DVA245 Lab Assignment 1
Object Oriented Programming
Iterators, Generators

Anna Friebe

January 2021

This lab consists of two parts, one where you implement a Rectangle Shape python class, and one where you get familiar with iterators/ generators in Python, and the yield keyword.

On Canvas you can download a zip-file containing the files for the lab assignment. There are two implementation files and two test files:

- `shapes_begin.py` contains the Shape classes
- `shapes_container_begin.py` contains a ShapeContainer class for generator implementation and a small program.
- `shapes_test.py` contains tests for the Shape classes.
- `shapes_container_test.py` contains tests for the ShapeContainer class.

See the labInfo2022 document on Canvas for some more information on the unittest framework and the lab course in general.

1 Implement a Rectangle Shape class

In this part of the assignment you will learn about the class concept in Python by implementing a Rectangle Shape class

The file `shapes_begin.py` that contains code for an abstract `Shape` base class, concrete subclasses `Circle` and `Triangle`, and a skeleton for the concrete subclass `Rectangle`. The file `shapes_test.py` contains tests for the methods of the Shapes classes. This structure is illustrated in fig. 1.

1.1 The `Shape` base class

The `Shape` base class has a constructor that besides the special variable `self` takes two arguments, the name and color of the shape, and stores those as data members. It contains implementation of the functions `name` and `color`, that return the name and color data members. It also contains the functions `area`

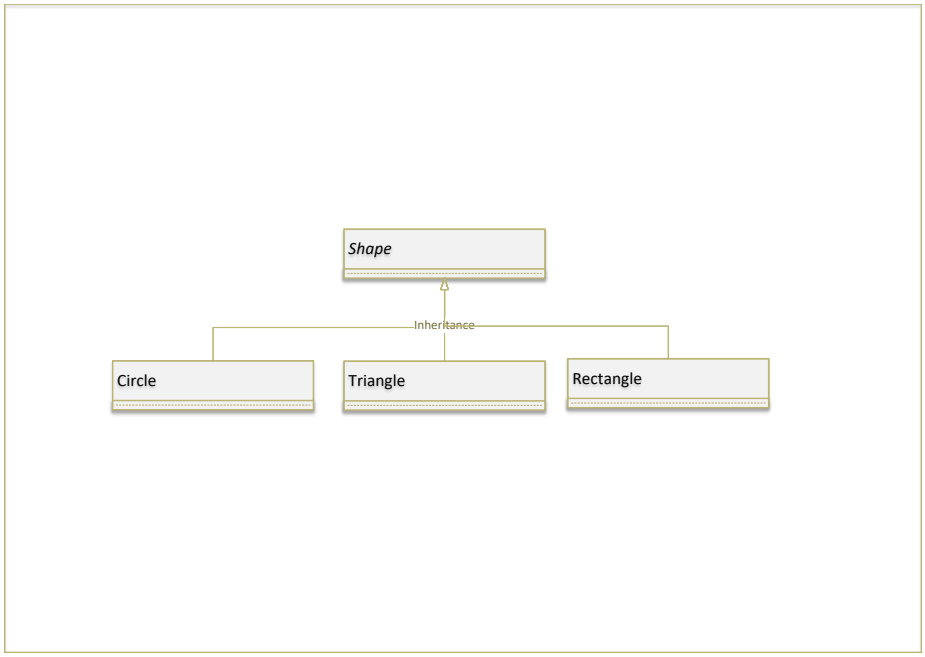

Figure 1: The `Circle`, `Triangle` and `Rectangle` classes all have `Shape` as a common base class.

and `circumference`. We require that subclasses implement these functions, but the implementation in the base class just raises an exception, because the area and circumference cannot be calculated from a name and color. The exception raised is Python's built-in `NotImplementedError`, that is a derived from `RunTimeError`. In fact the Python exceptions are also classes, that all derive from `BaseException`. If you want to know more about Python's exceptions, and how to create your own exception types by subclassing them, you can find information here: `https://docs.python.org/3/library/exceptions.html`.

