Coding Tasks - Week 5

Welcome to the Python Programming Exercise Sheet!
In this exercise sheet, we will cover some of the fundamental concepts in Python programming.
Topics covered are: Object oriented programming (OOP).

DEADLINE: 5th June until 12:15
Your name here: Shanza amber **Your university mail**: samber@uni-osnabrueck.de

---

Important information:
In order to pass this sheet you need to achieve 10/20 points.
For the best possible grade you require 20/20 points, however, since some harder tasks may take a lot of time you don't have to pressure yourself.
If you complete any three tasks on a sheet you will definitely get a good grade for that sheet.
Hand in your sheet in studip in the respective folder until the deadline.
If you receive no email until a few days after submission you will have passed, the sample solution will also be uploaded around then.
If you receive an email you don't need to worry, you can fail one sheet and also your total points will also be taken into account for the final pass or fail.

Imports

From now on you will encounter an import cell here in the top which you should **always** execute first to import necessary libraries for later.

Additionally, we want to provide you with a rundown of how imports work since it is related to classes and methods.
Imports always consist of either a full or a partial import, full imports are written like `import library`, partial imports are written like `from library import Class`.

You can think of libraries as separate python scripts from which you import code written, if you say `from myscript import ThisClass`, what you are essentially doing is run the `ThisClass` class definition part of the script `myscript.py`.

Many libraries are built-in and do not require installing such as `math`, `re` (regex), or `time`.
Others, like `numpy`, `pandas`, or `tensorflow`, require an installation via pip or another method.

Additionally, you can use `as` and `,` in your imports.
If you type `import numpy as np` for instance, then you do not have to type `numpy.array`, but rather

just `np.array` which can be a useful abbreviation or be used to rename something important if you have two functions with the same name.

If you do `from datetime import datetime, timedelta` for example, you are selecting specifically the part `datetime` and `timedelta` only from the `datetime` library.

You may even do something like `from library import Function as anothername`.

We hope this explanation helps you with formulating and understanding import statements.

```
from datetime import datetime, timedelta
from math import pi, sqrt
```

Task 1 - Creating a Student Class (6 points)

Create a Python class called `Student` which should take the following attributes: `name`, `age`, and `grade`. After you define the class, implement the methods `get_name()`, `get_age()`, and `get_grade()` to access these attributes.

You will have to write the whole class from scratch here, then in the cell right below, create an instance of the class and name it `student`.
Finally, you can execute the third code cell which is already pre-written code to retrieve the attributes of the class from your written getter-methods.

Hint: You will need to define a `__init__()` method to be able to receive three arguments or a tuple which represents the three arguments.
Additionally, you will need three getter-methods which **return** the value of a certain attribute of the class.

Note: This is more like a toy task honestly, in reality you would not use getters for simple attributes like this and you could also write a `__repr__` or a `__str__` method to make the instance print it's attributes, this is mainly for practice. ;)

```
class Student:
    def __init__(identity, name, age, grade):
        identity.name = name
        identity.age = age
        identity.grade = grade

    def get_name(identity):
        return identity.name

    def get_age(identity):
        return identity.age
```

```
    def get_grade(identity):
        return identity.grade



student = Student("Amber", 25, "S")



print("Name:", student.get_name())
print("Age:", student.get_age())
print("Grade:", student.get_grade())
```

```
Name: Amber
Age: 25
Grade: S
```

Task 2 - Bank Account Class (5 points)

Create a class called `BankAccount` with the following:

- It should have the attributes: `account_number` and `balance`.
- The dunder methods: `__init__` and `__str__`.
- The methods: `deposit(amount)` and `withdraw(amount)`.
- Initialisation method: `__init__(self, account_number, balance)` which sets the attributes `account_number` and `balance`.
- String conversion method; also known as print method: `__str__(self)` which simply returns the string `f'The account {self.account_number} has a balance of {self.balance}.'`.
- Deposit method: `deposit(self, amount)` which takes an amount (int) as input and adds it to the current balance (attribute) of the account.
- Withdrawal method: `withdraw(self, amount)` which takes an amount (int) as input and subtracts it from the current balance of the account, if sufficient funds are available, else it will return the string `"Insufficient funds."`.

Note: Here for the "Insufficient funds" case of the withdrawal method one could also raise a custom error named, for example, InsufficientFundsError. This is not needed and not wanted here, but this is something for you to think about in case you in the future encounter this situation where it makes sense to implement an error for certain cases with classes.

```
class BankAccount:
    def __init__(identity, account_number, balance):
        identity.account_number= account_number
        identity.balance= balance
```

```
    def __str__(self):
      return f'The account {self.account_number} has a balance of {self.balance}.'


    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds."
        else:
            self.balance -= amount


# Create a BankAccount instance with an account number and initial balance
account = BankAccount("123456789", 50)

# Display the initial balance (str method)
print(account)

# Perform a deposit
account.deposit(1000)

# Display the updated balance
print("Balance after deposit:", account.balance)

# Perform a withdrawal
account.withdraw(5000)

# Display the final balance
print("Balance after withdrawal:", account.balance)

# Try to withdraw too much
print(account.withdraw(10000))
```

```
The account 123456789 has a balance of 50.
Balance after deposit: 1050
Balance after withdrawal: 1050
Insufficient funds.
```

Task 3 - Geometric Classes (5 points)

Finish the following three geometric classes by implementing the four missing statements in the methods.

