class Room:
    def __init__(self, room_number, room_type, rate):
        self.room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability

class Hotel:
    def __init__(self):
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number and room.is_available():
                room.set_availability(False)
                print(f"Room {room_number} booked successfully.")
                return room
        else:
            print("Room not available.")
            return None

    def check_in(self, guest_name, room):
        if not room.is_available():
            print(f"Guest {guest_name} checked in to room {room.room_number}.")

    def check_out(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number and not room.is_availability():
                room.set_availability(True)
                print(f"Guest checked out from room {room_number}.")
                return
        else:
            print("Room not occupied.")
```

```
# Q.44 Write a code for hotel management system using Oops.
# c) use encapsulation to hide the room's unique identification number.

class Room:
    def __init__(self, room_number, room_type, rate):
        self.__room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True

    def get_room_number(self):
        return self.__room_number

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability

class Hotel:
    def __init__(self):
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, room_number):
        for room in self.rooms:
            if room.get_room_number() == room_number and room.is_available():
                room.set_availability(False)
                print(f"Room {room_number} booked successfully.")
                return room
        else:
            print("Room not available.")
            return None

    def check_in(self, guest_name, room):
        if not room.is_available():
            print(f"Guest {guest_name} checked in to room {room.get_room_number()}.")

    def check_out(self, room_number):
        for room in self.rooms:
            if room.get_room_number() == room_number and not room.is_availability():
                room.set_availability(True)
                print(f"Guest checked out from room {room_number}.")
                return
        else:
            print("Room not occupied.")
```

```
# Q.44 Write a code for hotel management system using Oops.
# d) Inherit from the room class to create a suitroom class and a standardroom class,each with their own
class Room:
    def __init__(self, room_number, room_type, rate):
        self.__room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True

    def get_room_number(self):
        return self.__room_number

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability

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

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

```
# Q.45 Write a Code for event management system using Oops.
# a) Create an event class that has attributes such as name,date,time,location and list of attendees(priva

class Event:
    def __init__(self, name, date, time, location):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)

    def get_attendees(self):
        return self.__attendees
```

```
# Q.45 Write a Code for event management system using Oops.
# b) Implement methods to create a new event,add or remove attendees, and get the total numbers of attende

class Event:
    def __init__(self, name, date, time, location):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)

    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)

    def get_attendees(self):
        return self.__attendees

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

```
# Q.45 Write a Code for event management system using Oops.
# c) use encapsulation to hide the events unique identification number.

class Event:
    def __init__(self, name, date, time, location, event_id):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []
        self.__event_id = event_id

    def get_event_id(self):
        return self.__event_id

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)

    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)

    def get_attendees(self):
        return self.__attendees

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

```
# Q.45 Write a Code for event management system using Oops.
# d) Inherit from the event class to create a privateevent class and a public event class ,each with thei

class Event:
    def __init__(self, name, date, time, location, event_id):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []
        self.__event_id = event_id

    def get_event_id(self):
        return self.__event_id

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)

    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)

    def get_attendees(self):
        return self.__attendees

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

class PrivateEvent(Event):
    def __init__(self, name, date, time, location, event_id, invitees):
        super().__init__(name, date, time, location, event_id)
        self.invitees = invitees

class PublicEvent(Event):
    def __init__(self, name, date, time, location, event_id, registration_fee):
        super().__init__(name, date, time, location, event_id)
        self.registration_fee = registration_fee
```

```
# Q.46 Write a code for airline reservation system using Oops.
# a) Create a flight class that has attributes such as flight number,departure and arrival airports,depar

class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time):
        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 = 150  # Assuming 150 seats as default

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False
```

```
# Q.46 Write a code for airline reservation system using Oops.
# b) Implement methods to book a seat,cancel a reservation , and get the remaining available seats.

class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time):
        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 = 150  # Assuming 150 seats as default

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False

    def cancel_seat(self):
        if self.__available_seats < 150:
            self.__available_seats += 1
            return True
        else:
            return False
```

```
# Q.46 Write a code for airline reservation system using Oops.
# c) Use encapsulation to hide the flight's uique  identification number.
class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time):
        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 = 150  # Assuming 150 seats as default

    def get_flight_number(self):
        return self.__flight_number

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False

    def cancel_seat(self):
        if self.__available_seats < 150:
            self.__available_seats += 1
            return True
        else:
            return False
```

```
# Q.46 Write a code for airline reservation system using Oops.
# d) Inherit from the flight class to create a domesticflight class and an international flight class,eacl

class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time):
        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 = 150  # Assuming 150 seats as default

    def get_flight_number(self):
        return self.__flight_number

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False

    def cancel_seat(self):
        if self.__available_seats < 150:
            self.__available_seats += 1
            return True
        else:
            return False

class DomesticFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time, m
        super().__init__(flight_number, departure_airport, arrival_airport, departure_time, arrival_time)
        self.meal_type = meal_type

class InternationalFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time, v
        super().__init__(flight_number, departure_airport, arrival_airport, departure_time, arrival_time)
        self.visa_required = visa_required


# Q.47 Define a python module  named constants.pycontaining constants like pi and the speed of light.
import constants

print(constants.pi)
print(constants.speed_of_light)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-90-5143486a197e> in <cell line: 2>()
      1 # Q.47 Define a python module  named constants.pycontaining constants like pi and the speed of light.
----> 2 import constants
      3 
      4 print(constants.pi)
      5 print(constants.speed_of_light)

ModuleNotFoundError: No module named 'constants'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

