Object Oriented Programming in Python
Lesson 1
Class definition (Encapsulation and information hiding)

Outline:

- How to create classes and instantiate objects in Python
- How to add attributes and behaviors to Python objects
- How to organize classes into packages and modules

Class definition:

```
class MyFirstClass:
    pass
```

Instantiate object from class:

```
a = MyFirstClass()
print(a)
```

Create and instantiate an empty class, and assign attributes to the object:

```
class Point:
    pass

p = Point()
p.x = 5
p.y = 4
print(p.x, p.y)
```

Add actions to class:

```
class Point:
    def reset(self):
        self.x = 0
        self.y = 0

p = Point()
p.reset()
print(p.x, p.y)
```

Method vs function

Methods have one required parameter named 'self'. The self argument to a method is a reference to the object that the method is being invoked on. We can access attributes and methods of that object as if it were any another object.

Notice that when we call the p.reset() method, we do not have to pass the self argument into it. Python automatically passes the reference of the 'p' object to the method.

However, a method is just a function that happens to be on a class. Instead of calling the method on the object, we can invoke the function on the class, explicitly passing our object as the self argument:

```
p = Point()
Point.reset(p)
```

Add more actions to the class:

```
import math
class Point:
	def move(self, x, y):
		self.x = x
		self.y = y

	def reset(self):
		self.move(0, 0)

	def calculate_distance(self, other_point):
		return math.sqrt( (self.x - other_point.x)**2 + (self.y - other_point.y)**2)


p1 = Point()
p2 = Point()

p1.reset()
p2.move(0,5)

# method testing
assert (p2.calculate_distance(p1) == p1.calculate_distance(p2))
```

Initializing the object

If we don't explicitly set the x and y positions on our Point object, either using
move or by accessing them directly, we will run into AttributeError if we try to access a missing attribute.

Most object-oriented programming languages have the concept of a constructor,
a special method that creates and initializes the object when it is created. Python is
a little different; it has a constructor and an initializer. The constructor function is
rarely used. The Python initialization method is the same as any other method, except it has a special name, **__init__**.

```
class Point:
	def __init__(self, x, y):
		self.move(x, y)
	def move(self, x, y):
		self.x = x
		self.y = y
	def reset(self):
		self.move(0, 0)

# Constructing a Point
point = Point(3, 5)
print(point.x, point.y)
```

We can define default parameters (the arguments are not required):

```
def __init__(self, x=0, y=0):
	self.move(x, y)
```

The constructor function is called **__new__** and accepts exactly one argument; the class that is being constructed (it is called before the object is constructed, so there is no self argument). It has to return the newly created object.

Documenting the code : Docstrings

Each class, function, or method header can have a standard Python string as the first line following the definition. Docstrings are simply Python strings enclosed with apostrophe (') or quote (") characters. Often, docstrings are quite long and span multiple lines, which can be formatted as multi-line strings, enclosed in matching triple apostrophe (''') or triple quote (""") characters.

A docstring should clearly and concisely summarize the purpose of the class or method it is describing. It should explain any parameters whose usage is not immediately obvious, and is also a good place to include short examples of how to use the API.

Modules and packages

For small programs, we can just put all our classes into one file and add a little script at the end of the file to start them interacting. However, as our projects grow, it can become difficult to find the one class that needs to be edited among the many classes we've defined.

A Python file is a module. If we have two files (modules) in the same folder, we can load a class from one module for use in the other module. The import statement is used for importing modules or specific classes or functions from modules.

import module
from module import class
from module import class as classalias
from module import <comma separated list of classes>

A package is a collection of modules in a folder. The name of the package is the name of the folder. This folder is a package if there is a file in the folder named **__init__.py**. If we forget this file, we won't be able to import modules from that folder.
If our modules are organized into packages, our working folder should also contain a main.py file to start the program.

```
parent_directory/
        main.py
        ecommerce/
                __init__.py
                database.py
                products.py
        payments/
                __init__.py
                square.py
                stripe.py
```

Absolute import
import ecommerce.products
from ecommerce.products import Product
from ecommerce import products

Relative import
from <.module inside the current package> import class
from <..module inside the parent package> import class

Information hiding:

Technically, all methods and attributes on a class are publicly available. If
we want to suggest that a method should not be used publicly, we can put a note in
docstrings indicating that the method is meant for internal use only. By convention, we should also prefix this attribute or method with an underscore character.