Wrocław University of Science and Technology

PYTHON
LABORATORY REPORT

Faculty of Electronics, Photonics and Microsystems

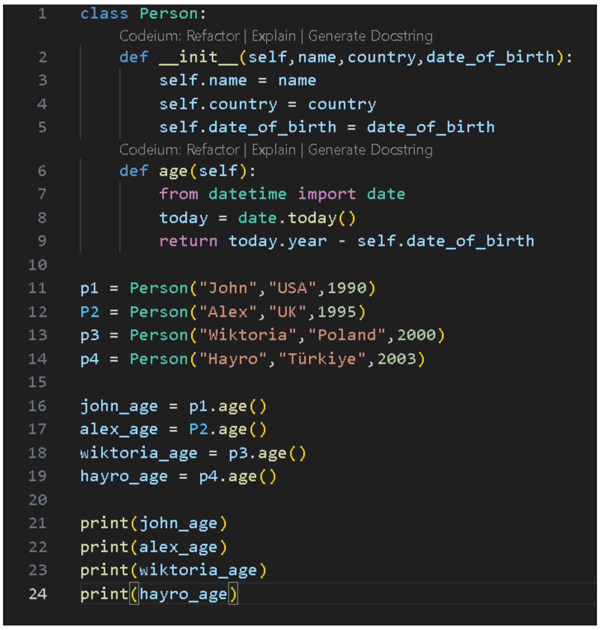
Theme of class: Classes

Student: Hayrettin Aycetin (276807)
Date of class: 20.11.2023 15:15-16:55
Group No:3
Submition Date:1.12.2023

Lab assistant: Aleksander Kubeczek, Alicja Kwaśny GRADE:

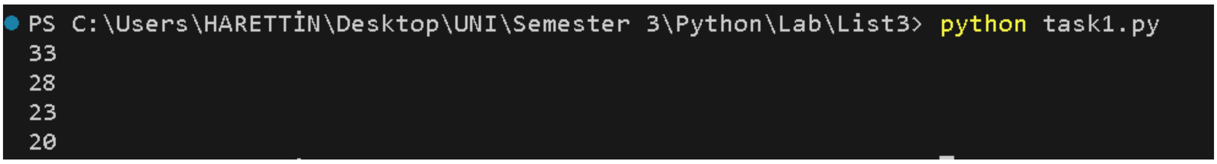
Task 1

Write a Python program to create a person class. Include attributes like name, country and date of birth. Implement a method to calculate the person's age.



```python
class Person:
    def __init__(self,name,country,date_of_birth):
        self.name = name
        self.country = country
        self.date_of_birth = date_of_birth
    def age(self):
        from datetime import date
        today = date.today()
        return today.year - self.date_of_birth

p1 = Person("John","USA",1990)
P2 = Person("Alex","UK",1995)
p3 = Person("Wiktoria","Poland",2000)
p4 = Person("Hayro","Türkiye",2003)

john_age = p1.age()
alex_age = P2.age()
wiktoria_age = p3.age()
hayro_age = p4.age()

print(john_age)
print(alex_age)
print(wiktoria_age)
print(hayro_age)
```

Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring



```
PS C:\Users\HARETTİN\Desktop\UNI\Semester 3\Python\Lab\List3> python task1.py
33
28
23
20
```

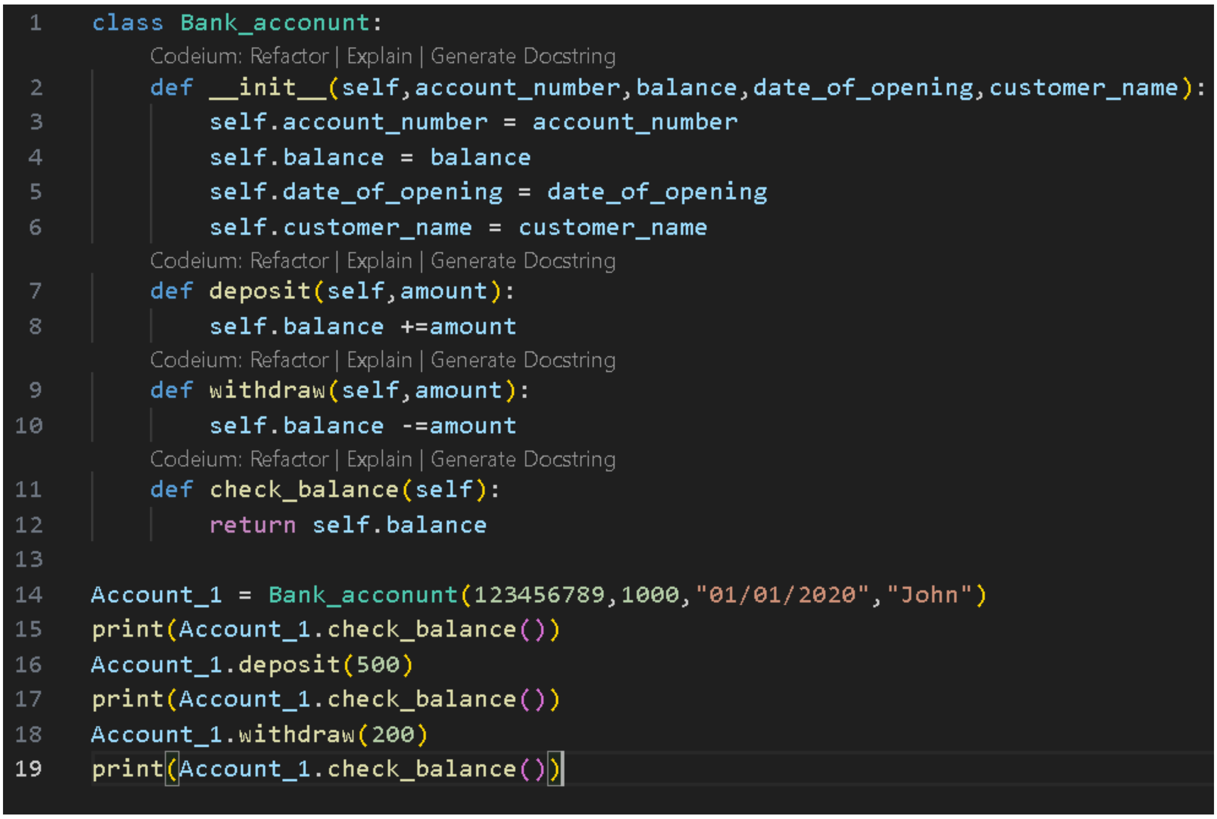
Comments:
Firstly, I defined a class named "Person" in Python. Using the class constructor __init__, I took parameters such as name, country, and date of birth. Subsequently, I assigned these parameters to corresponding attributes – name, country, and date of birth – using self for better readability.