```
# Q.48 Write a python module named calculator.py containing functions for additions,subtractions,multiplic
import calculator
```

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-91-1143fba2ff0f> in <cell line: 2>()
      1 # Q.48 Write a python module named calculator.py containing functions for additions,subtractions,multiplication and
division.
----> 2 import calculator
      3
      4 result = calculator.add(5, 3)
      5 print(result)

ModuleNotFoundError: No module named 'calculator'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

```
# Q.49 Implement a python package structure for a project named ecommerce,containing modules for  product

# Implementing a Python Package Structure for an Ecommerce Project

# Understanding the Structure

# We'll create a package named ecommerce with two subpackages: product_management and order_processing.

# Directory Structure
# ecommerce/
# ├── __init__.py
# ├── product_management/
# │   ├── __init__.py
# │   ├── product.py
# │   └── product_category.py
# ├── order_processing/
# │   ├── __init__.py
# │   ├── order.py
# │   └── order_item.py
# └── tests/
#     ├── __init__.py
#     ├── test_product.py
#     └── test_order.py
# Q.50 Implement a python module named string_utils.py containing functions for string manipulation such a

def reverse_string(text):
  """Reverses the order of characters in a string.

  Args:
      text: The string to be reversed.

  Returns:
      A new string with the characters in reversed order.
  """
  return text[::-1]

def capitalize_first(text):
  """Capitalizes the first letter of a string and returns the new string.

  Args:
      text: The string to be capitalized.

  Returns:
      A new string with the first letter capitalized.
  """
  return text.capitalize()

def is_palindrome(text):
  """Checks if a string is a palindrome (reads the same backward as forward).

  Args:
      text: The string to be checked.

  Returns:
      True if the string is a palindrome, False otherwise.
  """
  text = text.lower().replace(" ", "")  # Case-insensitive and remove spaces
  return text == text[::-1]

def slugify(text):
  """Converts a string to a slug (lowercase, spaces replaced with hyphens).

  Args:
      text: The string to be converted to a slug.

  Returns:
      A new string in lowercase with spaces replaced by hyphens.
  """
  return text.lower().replace(" ", "-")
```

```
# Q.51 Write a python module  named file_operations.py with functions for reading,writing and  appending

def read_file(filepath):
  """
  Reads the content of a file and returns it as a string.

  Args:
      filepath (str): The path to the file to be read.

  Returns:
      str: The content of the file, or None if an error occurs.
  """
  try:
    with open(filepath, 'r') as file:
      return file.read()
  except FileNotFoundError:
    print(f"Error: File not found: {filepath}")
    return None

def write_file(filepath, content):
  """
  Writes the provided content to a file.

  Args:
      filepath (str): The path to the file to be written to.
      content (str): The content to be written to the file.
  """
  try:
    with open(filepath, 'w') as file:
      file.write(content)
    print(f"Successfully wrote to file: {filepath}")
  except (IOError, OSError) as error:
    print(f"Error writing to file: {filepath} - {error}")

def append_to_file(filepath, content):
  """
  Appends the provided content to a file.

  Args:
      filepath (str): The path to the file to be appended to.
      content (str): The content to be appended to the file.
  """
  try:
    with open(filepath, 'a') as file:
      file.write(content)
    print(f"Successfully appended to file: {filepath}")
  except (IOError, OSError) as error:
    print(f"Error appending to file: {filepath} - {error}")
```

```
# Q.52 Write a python program to create a text file named 'employees.txt' and write the details of employe

def handle_file_operation(filepath, content, mode='w'):
  """
  Reads, writes, or appends content to a file based on the mode.

