1. The Product class is given below. Display the namespace (value of the _dict_ attribute) of this class as shown below.
Expected result:
```
__module__
__init__
__repr__
get_id
__dict__
__weakref__
__doc__
```

```python
import uuid
class Product:
    def __init__(self, product_name, price):
        self.product_id = self.get_id()
        self.product_name = product_name
        self.price = price
    def __repr__(self):
        return f"Product(product_name='{self.product_name}', price={self.price})"
    @staticmethod
    def get_id():
        return str(uuid.uuid4().fields[-1])[:6]
```

2.Implement a function called stick( ) that takes any number of bare arguments and return an object of type str being a concatenation of all arguments of type str passed to the function with the '#' sign (see below).
Example:
```
[IN]: stick('sport', 'summer', 4, True)
[OUT]: 'sport#summer’
```
As an answer call the stick( ) function in the following ways (print the result to the console):
• stick('sport', 'summer')
• stick(3, 5, 7)
stick(False, 'time'. True, 'workout', [], 'gym')
Expected result:
```
    Sport#sumer
    time#workout#gym
```

3. The implementation of the Vehicle class is given:
class Vehicle:
This is a Vehicle class.
Display the value of the name attribute of the Vehicle class to the console.
Expected result:
Vehicle

4. Define a simple class named Model. Then create an instance of this class named model.
Using the built-in function isinstance( ) check if the model is an instance of the Model class. Print the result to the console.

5.Implement a class named Phone. In the Phone class, define two class attributes with names:
• brand
• model
and set their values to:
• 'Apple'
• ‘iPhone X'
Then use the built-in functions getattr() and print () to display the values of the given attributes of the Phone class to the console as shown below.

6.A class named OnlineShop was defined with the class attributes set accordingly:
• Sector to the Value 'electronics'
• sector_code to the value 'ele'
• is_public_company to the value False
Using the del statement remove the class attribute named sector_code. In response, print the rest of the userdefined OnlineShop class attribute names as a list as shown below.
Expected result:
```
[ ' sector',‘is_public_company’]
```

7.Implement the HouseProject class with class attributes respectively:
• number_of_floors = 3
• area = 100
Then, in the HouseProject class implement a function (class callable attribute) called describe_project( ), which displays basic information about the project as follows:
```
Floor number: 3
Area: 100
```

8. The Book class is defined. A list books_data is also given.
```
    books_data = [
    {'author': 'Dan Brown', 'title': 'Inferno'},
    {'author': 'Dan Brown', 'title': 'The Da Vinci Code', 'year_of_publishnent': 2003}
    ]
```
Based on this data, create two instances of the Book class, where the instance attributes will be the keys from
the given dictionaries (books_data list) with their corresponding values.
In response, print the _dict_ attributes of the objects to the console as shown below.
Expected result:
```
    {'author': 'Dan Brown', 'title': 'Inferno'}
    {'author': 'Dan Brown', 'title': 'The Da Vinci Code', 'year_of_publishment': 2003}
```

9.A class called Laptop was implemented.
Implement a method in the Laptop class called display_attrs_with_values() , which displays the names of all
the attributes of the Laptop class with their values as shown below (attribute name -> attribute value).
Then create an instance named laptop with the following values:
• brand = 'Dell'
model = 'Inspiron'
price = 3699
In response, call display_attrs_with_values( ) method on the laptop instance.
Expected result:
```
    brand - Dell
    model - Inspiron
    price -3699
```

10.Implement a class called Laptop that sets the following instance
attributes when creating an instance:
• brand
• model
• price
When creating an instance, add validation for the price attribute. The value
of the price attribute must be an int or float type greater than zero. If it is
not, raise the TypeError with the following message:The price attribute must be a positive int or float
Then create an instance called laptop with the given attributes:
• brand = 'Acer'
• model = 'Predator'
• price = 5490
In response, print the value of the _dict_ attribute of the laptop instance.
Expected result:
```
{'brand ' :'Acer', 'model': 'Predator', 'price': 5490}
```