Next, I implemented a method called "age" within the class. This method calculates the age of the person by utilizing the datetime library.

Finally, I instantiated four objects of the "Person" class and calculated their ages.

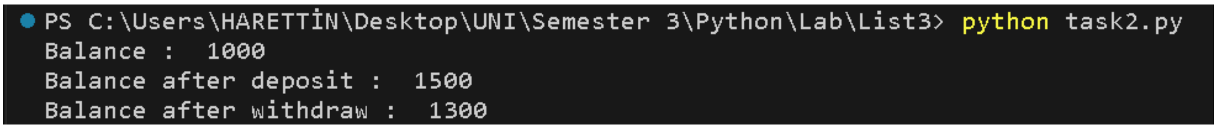
Task 2

Write a Python class Bank Account with attributes like account _ number, balance, date_ of_ opening and customer _ name, and methods like deposit, withdraw, and check _ balance.



```python
class Bank_acconunt:
    def __init__(self,account_number,balance,date_of_opening,customer_name):
        self.account_number = account_number
        self.balance = balance
        self.date_of_opening = date_of_opening
        self.customer_name = customer_name
    def deposit(self,amount):
        self.balance +=amount
    def withdraw(self,amount):
        self.balance -=amount
    def check_balance(self):
        return self.balance

Account_1 = Bank_acconunt(123456789,1000,"01/01/2020","John")
print(Account_1.check_balance())
Account_1.deposit(500)
print(Account_1.check_balance())
Account_1.withdraw(200)
print(Account_1.check_balance())
```

Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring



```
PS C:\Users\HARETTİN\Desktop\UNI\Semester 3\Python\Lab\List3> python task2.py
Balance :  1000
Balance after deposit :  1500
Balance after withdraw :  1300
```

Comments:
created a basic class named BankAccount in Python. In the class constructor (__init__), I defined parameters such as account_number, balance, date_of_opening, and customer_name. These parameters were then assigned to their corresponding attributes.

Within the class, I implemented three methods:

check_balance: This method returns the current balance of the account.

deposit: To deposit money, I added a parameter named amount to specify the amount to be added to the account.

withdraw: For withdrawing money, I included the amount parameter to indicate the amount to be subtracted from the account.

To test the class, I instantiated an object called account_1. I performed the following operations:

Checked the initial balance.
Deposited $500 into the account.
Checked the updated balance after the deposit.

Withdrew $200 from the account.
Checked the final balance after the withdrawal.

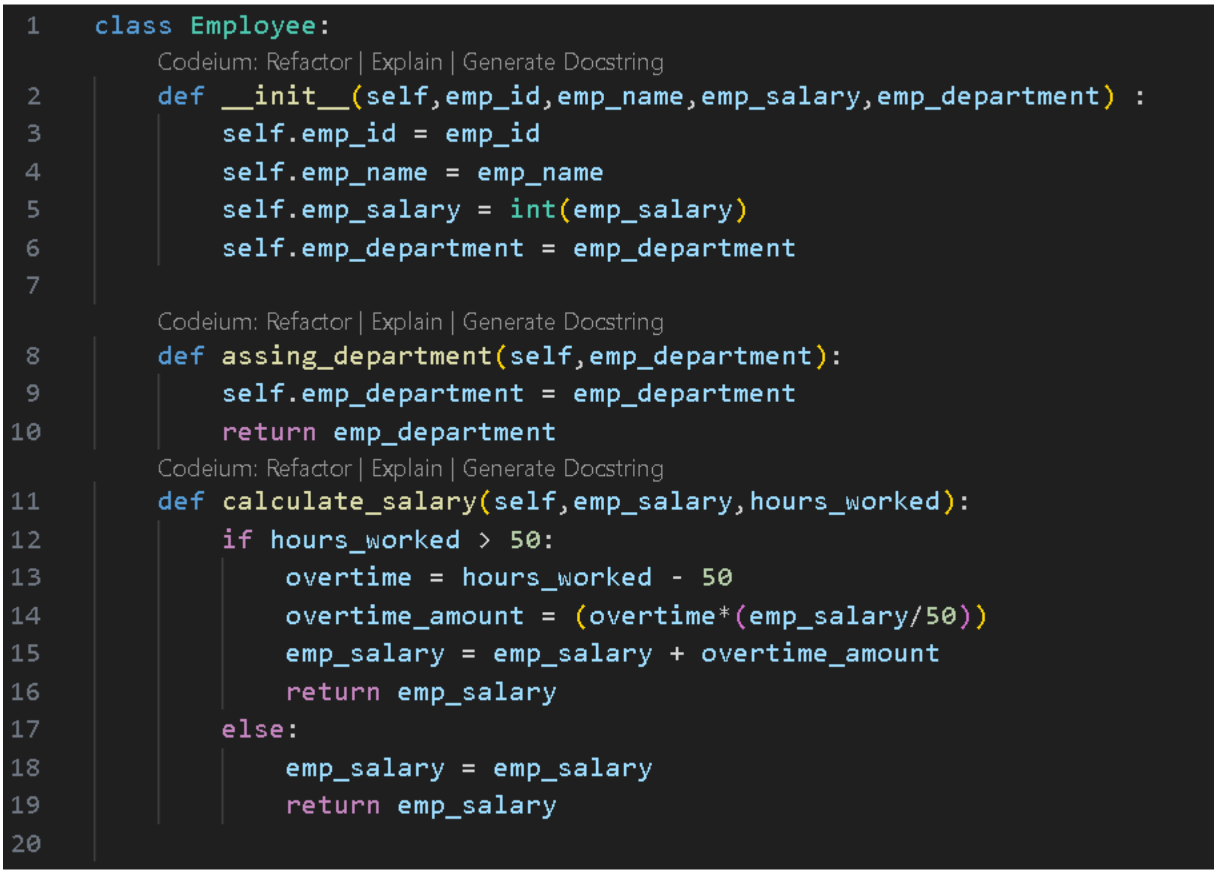
Task 3

Write a Python class Employee with attributes like emp_id, emp_name, emp_salary, and emp_department and methods like calculate_emp_salary, emp_assign_department, and print_employee_details.

