HOMEWORK WEEK 1

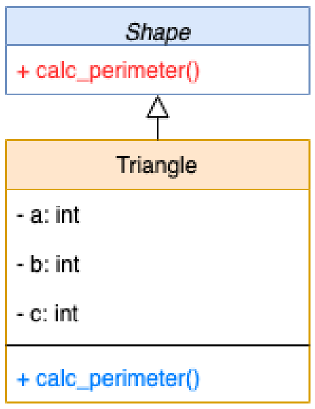
(handout for students)

25 marks in total

The purpose of this homework is for you to gain a greater context into how the concept of OOP is modelled and implemented.

TASK 1 - Understanding UML Diagrams

This task is unmarked but tells you what you need to know for later tasks.



In this diagram:

red text is showing this method is abstract

blue text is showing this method is being implemented/overriden

The above is an example of a UML (Unified Modelling Language) diagram, also known as a class diagram. It is the most well known and regularly used visual representation of programs that you'll find.

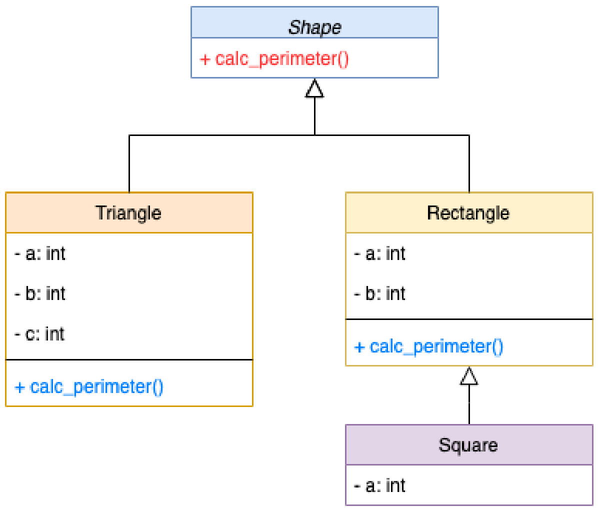
These are very useful when you're trying to plan your classes and especially handy in visualising what methods and attributes are being inherited or overridden in a system. For example the - and + symbols distinguish between methods and attributes, and the white arrow indicates inheritance.

This diagram visualises one of the OOP Tasks gone over in OOP Lesson 1. The diagram tells us that:

1. There's a parent class Shape
   a. This class has no attributes
   b. There is an abstract method called calc_perimeter
2. There's a child class called Triangle
   a. Class has three attributes a, b and c
   b. The method calc_perimeter is implemented

TASK 2 - Implementing UML Diagrams - 15 MARKS

The following is an extension of the UML diagram in the last section.
Make a new python file and build on the code from the session so it is reflective of this diagram.



You can find the session starter code here:

```
import abc


class Shape(object):
    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def calc_perimeter(self, input):
        """Method documentation"""
        return


class Triangle(Shape):

    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def calc_perimeter(self):
        perim = self.a + self.b + self.c
        print("Consider me implemented", perim)
        return perim
```

After implementation, create an instance of each new class and for both, print out the result of the calc_perimeter method.

TASK 3 - Creating UML Diagrams - 10 MARKS

The following is a code block from OOP Lesson 2:

```python
from abc import abstractmethod

class PlaySongsLyrics:

   @abstractmethod
   def sing_lyrics(self):
       pass

class PlaySongsMusic:

   @abstractmethod
   def play_guitar(self):
       pass

   @abstractmethod
   def play_drums(self):
       pass

class PlayInstrumentalSong(PlaySongsMusic):

   def play_drums(self):
       print("Bum-bum-bum")

   def play_guitar(self):
       print("Some guitar solo*")

class PlayRockSong(PlaySongsMusic, PlaySongsLyrics):

   def play_guitar(self):
       print("Heavy metal guitar solo*")

   def sing_lyrics(self):
       print("We will, we will rock you")

   def play_drums(self):
       print("Bum-bum-bum")
```

Make a UML Diagram that visualises the class relationships, attributes and methods for the above code.

You can do this on draw.io (a free online option which has a template for class diagrams), Microsoft Visio, and there are also more options out there if you wish.

Once you have finished your diagram you will need to export it as an image file for submission.

```
SUBMISSION CRITERIA:
   ● This homework is to be submitted solely via GitHub.
   ● Your pull request needs to contain the Python solution for Task 2
     and an image file for Task 3.
```