Lecture 14 - Introduction to Classes and Objects

Intro to Programming

Samia Hilal

Marianopolis College

April 21, 2022

Lesson objectives

- Introduction to Object Oriented Programming with a simple example.
- The difference between a class and an instance of a class.
- How to create instance objects.
- How to set and update class attributes and instance attributes.

Classes and objects

- Python supports a paradigm called "object-oriented programming" (OOP).
- This enables the creation of very general, reusable code.
- It supports defining *classes* or *types*.
- We can create (or instantiate) many *objects (instances)* from the same class.
- Classes can *inherit* functionality from one another.

What is a class?

- In Python: Everything is an object, Everything has a class
- A class can be defined as a blueprint/prototype or specification for creating an object
- A *class* is a type or category of an *object*.
- Depending on the specific problem domain, we might define classes to define things like:
  - Circle
  - Employee record
  - Student record
  - Player in a game
- Almost any concept can be represented by one or more classes.

What is a class?

- A class corresponds to a type or category.
- A class is described by its attributes.
  - Circle: center, radius, color
  - Employee record: name, SIN, date of birth
  - Student record: name, ID, date of birth, credits
  - Player in a game: username, skills
- An Attribute can be private to an instance, or shared across all class objects (instances).
- A class defines *methods* that operate on class instances:
  - Circles: change position, change size
  - Employee record: hire, add vacation, give raise
  - Student record: ChangeStatus, add credits
  - Player in a game: move, fight, discover

A Sample Class

```
class Student:
    """Base class for all students """
    Count = 0 #Number of instances
    def __init__(self,NAME,ID,T1,T2,st=1):
        self.name = NAME
        self.id = ID
        self.status = st #full-time=1, part-time=2
        self.t1 = T1
        self.t2 = T2
        Student.Count += 1
    def __del__(self):
        Student.Count -= 1
        print ("Deleting Student instance!")
    def __repr__(self):
        if self.status==1: stat="full-time"
        else: stat="part-time"
        format_str="{:20d} {:6s}{:10s}"
        d=format_str.format(self.id,self.name,stat)
        return(d)
```

What is an object?

- An object is a specific *instance* of a class:
  - Circle: a blue circle at 0,0 with radius 1.2
  - Employee record: John Doe, 111-222-333, 1958-01-24
  - Student record: Bob Smith, 123456, etc.
  - Player in a game: A specific player
- When we call a class method, it is usually called for a specific object of that class.
- There can be many instances of any single class.
- In Python 3, the terms object, instance, and value all mean the same thing.

Create Student Class Objects

```
>>> s1= Student("Bob Smith", 1234567,15,15)
>>> s2= Student("John Doe", 3456789,15,18,2)
>>> print(s1)
Bob Smith              1234567 full-time
>>> print(s2)
John Doe               3456789 part-time
>>> print(Student.Count)
2
```

- s1 & s2 are objects (instances) of class Student.
- The special function `__init__()` is *called implicitly* when creating new instances of a class.
- To create an instance of a class:
  Use class name and pass the arguments required by the __init__ method (without self)

More About Class Instances

- A class can keep track of number of instances.
- The attributes of a class are defined in its `__init__` method.
- Special functions like `__init__` are called *implicitly* by Python.
- Python creates an object by allocating memory for it and calling `__init__` method.
- Defining a class, does not create any objects. It describes how an object looks like when created.
- It is possible to define a class but not create any instances of that class.
- A class instance can be deleted.

Instance Attributes & Methods

- A method is a function defined in a class.
- A method has access to all the data attributes in an instance of the class.
- In the earlier example, none of the methods that can change instance attributes after an instance is created.
- The method update_status can be added to update a student status:

```
def update_status(self,new_status):
    self.status = new_status
>> s1.update_status(2) #update s1 status
>> print(s1)
Bob Smith                1234567 part-time
```

Builtin classes in Python

- The builtin types in Python are classes.
- The classes we create can mimic many of the features of the builtin types, such as:
- Implement method-style functions:

```
z = Myclass.Mymethod(x, y)
```

- Define conversions to string or other types:

```
print(str([1, 2, 3]))
```

- Re-define certain Python operators:

```
z=[0,1,2]*10 #Overload'*'with"__mul__"
```

- Re-define certain Python operators:

```
z1==z2 #Overload'=='with"__eq__"
```

Another Python Class

```
class point(object):
    '''A class for a simple 2-dimensional point.'''
    def __init__(self, new_x, new_y):
        'The constructor method.'
        self.x = new_x
        self.y = new_y

    def __eq__(self, other):
        '''Implements the '==' operator.'''
        return self.x==other.x and self.y==other.y

    def __repr__(self):
        '''Handles conversion of a point to a str.'''
        return "point({}, {})".format(self.x, self.y)

    def distance(self, other):
        'Euclidean distance to another point.'
        dx, dy = self.x - other.x, self.y - other.y
        return (dx * dx + dy * dy) ** 0.5
```

Class Statement in Python

- The header:

  ```
  class point(object):
  ```

  defines the class name, `point`, and its *base class*, the builtin `object` class.
- A class *inherits* its attributes from its base class.
- A class can have a document string on the line following the `class` keyword.
- By convention, the first argument to most class methods is called `self`.
- The `self` argument represents the object for which this method is called.

Using the circle class

- We can create circle instances a, b by calling the class name as a function:

```
>>> from point import point
>>> a = point(0, 0) # __init__(0, 0)
>>> b = point(1, 1) # __init__(1, 1)
>>> print('{:.4f}'.format(a.distance(b))
1.1412
```

- A call to `point()` creates an object and calls `__init__()` method.
- The arguments to `point()`, along with a new object, are passed to `__init__()` method.
- Methods defined using `__name__` get called implicitly. We will see many examples.

Using these methods

```
>>> x = point(0, 0)
>>> y = point(2, 1)
>>> z = x.copy()
>>> x == z    # Implicitly calls __eq__()
True
>>> x.distance(y)
2.2360679774997979
>>> z.distance(x)
0.0
>>> print(y)    # Implicitly calls __repr__()
point(2, 1)
>>> distance(x, y) # THIS WON'T WORK!!!
NameError:  name 'distance' is not defined
```

Summary

- A `class` statement defines a type.
- A class defines methods using `def` statements within the class.
- Certain special names surrounded with underscores implement internal functions.
- Instance attributes are usually defined in the `__init__()` function.
- Create an object by "calling" its class name like a function.
- Access methods and attributes of an object with the `'.'` operator.