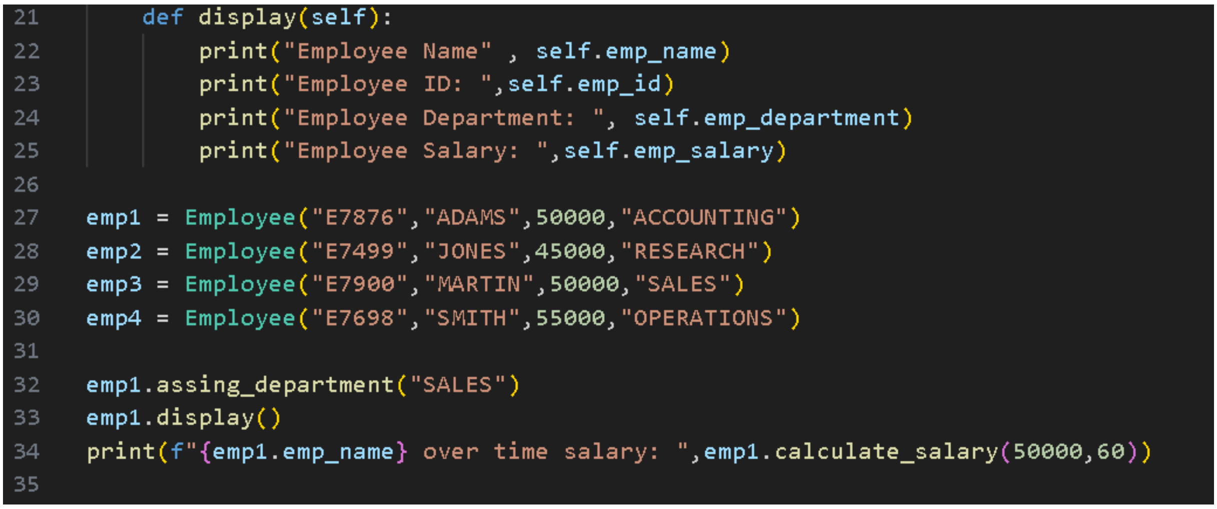
Sample Employee Data:
"ADAMS", "E7876", 50000, "ACCOUNTING"
"JONES", "E7499", 45000, "RESEARCH"
"MARTIN", "E7900", 50000, "SALES"
"SMITH", "E7698", 55000, "OPERATIONS"

- Use 'assign_department' method to change the department of an employee.
- Use 'print employee details' method to print the details of an employee.
- Use 'calculate_emp_salary' method takes two arguments: salary and hours_worked, which is the number of hours worked by the employee. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following formula:

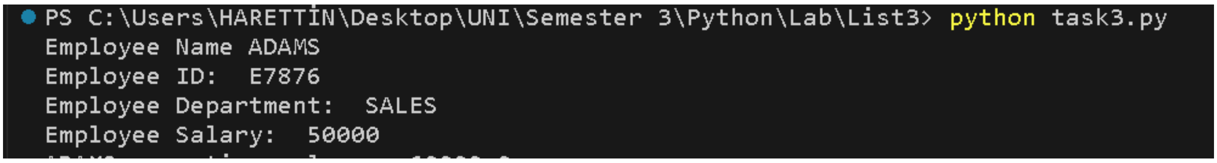
overtime = hours_worked - 50
overtime amount = (overtime * (salary / 50))


Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring

```python
class Employee:
    def __init__(self,emp_id,emp_name,emp_salary,emp_department) :
        self.emp_id = emp_id
        self.emp_name = emp_name
        self.emp_salary = int(emp_salary)
        self.emp_department = emp_department

    def assing_department(self,emp_department):
        self.emp_department = emp_department
        return emp_department
    def calculate_salary(self,emp_salary,hours_worked):
        if hours_worked > 50:
            overtime = hours_worked - 50
            overtime_amount = (overtime*(emp_salary/50))
            emp_salary = emp_salary + overtime_amount
            return emp_salary
        else:
            emp_salary = emp_salary
            return emp_salary

```



```python
    def display(self):
        print("Employee Name" , self.emp_name)
        print("Employee ID: ",self.emp_id)
        print("Employee Department: ", self.emp_department)
        print("Employee Salary: ",self.emp_salary)

emp1 = Employee("E7876","ADAMS",50000,"ACCOUNTING")
emp2 = Employee("E7499","JONES",45000,"RESEARCH")
emp3 = Employee("E7900","MARTIN",50000,"SALES")
emp4 = Employee("E7698","SMITH",55000,"OPERATIONS")

emp1.assing_department("SALES")
emp1.display()
print(f"{emp1.emp_name} over time salary: ",emp1.calculate_salary(50000,60))
```



```
PS C:\Users\HARETTİN\Desktop\UNI\Semester 3\Python\Lab\List3> python task3.py
Employee Name ADAMS
Employee ID:  E7876
Employee Department:  SALES
Employee Salary:  50000
```

Comments:
Firstly, I created a class called Employee in Python. In the class constructor (__init__), I defined parameters such as emp_id, emp_name, emp_salary, and emp_department. These parameters were assigned to their respective attributes within the class.

I implemented the following methods:

assign_department: This method takes a parameter new_department and updates the employee's department.

overtime_salary_calculator: With this method, I calculated overtime salary based on the hours_worked and the employee's current salary. If the employee worked more than 50 hours, the method applies an overtime amount formula and adds it to the salary.

