COMPSCI 105 S1 2017
Principles of Computer Science

Classes 1

Exercise

- What is the output of the following code fragment?

```
x = ['a', 'b', 'c']
y = x
z = ['a', 'b', 'c']
print (x is y)
print (x == y)
print (x is z)
print (x == z)
```


Object Oriented Programming

- An object represents an entity in the real world that can be distinctly identified, e.g., students, dogs, cars, cats, books.




- Object Oriented Programming (OOP) involves the use of objects to create programs.

Objects

- Cars may have:
  - information: colour, current speed, current gear, etc.
  - function: accelerate, brake, change gear, reverse, etc.

Car A

color: red
speed: 50
gear: 4th

Car B

color: white
speed: 5
gear: 1st

Car B – accelerate →

Car B

color: white
speed: 10
gear: 1st

State and Behaviour

- Every real world object has:
  - State – information that the object stores.
  - Behavior – functionality of the object, i.e., what the object can do.
- Example:
  - Consider a system managing university students.
  - A student object has:
    - State – id, name, age, contact number, address, stage, completed courses, current courses, faculty, …
    - Behavior – enroll in a new course, change contact number, change enrolment, choose degree, …

Object is state + behaviour

- A software object's state is represented by its variables, called data fields.
- A software object implements its behavior with methods.
- Every object is a bundle of variables and related methods.
- We make an object perform actions by invoking the methods on that object.
- Example:

```
my_list = [ 1, 2, 3]
my_list.reverse()
```


In a Program

- Our program consists of many different objects
- Two objects of the same kind would have the same set of behaviors, but independent state information
  - Two string objects store different words, but can perform same methods, e.g., lower(), split(), index(), etc.
- For an object in our program
  - State – is defined by variables (data fields).
  - Behaviors – is defined by methods (actions).
- The definition of a particular kind of objects is called a class. Once created, an object is an instance of a class.

Python Class

- A class is the structure we use to define a category of objects. It defines the state and behaviour of a category of objects.
- A class is a template or blueprint defining the date fields and actions (methods) that any instance (object) of that class can have.
- Analogies for class and object:
  - Cookie cutter and cookies.
  - Factory mold and products produced from that mold.

Classes

- Python has a number of classes built-in
  - lists, dictionaries, sets, int, float, boolean, strings
- We can define our own classes
  - creates a new type of object in Python

```
class name_of_the_class:
    # definition of the class goes here
    # initializer
    # methods
```

- Classes consist of:
  - state variables (sometimes called instance variables)
  - methods (functions that are linked to a particular instance of the class)

Example

Example01.py

- An example:

```
class foo:
    a, b, c = 0, "bar", (1,2)
```

- Instantiating Classes
  - A class is instantiated by calling the class object:

```
i = foo()
print (i.a)
print (i.b)
print (i.c)
```

```
0
bar
(1, 2)
```

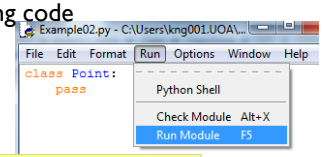

The simplest class possible

Example02.py

- Note: "Pass" is a statement that does nothing
  - It is often used as a placeholder when developing code

```
class Point:
    pass
```

You must run the Example02.py module in Python IDLE before executing the following code fragment




```
>>> p = Point()
>>> p
<__main__.Point object at 0x02702570>
>>> p.x
AttributeError: 'Point' object has no attribute 'x'
>>> p.x = 2
>>> p.y = 4
>>> p.x
2
>>> p.y
4
```

Simplest class, but no attribute has been defined.

Set x and y coordinates

Saving the class

Geometry.py

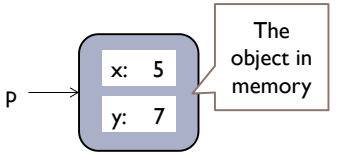
- Classes are designed to help build modular code
  - Can be defined within a module that also contains application code
  - Multiple classes can be defined in the same file
- In this course, we will typically store each class in their own module
  - To use the class in another module, you will need to import the module

```
class Point:
    ...
```

Saved in a file called Geometry.py

```
from Geometry import Point

p = Point(5,7)
```



Setting the initial state of the object

- We want to define the Point class so we can write code that sets the initial values of some variables

```
from Geometry import Point

p = Point(5, 7)
```

- First, we need to define a special method of the Point class called a *constructor*
  - The constructor is called whenever you create an object of the Point class.