- Rectangle: Implement the missing statement in the `perimeter` and `area` method.
- Triangle: Implement the missing statement in the `__init__` method and the equilateral test in the `is_equilateral` method.

- Circle: Implement the missing statement in the `diameter` method.

Hint: For the triangle you may want to look up the `max` function and for the rest you can get away with very simple mathematics, you can also look at the internet to get the formulas if you don't know them by heart.

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def perimeter(self):
        return 2*(self.width + self.height)

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


class Triangle:
    def __init__(self, a, b, c):
        #Triangles, given three lenghts, are only possible if any two side lenghts are greate
        if not ((a+b > c) and (b+c > a) and (a+c > b)):
            raise ValueError(f"The three lengths {a}, {b}, and {c} cannot form a triangle tog

        self.a = a
        self.b = b
        self.c = c

        #The width of a traingle is the length of the longest side (use the max function over
        self.width = max([self.a,self.b,self.c])

        #The height of a triangle is calculated, for all known sides, via the formula given h
        self.height = 0.25*self.width*sqrt(sum([self.a,self.b,self.c]))

    def perimeter(self):
        return self.a + self.b + self.c

    def area(self):
        return 0.5*self.width*self.height

    def is_equilateral(self):
        #A triangle is equilateral if two sides have the same length
        if (self.a == self.b) and (self.b == self.c):
            return True #the triangle is equilateral

        else:
            return False #the triangle is not equilateral


class Circle:
    def __init__(self, radius):
```

```
        self.radius = radius

    def perimeter(self):
        return self.radius*pi

    def area(self):
        return self.radius*pi**2

    def diameter(self):
        return  2 * self.radius


# Create an instance of the Rectangle class
rectangle = Rectangle(5, 3)

# Calculate and print the perimeter of the rectangle
print(f'Perimeter: {rectangle.perimeter()}')
print(f'Area: {rectangle.area()}')
```

```
Perimeter: 16
Area: 15
```

```
from math import sqrt
triangle_a = Triangle(3, 4, 5)

print(f'Width: {triangle_a.width}')
print(f'Height: {triangle_a.height}')
print(f'Perimeter: {triangle_a.perimeter()}')
print(f'Area: {triangle_a.area()}')
print(f'The triangle is equilateral: {triangle_a.is_equilateral()}\n')

triangle_b = Triangle(5, 5, 6)

print(f'Width: {triangle_b.width}')
print(f'Height: {triangle_b.height}')
print(f'Perimeter: {triangle_b.perimeter()}')
print(f'Area: {triangle_b.area()}')
print(f'The triangle is equilateral: {triangle_b.is_equilateral()}')
```

```
Width: 5
Height: 4.330127018922193
Perimeter: 12
Area: 10.825317547305481
The triangle is equilateral: False

Width: 6
Height: 6.0
Perimeter: 16
Area: 18.0
The triangle is equilateral: False
```

```
#Verifying the error for an impossible triangle works
triangle = Triangle(1, 1, 1.5) #reason this does not work is due to 1+1>2 being wrong


from math import pi
circle = Circle(5)

# Calculate and print the perimeter of the rectangle
print(f'Perimeter: {circle.perimeter()}')
print(f'Area: {circle.area()}')
print(f'Diameter: {circle.diameter()}')
```

```
Perimeter: 15.707963267948966
Area: 49.34802200544679
Diameter: 10
```

Task 4 - Time based while loop (4 points)

Implement a while loop which runs exactly one minute.

You must define a condition and the start time. For this task, orient yourself on the given help in the cells right under this to understand how the `datetime` library may be used and how to calculate distances with it.

Then, write a starttime and a conditional for the while loop below.

```
from datetime import datetime

current_time = datetime.now()
print(current_time)
```

```
2023-06-05 04:37:43.776698
```

```
starttime = datetime(year=2023, month=5, day=22, hour=12, minute=15, second=0)
print(starttime)
```

```
2023-05-22 12:15:00
```

```
endtime = datetime(year=2023, month=5, day=22, hour=13, minute=45, second=0)
print(endtime)
```

```
2023-05-22 13:45:00
```

```
distance = endtime - starttime
print(distance)
print(f'The distance between start and end in seconds is equal to {distance.seconds}sec secon
print(f'In minutes this would be {distance.seconds/60}mins, in hours it would be {round(dista
```

```
     1:30:00
     The distance between start and end in seconds is equal to 5400sec seconds.
     In minutes this would be 90.0mins, in hours it would be 1.5h.


counter = 0
start = datetime.now()

while datetime.now() < endtime:
    counter += 1

print(f'Loop started at {start}.')
print(f'Loop ended at approximately {datetime.now()}\n')
print(counter)

     Loop started at 2023-06-05 04:37:59.073195.
     Loop ended at approximately 2023-06-05 04:37:59.074361

     0
```