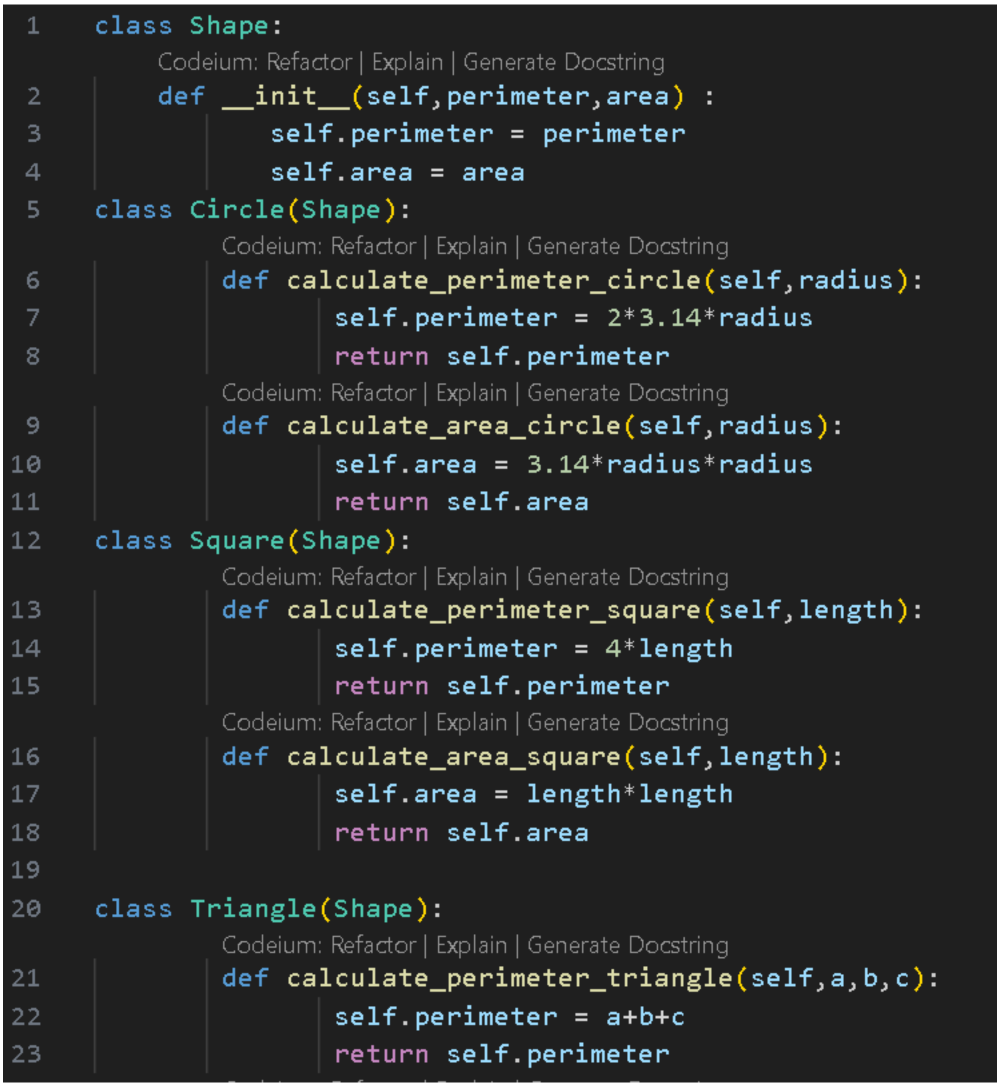
display: The display method prints the employee's data using the print function and basic self notation.

To test the class functionality, I instantiated four Employee objects. I demonstrated the following operations:

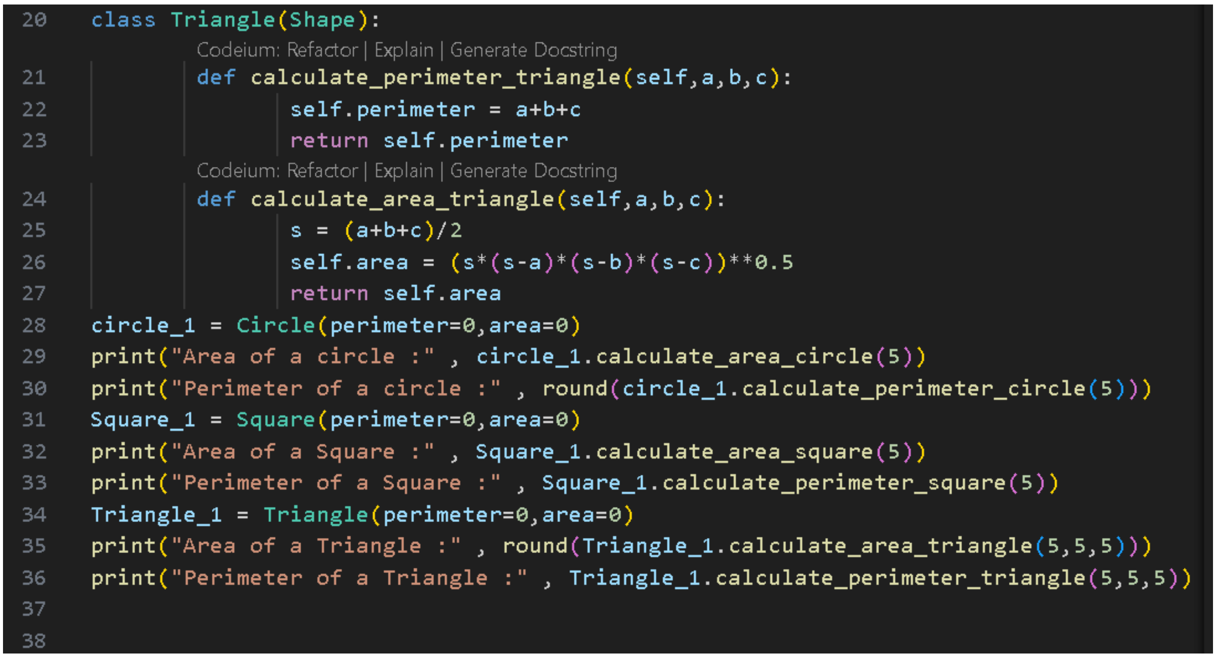
Changed the department of emp_1 using the assign_department method.
Displayed the updated information for emp_1.
Checked the overtime salary for emp_1 by assuming they worked 60 hours.