Lastly, the Shape class contains the `printInfo` function that gives information about the Shape object.

1.2 Shape subclasses

By specifying the Shape in the class definition, it is set as the base class of the concrete Shape classes, which means that the derived subclasses have the data members and member functions of the base class, and can add data members, and add or override member functions. For example, see the `Circle` class:

```
class Circle(Shape):
```

The `Circle` constructor takes the name, color and radius as arguments.

```
def __init__(self, name, color, radius):
    super().__init__(name, color)
    self._radius = radius
    return
```

The first line in the constructor calls the constructor of the base class/ super class `Shape` with the name and color arguments. This means that the Shape constructor is used to initialize the `_name` and `_color` data members of a `Circle` object. The radius is stored as the data member `self.radius`. Now that we know that the shape is a circle, and the radius of it, we can implement the `area` and `circumference` methods. We override the `Shape` implementations (that raised `NotImplementedErrors`) with implementations that perform the calculations using the radius.

The `Triangle` constructor is more complicated. The `Triangle` is specified by the lengths of two sides, and the angle between them in degrees. In the constructor the height of the triangle and the length of the third side are calculated and stored, you don't need to go into the details on how this is done. These are stored as data members so that the area and circumference can easily be calculated.

There is a skeleton for the `Rectangle` subclass, and you must complete the implementation of the constructor, `area` and `circumference` methods. All of these methods are very short, 1-5 lines of code.

1.3 Shapes unit test program

The file `shapes_test.py` contains tests for the classes implemented in the `shapes_begin.py` file. If you run this before you completed the `Rectangle` implementation, you

will get three errors. When the implementation is correctly done, all tests should pass. You can see in the tests that the functions implemented in the base class can be accessed in the subclass object instances.

The unittest framework supports test automation and allows for simple setup of test cases, and running of the test program. You can run the test program as: `python -m unittest shapes_test`, or use the verbose flag `-v` to get more detailed output, `python -m unittest -v shapes_test`.

2 Iterators/ Generators

2.1 ShapesContainer

The `ShapesContainer` in the `shapes_container_begin.py` is a class that keeps a number of shapes in a python list, that is a data member, `_shapesList`. The `ShapesContainer` implements the `append` function that just appends the shape to `_shapesList`. It also implements the `__iter__` function, that provides `Iterable` functionality to `ShapesContainer`. This means that we can write

```
for s in myShapesContainer:
    # do something with each shape
```

Under the hood in the for loop, an iterator is retrieved from `myShapesContainer`, and then `next(it)` is called until a `StopIteration` exception is raised.

The implementation of the `__iter__` function uses the `yield` keyword.

```
def __iter__(self):
    for s in self._shapesList:
        yield s
```

By using the `yield` keyword, a generator iterator will be created, with a `next(it)` function that returns shape by shape until there are no more `yield` statements, and then raises a `StopIteration` exception. We can define an Iterator class with `__iter__` and `next()` functions, but by using `yield` Python will do this for us.

Your task is to implement the `areas` function, that provides a generator for iteration over the areas of all shapes in `_shapesList`. Use the `yield` keyword. This is also a short function, 1-5 lines of code.

2.2 Small program with Shapes and ShapeContainer

The code

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

in `shapes_container_begin.py` specifies that `main()` is executed when the file is run, but not when the module is imported into another file.

When you implemented the `Rectangle` class and the `areas` function, the program should run and print out information about the shapes.

2.3 `ShapesContainer` test

In the shapes_container_test.py there are tests of the `ShapesContainer` class. When you implemented the `Rectangle` class and the `areas` function, all tests shall pass.

In this test you can see direct calls to `iter()` and `next()` that are used under the hood in a for loop.

3 How to display

When you have completed the implementation, you should run the program in `shapes_container_begin.py`, and the tests in `shapes_test.py` and `shapes_container_test.py`