  Args:
      filepath (str): The path to the file.
      content (str): The content to write or append.
      mode (str, optional): The file open mode ('r' for read, 'w' for write, 'a' for append). Defaults to

  Returns:
      str: The content of the file (for read mode) or None on error.
      Exception: The specific exception encountered during the operation (if any).
  """
  try:
    with open(filepath, mode) as file:
      if mode == 'r':
        return file.read()
      else:
        file.write(content)
        return None  # Or return success message
  except FileNotFoundError:
    return f"Error: File not found: {filepath}"
  except (PermissionError, UnicodeDecodeError) as error:
    return error  # Return specific exception

# Usage (Read)
file_content = handle_file_operation("myfile.txt")
if isinstance(file_content, str):
  print(file_content)
else:
  print(file_content)  # Handle specific error returned

# Usage (Write)
write_result = handle_file_operation("newfile.txt", "This is new content.")
if not isinstance(write_result, Exception):
  print("Successfully wrote to file.")
else:
  print(write_result)  # Handle specific error returned
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-98-818a8948bc43> in <cell line: 29>()
     27 
     28 # Usage (Read)
---> 29 file_content = handle_file_operation("myfile.txt")
     30 if isinstance(file_content, str):
     31   print(file_content)

TypeError: handle_file_operation() missing 1 required positional argument: 'content'
```

```
# Q.53 Develop a python  script that opens  an existing text file named 'inventory.txt' in read mode and
def read_inventory_file(file_path):
  """Reads the content of an inventory file line by line.

  Args:
      file_path (str): The path to the inventory file.
  """

  try:
      with open(file_path, 'r') as file:
          for line in file:
              print(line.strip())
  except FileNotFoundError:
      print(f"Error: Inventory file '{file_path}' not found.")

# Example usage:
file_path = 'inventory.txt'
read_inventory_file(file_path)
```

```
Error: Inventory file 'inventory.txt' not found.
```

```
# Q.54 Create a python script that reads a text file named 'expenses.txt' and calculate the total amount

 def calculate_total_expenses(file_path):
  """Calculates the total expenses from a text file.

  Args:
      file_path (str): The path to the expenses file.

  Returns:
      float: The total amount spent.
  """

  total_expenses = 0.0
  try:
    with open(file_path, 'r') as file:
      for line in file:
        amount = float(line.strip())
        total_expenses += amount
  except FileNotFoundError:
    print(f"Error: Expenses file '{file_path}' not found.")
  except ValueError:
    print("Error: Invalid data format in expenses file.")
  return total_expenses

# Example usage:
file_path = 'expenses.txt'
total_amount = calculate_total_expenses(file_path)
print("Total expenses:", total_amount)
```

```
# Q.55 Create a python program that reads a text file named  'paragraph.txt' and  counts the occurences o

import collections

def count_word_occurrences(file_path):
  """Counts the occurrences of each word in a text file.

  Args:
      file_path (str): The path to the text file.
  """

  word_counts = collections.defaultdict(int)

  try:
    with open(file_path, 'r') as file:
      for line in file:
        words = line.split()
        for word in words:
          word_counts[word.lower()] += 1
  except FileNotFoundError:
    print(f"Error: File not found: {file_path}")
  else:
    for word, count in sorted(word_counts.items()):
      print(f"{word}: {count}")

# Example usage:
file_path = 'paragraph.txt'
count_word_occurrences(file_path)
```

```
Error: File not found: paragraph.txt
```

```
# Q.60  Calculate coefficient  of correlation between the marks obtained by 10 students in accountancy an
# STUDENT           ACCOUNTANCY             STATISTICS
#   1                   45                     35
#   2                   70                     90
#   3                   65                     70
#   4                   30                     40
#   5                   90                     95
#   6                   40                     40
#   7                   50                     60
#   8                   75                     80
#   9                   85                     80
#   10                  60                     50

# Use karl pearson's coefficient of correlation method to find it.

# Understanding the Data
# We have the marks of 10 students in Accountancy and Statistics.

# Formula for Karl Pearson's Coefficient of Correlation
# The formula for Karl Pearson's Coefficient of Correlation (r) is:

# r = (nΣxy - ΣxΣy) / sqrt((nΣx^2 - (Σx)^2)(nΣy^2 - (Σy)^2))

# Where:

# n = number of observations (in this case, 10)
# Σxy = sum of the product of corresponding values of x and y
# Σx = sum of x values (Accountancy marks)
# Σy = sum of y values (Statistics marks)
# Σx^2 = sum of the squares of x values
# Σy^2 = sum of the squares of y values