Task 4

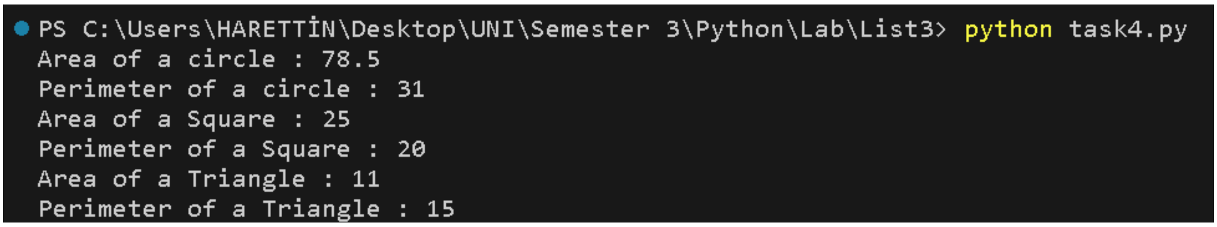
Write a Python program to create a class that represents a shape. Include methods to calculate its area and perimeter. Implement subclasses for different shapes like circle, triangle, and square.


Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring

```python
class Shape:
    def __init__(self,perimeter,area) :
           self.perimeter = perimeter
           self.area = area
class Circle(Shape):
        def calculate_perimeter_circle(self,radius):
               self.perimeter = 2*3.14*radius
               return self.perimeter
        def calculate_area_circle(self,radius):
               self.area = 3.14*radius*radius
               return self.area
class Square(Shape):
        def calculate_perimeter_square(self,length):
               self.perimeter = 4*length
               return self.perimeter
        def calculate_area_square(self,length):
               self.area = length*length
               return self.area

class Triangle(Shape):
        def calculate_perimeter_triangle(self,a,b,c):
               self.perimeter = a+b+c
               return self.perimeter
```


Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring

```python
class Triangle(Shape):
        def calculate_perimeter_triangle(self,a,b,c):
               self.perimeter = a+b+c
               return self.perimeter
        def calculate_area_triangle(self,a,b,c):
               s = (a+b+c)/2
               self.area = (s*(s-a)*(s-b)*(s-c))**0.5
               return self.area
circle_1 = Circle(perimeter=0,area=0)
print("Area of a circle :" , circle_1.calculate_area_circle(5))
print("Perimeter of a circle :" , round(circle_1.calculate_perimeter_circle(5)))
Square_1 = Square(perimeter=0,area=0)
print("Area of a Square :" , Square_1.calculate_area_square(5))
print("Perimeter of a Square :" , Square_1.calculate_perimeter_square(5))
Triangle_1 = Triangle(perimeter=0,area=0)
print("Area of a Triangle :" , round(Triangle_1.calculate_area_triangle(5,5,5)))
print("Perimeter of a Triangle :" , Triangle_1.calculate_perimeter_triangle(5,5,5))


```



```
● PS C:\Users\HARETTİN\Desktop\UNI\Semester 3\Python\Lab\List3> python task4.py
  Area of a circle : 78.5
  Perimeter of a circle : 31
  Area of a Square : 25
  Perimeter of a Square : 20
  Area of a Triangle : 11
  Perimeter of a Triangle : 15
```

Comments:
I began by defining a base class called Shape. Using the __init__ constructor, I took perimeter and area as parameters and assigned them to their respective attributes within the class.

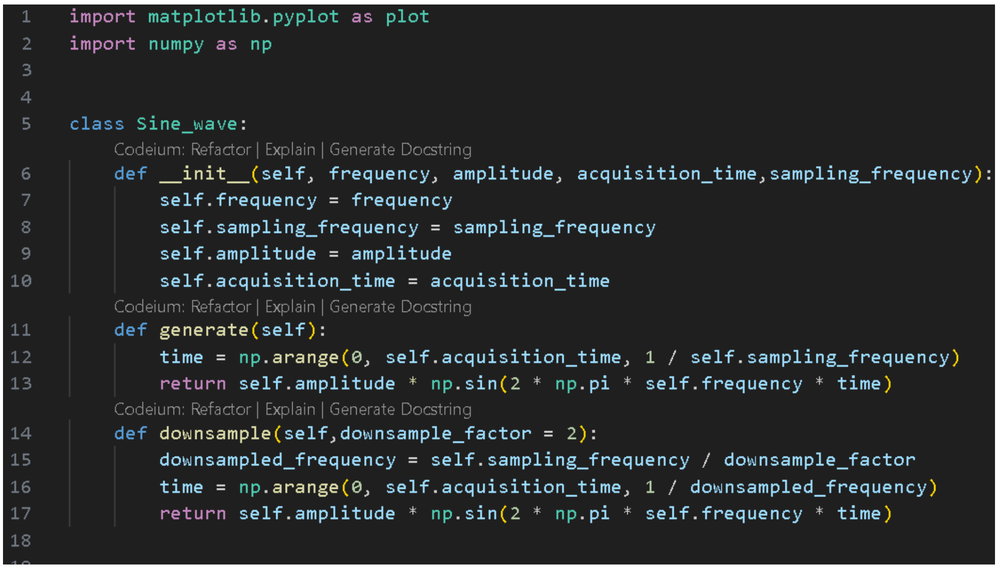
Next, I created three subclasses – Circle, Square, and Triangle – each inheriting from the Shape class. For each subclass, I implemented methods to calculate their specific area and perimeter.

For the Circle class, the area and perimeter calculation methods were tailored to the properties of circles. Similarly, for the Square and Triangle classes, methods were defined to compute the area and perimeter based on their respective shapes.

To demonstrate the functionality, I instantiated one object for each subclass. Subsequently, I calculated and obtained the area and perimeter for each shape.

