Classes

Friday, 25 March 2022 14:32

So far we have been working with Procedural Programming, in which we work with data and functions. However, they are separated (passing data to functions, changing them, and returning, etc.

Object - Oriented Programming encapsulates Data and Functions together.

Comparision: Procedural vs Object - Oriented Programming

Procedural

```
x = 0
y = 0
y = up(y)
x, y = goto(-150, 200)
x = right(x)

print(x, y)
```

```
def up(y):
    return y + 1
def goto(x_new, y_new):
    return x_new, y_new
def right(x):
    return x + 1
```

Object Oriented

```
alex = Turtle(0,0)
alex.up()
alex.goto(-150, 100)
alex.right()
print(alex.x, alex.y)
```

1. Classes and Objects

Recall: Everything in Python is an object (variables, functions, types, ...)
We know about various classes (integer, float, string, ...)
⇒ These are all sub-classes of master class: Object!

- Classes
  - +) Can be thought of as a template for the objects that are instances of it
  - +) An instance of a class refers to an object whose type is defined as the class.
  - +) The words "instances" and "object" are used interchangibly
  - +) A class is made out of attributes (data) and methods (functions)

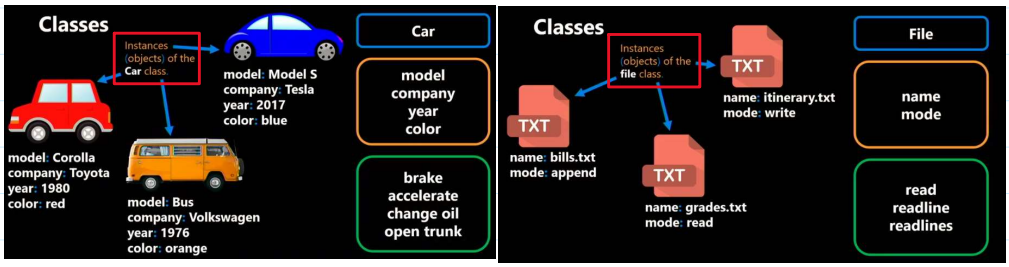
e.g. >>> isinstance("Hello", str)
True

Is this an instance of this class

Class (list)

Data → Attributes

Functions → Methods (.append)



- Objects → Instances of the class.

(e.g) A class Turtle can have multiple instances: Alex, Jennifer

To create an instance of a class, we instantiate:

(eg) `alex = Turtle(0, 0)`

2. Building your own class

⊛ General form of a Class:

- Class Name
- Constructor
- Methods

```
class Name:

    def __init__(self, param1, param2, ...):
        self.param1 = param1
        self.param2 = param2
        ...
        body

    def method1(self, parameters):
        body
        ...
```

(attributes)

+) Constructor: A code that automatically runs whenever we instantiate an object of a class.

+) Self:
- Reference to the instance of the class
- Although you do not need to use this word, it is widely adopted and is recommended.
- (more details later)

3. Encapsulation

- The core of OOP is the organization of the program by encapsulating related data or function together in an object.
- To encapsulate something means to enclose it in some kind of container.
- In programming, encapsulation means keeping data and the code that uses it

- To encapsulate something means to enclose it in some kind of container.
- In programming, encapsulation means keeping data and the code that uses it in 1 place and hiding the details of exactly how they work together

Constructors, Self, Attributes, Methods

Wednesday, 6 April 2022 16:12

Constructors

- A code that automatically runs whenever we instantiate an object of a class.
- Initiating datas (attributes) for the instantiated objects
- Not necessarily have to be included ☆
- General form of Constructors :

```
def __init__ ( self, para1, para2, ... ) :
    self.x = para1
    self.y = para2
```

(Required: self; para1, para2, ... : optional)

```
instance_name = Class ( para1, para2 )
```

(grabbing parameters input by user and assign to attributes)

Self is a overall representation (in class definition) for the objects being instantiated

☆ All methods (including constructors) in class definition MUST have "self" as parameter

However, defining methods does not cause an error. An error will occur when we call the method outside the class definition. When we do so, Python automatically pass the object as the first parameter.

