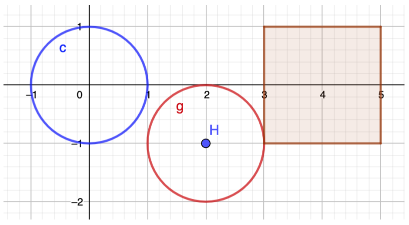
Lab 3 – Geometry OOP

The purpose of this lab is to use object-oriented programming in Python to reuse code and design well-structured programs.



In this lab you should first plan how to structure your classes using UML and then implement your plan in Python.

Task 1

Create the following geometric classes in two different py scripts:

- Rectangle
- Circle

Note that you may also need additional class(es) if you want to use inheritance and/or composition.

The classes must have:

- a read-only area property
- a read-only perimeter property
- an operator overload of `==` to check equality
- operator overloads of the comparison operators `<, >, <=, >=` for comparisons
- an override of `__repr__()`
- an override of `__str__()`
- x and y representing the center position of the object
- a translation method that makes it possible to move x and y
- error handling
- a method that checks whether a Circle instance is a unit circle
- a method that checks whether a Rectangle instance is a square

Give the classes more suitable functionality as appropriate. For example, it would be nice to draw the geometric figures and visualize translations. Your classes should live in a .py files. For example, one should be able to use your classes like this:

```python
from circle import Circle
from rectangle import Rectangle

circle1 = Circle(x=0, y=0, radius=1)  # unit circle
circle2 = Circle(x=1, y=1, radius=1)
rectangle = Rectangle(x=0, y=0, width=1, height=1)
```

```
print(circle1 == circle2)  # True

print(circle2 == rectangle)  # False

circle1.translate(5, 3) # moves its center 5 points in x and 3 points in y

circle1.translate("THREE", 5)  # raise TypeError with an appropriate
message

circle3 = Circle(radius=3) # a circle with center 0,0 with radius 3

circle3 >= circle1 # True

rectangle2 = Rectangle(width=3, height="5") # raise TypeError
```

Start by creating a UML with lucidchart and then implement your UML in Python.

Do manual tests like the ones above, but test more. A good tip is to have a separate .ipynb file that imports your classes so you can run manual tests while writing the classes.

Task 2 (bonus)

- create a Cube class with corresponding functionality
- create a Sphere class with corresponding functionality
- consider how various methods and properties change (e.g., area, perimeter/circumference)
- write unit tests for your classes
  - have separate .py files for the unit tests
  - for example `test_circle.py, test_rectangle.py`
- a Shape2dPlotter class that can plot several 2D shapes in the same axes
  - use `matplotlib.patches.Rectangle` and `matplotlib.patches.Circle` to achieve this

Task 3 - video pitch

Create a video 5-10 min long where you record the screen and you go through your UML and code. Your target audience should be a classmate. Explain with correct computer science terminology but keep it simple.

For simplicity use microsoft teams and open up a meeting with yourself, then share screen and record. Afterwards download the video file and upload to learnpoint. If the file size is too big (100 mb is the limit for learnpoint) then you could upload it to youtube but keep it unlisted. Or use google drive, onedrive, dropbox or similar, but make sure that anyone with the link can view the file.

Handin

Handin the following to learnpoint:

- a link to your public github repository for this lab
- a video file or link to the video

Assessment

If you have received any code snippet from someone else, LLM or found it on a website, it is important that you provide a source reference. Write a comment next to the code you have taken.

Godkänt

- produced a simple UML
- solved the tasks correctly except for bonus
- code is commented with relevant comments (not too much comments)
- use type hinting and docstrings for documentation
- variable names are descriptive and well chosen
- made several relevant git commits

Väl godkänt

- code is easy to follow
- code is well structured with appropriate classes, functions, and methods
- code is reused where possible instead of repeated, follow DRY principle
- you can explain the code in your video in a way that shows depth in your understanding
- you should use correct computer science terminology in your videos
- you have completed all tasks