Python Lesson 3: Object Oriented Python

vanderbi.lt/py

Steve Baskauf

DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Comments

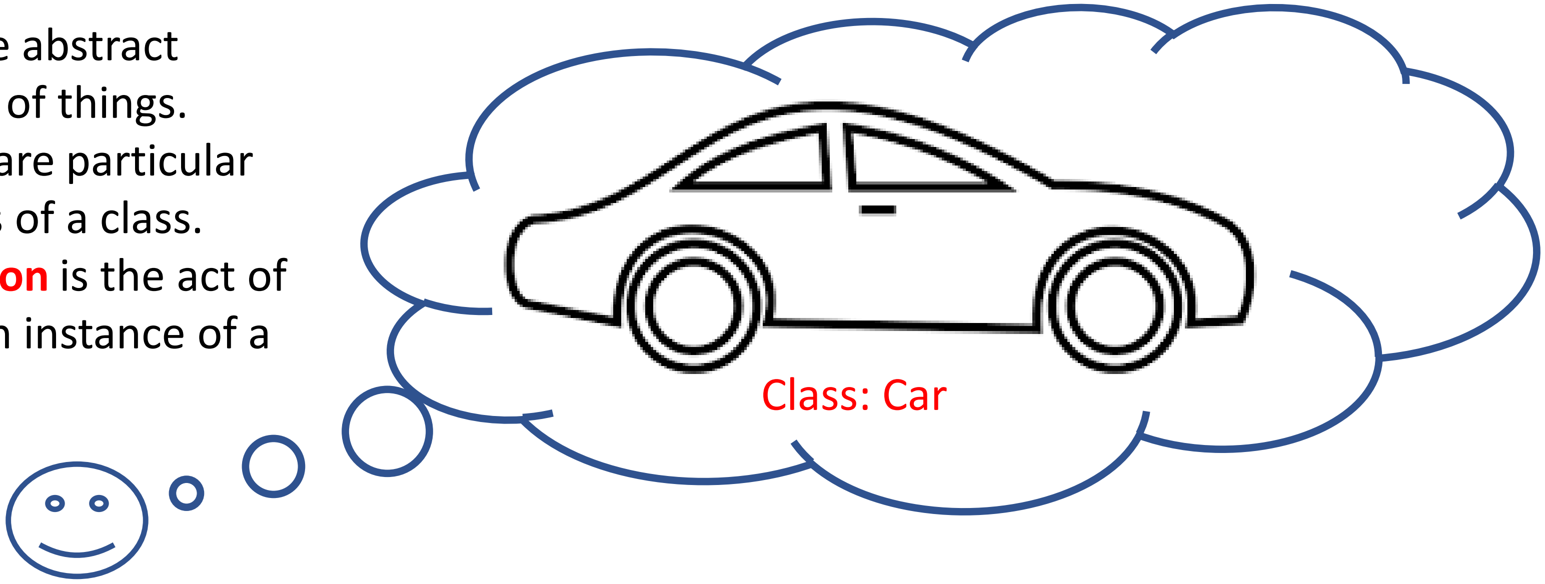
- Single-line comments use the hash (#) character
  - comment can start at beginning of line
  - comment can follow end of regular code on line
- Pseudo-multiline comment are multiline strings
  - delimit using triple single-quotes (''')
  - hash method is preferred; editors can apply to many lines

```
text = prefix + suffix
'''
print(prefix)
print(suffix)
'''
print('The whole text is: ' + text)
```

This text is "commented out"

Classes and instances

Classes are abstract categories of things. **Instances** are particular individuals of a class. **Instantiation** is the act of creating an instance of a class.



instance: toyotaPrius

instance: ferrari

instance: volkswagenBeetle

Recall: convention of **upper** camelCase for classes and **lower** camelCase for instances.

Defining and instantiating classes

- We are not going to worry about the details of defining classes.
- Classes can be defined in code we write (below) or in modules we import.
- Create class instances by writing the class name.

```
# define Duck class
class Duck:
    def __init__(self):
        # code here
        # more code

# instantiate Duck instances
donald = Duck()
daffy = Duck()
```

This is typical of how we instantiate a class (assign an instance to a named variable).

Attributes and Methods

- **Attributes** are essentially variables attached to a class.
- **Methods** are essentially functions attached to a class.

Attributes

- Attributes are essentially **variables** tied to an instance of a class.
- Attribute names follow the instance name, separated by a dot.
- In this example, all instances of the class **`Car`** have the attribute **`color`**.

`toyotaPrius.color = 'blue'` **`ferrari.color = 'red'`** **`volkswagenBeetle.color = 'white'`**

Ways to set attributes

- Instantiate, then assign attributes

```
myDuck = Duck()
myDuck.name = "Donald"
myDuck.company = "Disney"
```

- Pass attributes as arguments at instantiation
  - (need to know order of arguments)

```
myDuck = Duck("Donald", "Disney")
```

- Pass attributes as key/value pairs at instantiation
  - (order is not important)

```
myDuck = Duck(name = "Donald", company = "Disney")
myOtherDuck = Duck(company = "Warner Brothers", name = "Daffy")
```

- Available options depend on the class definition.

Try this…

- First Duck creation example
- Note that there are default attribute values.
- Notice that the `printDuck()` function does not return anything. It just "does" something. So no assignment is necessary.
- By associating the attributes with the instance, when we pass the duck instance into the function, all of the attributes go with it.

Try this…

- Second Duck creation example
- What's up with **`thirdDuck.company`** ? Use **`printDuck(thirdDuck)`** to find out.
- Default attribute values are used if no argument. Try **`printDuck(genericDuck)`**

Try this…

- Third Duck creation example
- Does **`thirdDuck.company`** get assigned correctly here? Use **`printDuck(thirdDuck)`** to find out.

Methods

- Methods are essentially **functions** tied to a class.
- We can apply a method to any instance of the class it's associated with.
- Method names follow the instance name, separated by a dot, followed by parentheses.
- Like functions, methods may or may not return any value.

```
toyotaPrius.drive('Nashville')
```

doesn't return a value.

```
newSpeed = toyotaPrius.accelerate(15)
```

returns a value.

Try this…

- **`poetry.py`** example
- Notice:
  - attributes printed in lines 39 and 41 are strings

    `.lines()` method (line 44) returns a data structure called a list (more on this next week)

    `.words()` method (line 46) returns a list of words; the **`len()`** function counts the number of items in the list

    `.abuse()` method (line 50) doesn't return anything – it modifies the poem instance itself.
  - What happens if lines 49 and 50 are switched?

GUI code from Latte Maker answer

- Note about **`tkinter`** crashing Anaconda installations.
- **`tkinter`** objects are actual objects (buttons, input boxes, etc.) on a form.
- Instances of the same class of object (e.g. **`Button`**) have the attributes and methods that make sense for that kind of object.
- Python dot notation can be confusing because methods of instances like **`firstInputBox.get()`** look similar to classes from modules like **`ttk.Button()`**. That's why capitalization is important.