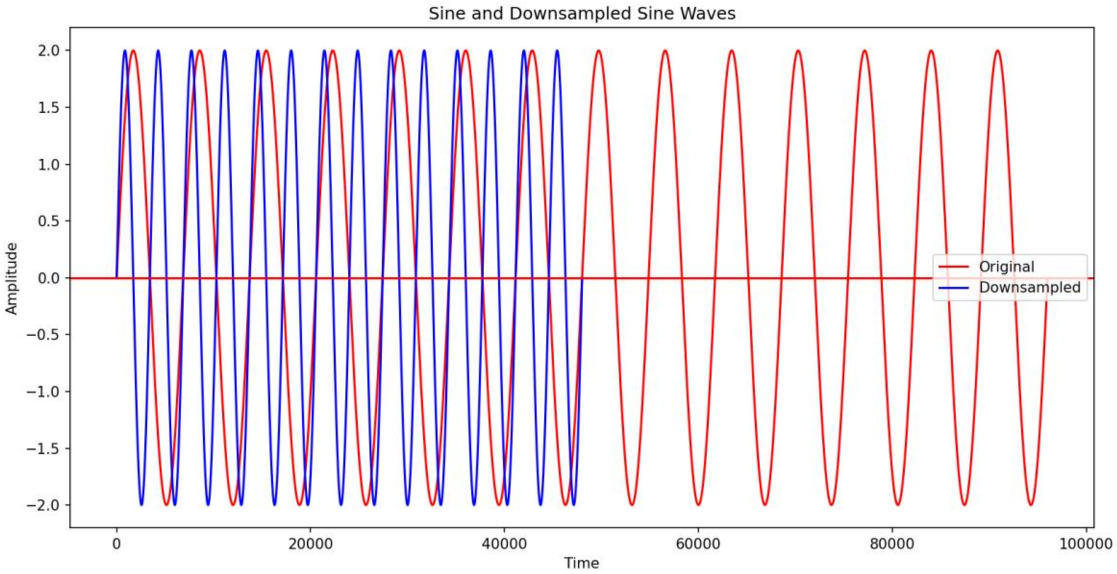
Task 5

Write a program that generates a set of samples of a sine wave. The parameters are as follows: Frequency = last 2 digits of your index number [Hz] (if it is ‘00’ then use ‘01’), Sampling Frequency = 48 [kHz], Acquisition time = 2 [s], Amplitude = 2. Hint – use numpy and matplotlib libraries. It should be realized as a class which stores the sine wave, has a method to generate sine wave samples with chosen parameters, plotting the sine wave as well as returning samples of down sampled wave. Plot both sets on one figure for comparison.



Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring

```python
import matplotlib.pyplot as plot
import numpy as np


class Sine_wave:
    def __init__(self, frequency, amplitude, acquisition_time,sampling_frequency):
        self.frequency = frequency
        self.sampling_frequency = sampling_frequency
        self.amplitude = amplitude
        self.acquisition_time = acquisition_time
    def generate(self):
        time = np.arange(0, self.acquisition_time, 1 / self.sampling_frequency)
        return self.amplitude * np.sin(2 * np.pi * self.frequency * time)
    def downsample(self,downsample_factor = 2):
        downsampled_frequency = self.sampling_frequency / downsample_factor
        time = np.arange(0, self.acquisition_time, 1 / downsampled_frequency)
        return self.amplitude * np.sin(2 * np.pi * self.frequency * time)

```



Comments:

To begin, I imported the necessary libraries and defined a class named SineWave. Using the __init__ constructor, I took parameters such as frequency, amplitude, acquisition_time, and sampling_frequency, assigning them to corresponding attributes of the class.

I implemented a method called generate, which serves as the sine wave generator. Utilizing the time formula obtained from online sources and the np.arange() function, I defined the time for the wave. The method then returns the complete formula for generating a sine wave.

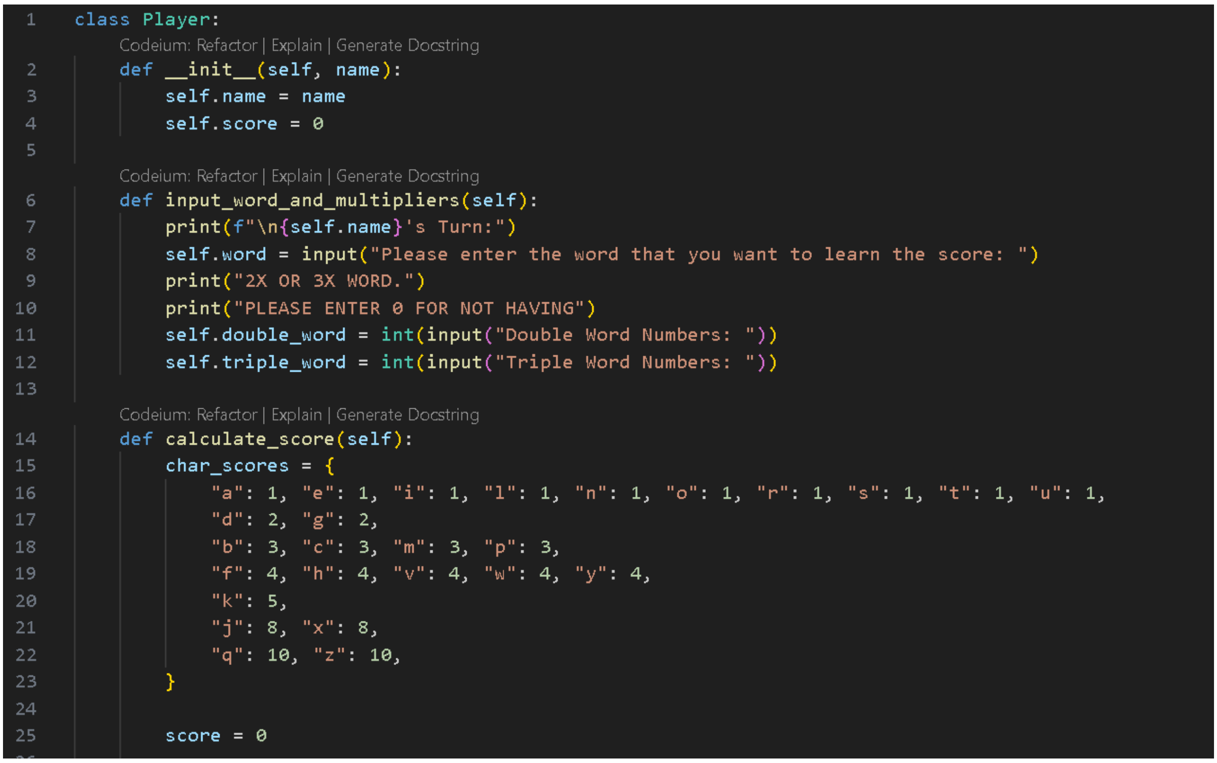
Additionally, I created a method named downsample with a parameter downsample_factor and a constant value of 2. This factor determines how much downsampling is applied manually. The downsampled frequency is calculated as the sampling frequency divided by the downsample factor. Similar to the generator method, the time formula is adjusted, replacing the sampling frequency with the downsampled frequency. The return statements, however, remain the same as in the generator method.