Constructors

- Each class should contain a constructor method
  - Name of the method is __init__
  - The method always has at least one parameter, called self
  - Self is a reference to the object that we are creating
  - The constructor can have other parameters

```
class Point:
    def __init__(self, loc_x, loc_y):
            self.x = loc_x
            self.y = loc_y
```

It creates an object in the memory for the class.

```
from Geometry import Point

p = Point(5, 7)
```

Saved in a file called Geometry.py

Accessing Objects

Example02b.py

- After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the object member access operator.
- For example, the following code accesses the x, y coordinates

```
from Geometry import Point

p = Point(5, 7)
print(p.x)
print(p.y)
```


Example: the datetime class

Example03.py

- Example:

```
from datetime import datetime
d = datetime.now()
print("Current year is " + str(d.year))
print("Current month is " + str(d.month))
print("Current day of month is " + str(d.day))
print("Current hour is " + str(d.hour))
print("Current minute is " + str(d.minute))
print("Current second is " + str(d.second))
```

```
Current year is 2015
Current month is 12
Current day of month is 16
Current hour is 15
Current minute is 14
Current second is 50
```

Adding functionality

- Defining more methods
  - A method to shift a point by a given amount in horizontal and vertical directions

```
class Point:
  def __init__(self, loc_x, loc_y):
    self.x = loc_x
    self.y = loc_y

  def translate(self, dx, dy):
    self.x += dx
    self.y += dy
```

- Note: the method is named normally, but has the additional parameter (self) as the first parameter
  - All methods that are called on an instance of an object need the self parameter

Why "self"?

- Note that the first parameter is special. It is used in the implementation of the method, but not used when the method is called. So, what is this parameter self for? Why does Python need it?
- self is a parameter that represents an object.
  - Using self, you can access instance variables in an object. Instance variables are for storing data fields.
  - Each object is an instance of a class.
  - Instance variables are tied to specific objects.
  - Each object has its own instance variables. You can use the syntax self.x to access the instance variable x for the object self in a method.

Using the Point class

- Methods are defined to accept self as the first parameter

```
class Point:
    def __init__(self, loc_x, loc_y):
            self.x = loc_x
            self.y = loc_y

     def translate(self, dx, dy):
             self.x += dx
             self.y += dy
```

- We call the method using:

  object_name.method(params)

```
from Geometry import Point
p = Point(0,0)
...
```


Exercise 1

- Write a method named *halfway(target)* which takes a Point as an argument and returns the halfway point between itself and the parameter Point.
  - For example:

```
p = Point(3, 4)
q = Point(5, 12)
r = p.halfway(q)
print(r.x, r.y) #4.0 8.0
```

Compare …

- Now, compare the midpoint() function and the halfway method
  - Midpoint takes two parameters but halfway takes one

```
p = Point(3, 4)
q = Point(5, 12)
r = midpoint(p, q)
print(r.x, r.y)
```

```
p = Point(3, 4)
q = Point(5, 12)
r = p.halfway(q)
print(r.x, r.y) #4.0 8.0
```


Exercise 2

- Define a class that will be used to represent a square with a given side length.
  - Your class should include a constructor that will allow the square to be used as follows:

```
from Geometry import Square
side = 10
s = Square(side)
```

  - Add a method to the class to calculate the perimeter of the square. The following code shows how the method may be used.

```
print (s.perimeter())
```

```
40
```


Data Field Encapsulation

- To protect data.
- To make class easy to maintain.
- To prevent direct modifications of data fields, don't let the client directly access data fields.
- This is known as data field **encapsulation**.
  - This can be done by defining private data fields. In Python, the private data fields are defined with **two leading underscores**.
  - You can also define a private method named with two leading underscores.

Example: Circle

Example04.py

- __radius : No direct access outside the Circle class

```
class Circle:
    def __init__(self, r):
        self.__radius = r

    def get_radius(self):
        return self.__radius
  ...
```

```
c = Circle(5)
print(c.__radius)
```

```
AttributeError:
'Circle' object has no
attribute '__radius'
```

```
c = Circle(5)
print(c.get_radius())
```

```
5
```

- If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, define data fields **private**.

Exercise 3

- Q1: Write a method named reflect_x() which **returns** a new Point, one which is the reflection of the point about the x-axis. For example, Point(3, 5).reflect_x() is (3, -5)