↳ by using object.method()

Attributes

- Are the "characteristics" (variables) initialized as an object is instantiated to a class.
- These attributes are given to all objects in class
- Attributes can be overwritten later

```
In [1]: class Point:

            """A class that represents and manipulates 2D points"""

            def __init__(self, x=0, y=0):
                """
                (self, float, float) -> None
                Initializes a new point at (0, 0)
                """
                self.x = x
                self.y = y
```

```
In [2]: p = Point()
        print(p.x, p.y)
```

```
0 0
```

```
In [3]: q = Point(8, 15)
        print(q.x, q.y)
```

```
8 15
```

⊛ Printing instances to the screen do not print attributes. It will print the position of objects in memory.

```
In [24]: point_list = [p, q, r]
         point_list
```

```
Out[24]: [<__main__.Point at 0x2034838b148>,
          <__main__.Point at 0x2034838bc48>,
          <__main__.Point at 0x2034838e6c8>]
```

Method

- First parameter must be self
- Other parameters are the ones input by the user. The attributes, if used in methods, are already passed by calling self (since they're attributes!)
- We can even create methods between 2 objects, or return an object of any defined classes
- ☆ We can use other methods in the same class when defining a method. That "other method" may be defined before or after the method.

NOTE: When calling other methods, we must use self.method(...), not method() (since method is not defined elsewhere, other than class definition)

In [ ]:
```
class Account:

    """A class that represents a personal bank account."""

    def __init__(self, name, balance):
        """
        (self, str, numeric) -> None
        Initializes a new bank account.
        """
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        """
        (self, numeric) -> None
        """
        self.balance += amount
        self.print_balance()

    def withdraw(self, amount):
        """
        (self, numeric) -> None
        Subtracts amount from balance.
        """
        self.balance -= amount
        self.print_balance()

    def print_balance(self):
        """
        (self) -> None
        Prints the account balance.
        """
        print("Account balance is ${}.".format(self.balance))
```

- There are 2 ways to call a method:

1) Class.method(object)

2) object.method() — Python does automatically

In [20]:
```
sebs_account = Account('Sebastian', 500)
```

Now, let's add 500 dollars to my account.

In [21]:
```
sebs_account.deposit(500)
```

```
Account balance is $1000.
```

It printed the balance out after I deposited the money.

Classes in Classes

Wednesday, 6 April 2022 17:37

This is a summary of places in a class definition where we can call another object from that class, or from other defined classes

1. Objects as Data Attributes

In [2]:
```
class Square:

    """A square represent by 2 Points: lower left and upper right."""

    def __init__(self, x1=0, y1=0, x2=0, y2=0):

        """
        (self, number, number, number, number) -> None
        Initializes a point with (x1, y1) as lower left corner and
        (x2, y2) as upper-right corner. All default to zeros.
        """
        self.lower_left = Point(x1, y1)
        self.upper_right = Point(x2, y2)
```

Let's create an instance.

In [3]:
```
square = Square(0, 0, 5, 5)
print(square.lower_left.x, square.lower_left.y)
print(square.upper_right.x, square.upper_right.y)
```

```
0 0
5 5
```

2. Methods

While defining, we can can use another method for that class or a method previously defined for other classes (as long as the objects using (attributes of this class) are instances of other class

```
def centre(self):
    """
    (self) -> Point
    Returns the Point in the middle of the square
    """
    return self.upper_right.calculate_midpoint(self.lower_left)
```

3. Return

We can indeed return an object of that class or other class.

☆ Note that printing this result out will print

```
[<__main__.Point at 0x2034838b148>,
 <__main__.Point at 0x2034838bc48>,
 <__main__.Point at 0x2034838e6c8>]
```

More OOPs

Tuesday, 29 March 2022 15:10

1. File Class

When we do
```
f = open('text.txt', 'w')
f.write('hola')
f.close()
```
we are instantiating a File object named f. Then, we can implement methods like .write(), .readlines()

2. Printing Object

- We can encapsulate the printing of attributes into a separate method in class

```
def to_string(self):
    print('(' + str(self.x) + ',' + str(self.y) + ')')
```

- Alternative Python method: __str__

```
def __str__(self):
    return('(' + str(self.x) + ',' + str(self.y) + ')')
```

⇒ When we print object (self), we will print exactly what we customed in __str__

Always make use of methods already defined above