11. Implement a class called Laptop that sets the following instance attributes
when creating an instance:
• brand as a bare instance attribute
• model as a protected attribute
• price as a private attribute
12.Implement a class called Laptop which in the init ( ) method sets the value of
the price protected attribute that stores the price of the laptop (without any validation).
Then implement a method to read that attribute named get_price() and a method
to modify that attribute named set_price( ) without validation as well.
Then create an instance of the Laptop class with a price of 3499 and follow these
steps:
• using the get_price( ) method print the value of the price protected attribute to
the console
• using the set_price( ) method, set the value of the price protected attribute to
3999
• using the get_price( ) method print the value of the price protected attribute to the console
Expected result:

```
3499
3999
```

12. You are given a list of student dictionaries, where each dictionary contains information about a student’s name, grades, and attendance. Write a Python function that filters out students who meet the following criteria:

    1.Have an average grade below 75.
    2.Have attended less than 80% of classes.

The function should return a new list of dictionaries with only the students who meet both criteria, along with their calculated average grades.

```
students = [
    {"name": "Alice", "grades": [80, 90, 70], "attendance": 0.9},
    {"name": "Bob", "grades": [60, 65, 70], "attendance": 0.85},
    {"name": "Charlie", "grades": [95, 100, 92], "attendance": 0.78},
    {"name": "David", "grades": [55, 60, 65], "attendance": 0.9},
    ]
```

13. Create a function greet_students that greets a variable number of students by their names. The function should:

    1.Take greeting, which specifies the greeting message must be first word (e.g., "Hello").
    2.accept any number of names.
    3.Optionally take a keyword argument, capitalize, which, when set to True, capitalizes each name in the greeting.

The function should print the greeting followed by each student's name. If capitalize is True, print names in uppercase.

```
# Expected output:
     Welcome ALICE
     Welcome BOB
     Welcome CHARLIE
```

14. Write a function order_summary that calculates the total cost of an order after applying a discount, if specified, and then prints any additional details provided about the order.

The function should:
    1.Take order_items, a list of item prices, as a required positional argument.
    2.Optionally accept a discount percentage (e.g., 10 for 10% off).
    3.If additional information is provided (such as customer_name, address, or delivery_date), print each one in the format: Key: Value.

    The function should print:
    The total cost after applying the discount.
    Each piece of extra information on a new line if provided.

```
    order_items = [50, 100, 25]
     Expected output:
         Total after discount: 157.5
         Customer Name: John Doe
         Address: 123 Elm St
```

15. Write a list comprehension that takes a list of numbers and returns a list of squares for only the even numbers.
16. Write a function sum_above_threshold that takes a list of numbers and a threshold. It should return the sum of numbers that are greater than the threshold
17. Write a function merge_dictionaries that takes two dictionaries and merges them. If they have the same keys,
    the values should be added (assuming they are integers)
18. Given a dictionary of item names and prices, write a function sort_items_by_price that returns a list of tuples sorted by price in descending order.
19. Write a function rotate_list that rotates a given list to the right by a specified number of steps. The rotation should be done in such a way that
    the elements that move past the end of the list wrap around to the beginning.

```
    rotate_list([1, 2, 3, 4, 5], 2)
    # Output: [4, 5, 1, 2, 3]

    rotate_list([1, 2, 3, 4, 5], -3)
    # Output: [3, 4, 5, 1, 2]

    rotate_list([1, 2, 3], 0)
    # Output: [1, 2, 3]
```

20. Write a function count_char_frequencies that takes a string and returns a

```
    dictionary with the frequency count of each character in the string,
        ignoring spaces and considering uppercase and lowercase characters as the same.
        example:
        count_char_frequencies("Hello World")
        # Output: {'h': 1, 'e': 1, 'l': 3, 'o': 2, 'w': 1, 'r': 1, 'd': 1}

```