Subsequently, I instantiated an object called my_sine_wave, specifying its parameters. I then applied the downsample function to this object, creating a second object called my_sine_wave_downsampled.

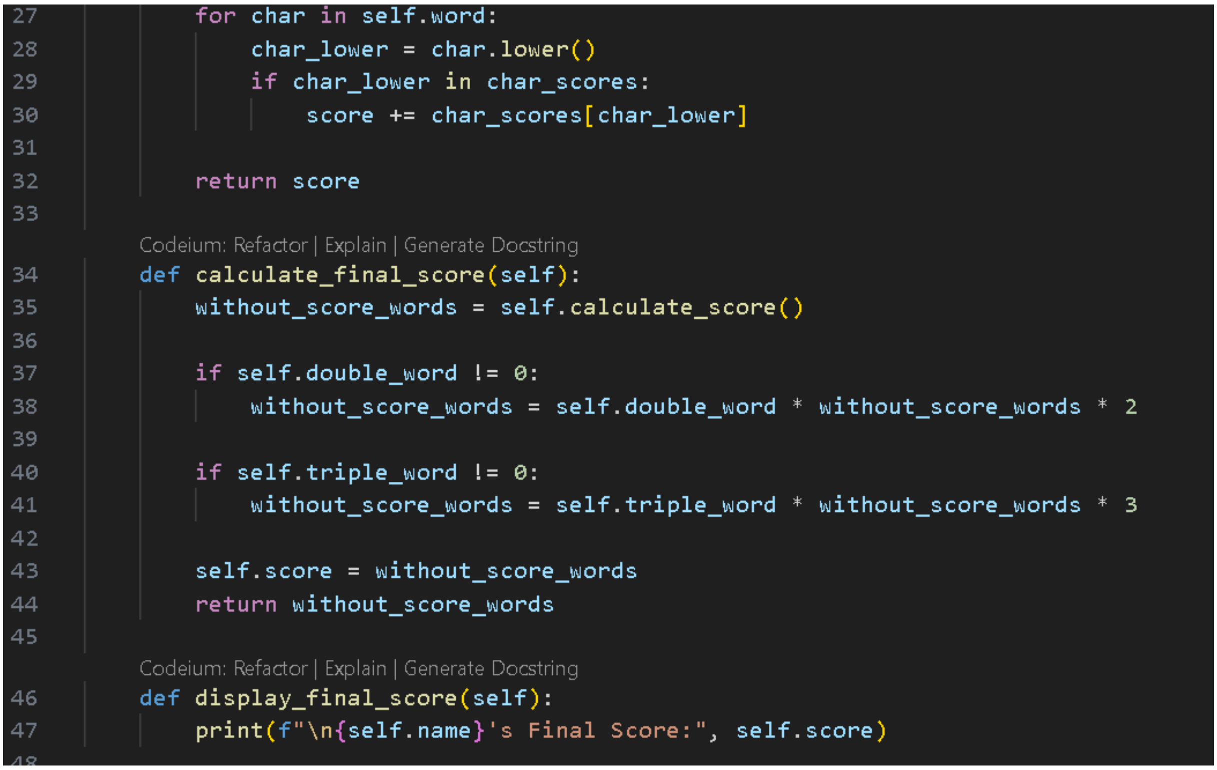
Finally, I designed a plot using the matplotlib.pyplot library and its attributes.

Task 6

Write a program with the same functionality as in List 1 task 6. It should be realized as a ‘Player’ class, where each of the player is a separate object. Inside it stores the score of the player and has method for calculating the score of a given word. Make a simple text UI to enhance the program's accessibility for players.