# Calculations
# Let's create a table to calculate the necessary values:
STUDENT     ACCOUNTANCY (x)     STATISTICS (y)      xy          x^2          y^2
1                   45                35               1575         2025          1225
2                   70                90                6300          4900        8100
3                   65                70                4550          4225        4900
4                   30                40                1200          900           1600
5                   90                95                8550          8100        9025
6                   40                40                1600          1600        1600
7                   50                60                3000          2500        3600
8                   75                80                6000          5625        6400
9                   85                80                6800          7225        6400
10                60                  50                3000          3600        2500
Total               600               640               44775         40700       45350

# From the table:

Σx = 600
Σy = 640
Σxy = 44775
Σx^2 = 40700
Σy^2 = 45350
n = 10
Now, substitute these values into the formula:

r = (10 * 44775 - 600 * 640) / sqrt((10 * 40700 - 600^2)(10 * 45350 - 640^2))
```

```
  File "<ipython-input-9-01e17a5fae4c>", line 35
    STUDENT        ACCOUNTANCY (x)        STATISTICS (y)          xy          x^2             y^2
                   ^
SyntaxError: invalid syntax
```

```
# Q.63 In a partially destroyed laboratory record  of an analysis of correlation data,the following resul
# a) what are the mean value of x and y.

# Understanding the Problem
# We have two regression equations:
# 40x-18y=214
# 8x-10y = -66
# We also know that the variance of x (σ²x) is 9.

# Our goal is to find the mean values of x (x̄) and y (ȳ).

# Solution
# Key point: The point of intersection of the two regression lines is the point (x̄, ȳ).

# Steps:

# Convert the given equations into the standard form of regression lines:
y = bxy * x + a
x = byx * y + a
#Find the values of bxy and byx from the given equations.
# Use the formula for the intersection point (x̄, ȳ) of the two regression lines:
x̄ = (a1b2 - a2b1) / (b1b2 - 1)
ȳ = (b1a2 - b2a1) / (b1b2 - 1)
# Where:

# a1 and b1 are the constants in the regression equation for y on x.
# a2 and b2 are the constants in the regression equation for x on y.
# Calculations:

# Rewrite the equations in standard form:
y = (4/5)x + 6.6
x = (9/10)y + 5.35
# Identify the values:
bxy = 4/5
byx = 9/10
a1 = 6.6
a2 = 5.35
# Calculate the means:
x̄ = ((6.6 * 9/10) - (5.35 * 4/5)) / ((4/5 * 9/10) - 1)
ȳ = ((4/5 * 5.35) - (9/10 * 6.6)) / ((4/5 * 9/10) - 1)
```

```
  File "<ipython-input-21-2fb788bc4edf>", line 31
    y = (4/5)x + 6.6
             ^
SyntaxError: invalid syntax
```

```
# Q.63 In a partially destroyed laboratory record  of an analysis of correlation data,the following resul
# b) the coefficient of correlation between x and y.

# Understanding the Problem
# We have the regression equations:

8x - 10y = -66
40x - 18y = 214
# We also know the variance of x (σ²x) = 9.

# Our goal is to find the coefficient of correlation (r).

# Solution
# Key point: The coefficient of correlation (r) is the square root of the product of the regression coeff

# Steps:

# Convert the given equations into the standard form of regression lines:
y = bxy * x + a
x = byx * y + a
# Find the values of bxy and byx from the given equations.
# Calculate the coefficient of correlation (r) using the formula:
# r = sqrt(bxy * byx)
# Calculations:
# We already found bxy and byx in the previous part:

bxy = 4/5
byx = 9/10
# Now, calculate r:

r = sqrt((4/5) * (9/10))
r = sqrt(36/50)
r = 6/5
# Therefore, the coefficient of correlation (r) between x and y is 6/5.
```

```
  File "<ipython-input-22-e3d3870aad7e>", line 7
    8x - 10y = -66
    ^
SyntaxError: invalid decimal literal
```

```
# Q.64 What is the normal distribution? what are the 4 main assumption of normal distribution?explain its

# Normal Distribution:-

# The normal distribution, often referred to as the Gaussian distribution or bell curve, is a probability

#  It is characterized by its bell shape, with the majority of data points clustered around the mean and

#  Four Main Assumptions of Normal Distribution:-

# Continuity: The data is continuous, meaning it can take on any value within a given range.

# Symmetry: The distribution is symmetrical around the mean.

# Unimodality: There is only one peak in the distribution, representing the most frequent value.

# Finite Variance: The distribution has a finite variance, indicating that the data points are not infini
```

```
#. Q.66 The mean of a distribution is 60 with a standard deviation of 10.Assuming that the distribution i
# (i) between 60 and 72
# (ii) between 50 and 60
# (iii) beyond 72
# (iv) between 70 and 80

# Q.68 if the height of 500 students are normally distributed with  mean 65 inch and standard deviation 5
```