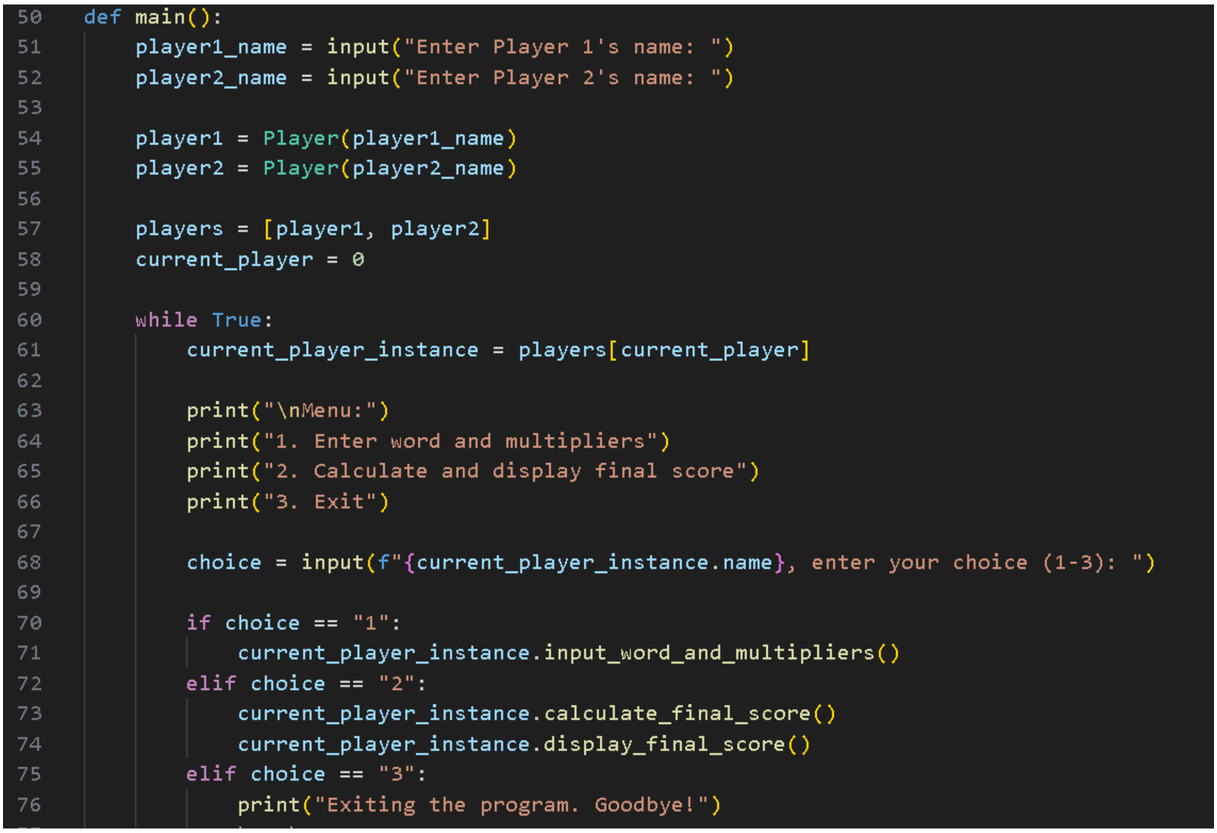

Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring

```
class Player:
    def __init__(self, name):
        self.name = name
        self.score = 0

    def input_word_and_multipliers(self):
        print(f"\n{self.name}'s Turn:")
        self.word = input("Please enter the word that you want to learn the score: ")
        print("2X OR 3X WORD.")
        print("PLEASE ENTER 0 FOR NOT HAVING")
        self.double_word = int(input("Double Word Numbers: "))
        self.triple_word = int(input("Triple Word Numbers: "))

    def calculate_score(self):
        char_scores = {
            "a": 1, "e": 1, "i": 1, "l": 1, "n": 1, "o": 1, "r": 1, "s": 1, "t": 1, "u": 1,
            "d": 2, "g": 2,
            "b": 3, "c": 3, "m": 3, "p": 3,
            "f": 4, "h": 4, "v": 4, "w": 4, "y": 4,
            "k": 5,
            "j": 8, "x": 8,
            "q": 10, "z": 10,
        }

        score = 0
```

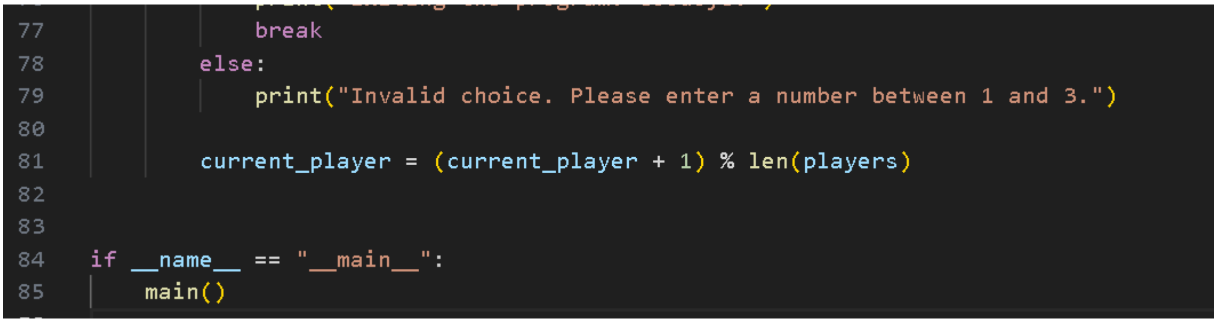


Codeium: Refactor | Explain | Generate Docstring
Codeium: Refactor | Explain | Generate Docstring

```python
        for char in self.word:
            char_lower = char.lower()
            if char_lower in char_scores:
                score += char_scores[char_lower]

        return score

    def calculate_final_score(self):
        without_score_words = self.calculate_score()

        if self.double_word != 0:
            without_score_words = self.double_word * without_score_words * 2

        if self.triple_word != 0:
            without_score_words = self.triple_word * without_score_words * 3

        self.score = without_score_words
        return without_score_words

    def display_final_score(self):
        print(f"\n{self.name}'s Final Score:", self.score)
```



```python
def main():
    player1_name = input("Enter Player 1's name: ")
    player2_name = input("Enter Player 2's name: ")

    player1 = Player(player1_name)
    player2 = Player(player2_name)

    players = [player1, player2]
    current_player = 0

    while True:
        current_player_instance = players[current_player]

        print("\nMenu:")
        print("1. Enter word and multipliers")
        print("2. Calculate and display final score")
        print("3. Exit")

        choice = input(f"{current_player_instance.name}, enter your choice (1-3): ")

        if choice == "1":
            current_player_instance.input_word_and_multipliers()
        elif choice == "2":
            current_player_instance.calculate_final_score()
            current_player_instance.display_final_score()
        elif choice == "3":
            print("Exiting the program. Goodbye!")
```



```
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 3.")

        current_player = (current_player + 1) % len(players)


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

Comments:

Firstly, I defined a class called Player and constructed it by taking a name parameter, which I assigned to the name attribute. I also set a constant score of 0 as an attribute. Later, I created a method called input_words_and_multipliers, similar to my previous Scrabble game. In this version, I assigned the word, triple_word, and double_word as attributes to this class because I plan to use them later.

My second method, calculate_score, computes the score of the word without considering multipliers and returns it as a score. The third method, calculate_final_score, is similar to the function in my previous Scrabble game, with the only difference being that I assign without_score_words to self.score. So, we can say methods are functions of classes, basically.

My fourth method is to display the final score.

In the main function, I created two players, named them, and stored them in a list. I initialized current_player to zero to start from player one. Later, I implemented a while True statement, which finishes when we choose option 3. This is controlled by a menu using choice and if statements."Lastly I checked with if __name__ == "__main__" if the script is being run as the main program.

Conclusion

This list was very helpful to learn main techniques and usage of OOP in Python which is very important I think and related to real life examples.I think first four tasks were quite similar but they gave me a solid knowledge and synthax knowledge of classes in Python.