Python 3 Classes

Eddie Guo

September 2019

1 Introduction to Classes

(i) Procedural vs object-oriented programming

(ii) Define new class

(iii) Instantiate new obj

(iv) Encapsulation

1.1 Procedural vs Object-Oriented Programming (OOP)

- **Procedural programming**
  - Emphasis on actions (verb)
  - Ex: **roll** dice n times, then **build** table of data
- **OOP**
  - Emphasis on objs (noun) w/ properties & behavs
  - Allows us to model real-world objs (ex: car, dog)

1.2 Example: Dogs

- unique property values define each dog (ex: age, colour, size)
- Common behavs (ex: bark, wag tail)

2 Python Class

- Class is template/blueprint
- Defines all attributes (properties) & methods (behavs) that an obj will have
  - Attributes: age, size, colour
  - Behavs: bark, wag_tail
- Obj is **instance of a class**
  - Classes give values to all attributes of an obj
- Attributes of one obj differentiates it from other objs that are instances of same class

```
# Ex: Def New Class
# dice.py
import random

class Dice: # class name is by convention capitalized
    # self must be 1st parameter for all class defs
    def __init__(self): # method def (initializes attributes of class)
        self.sides = 6  # 'sides' is an attribute

    def roll(self):
        # self is ref to the obj itself (self refers to the spec die roll obj)
        return random.randint(1, self.sides) # no need to pass attribute to method inside
    class def

    def __str__(self):
        return 'Die has ' + str(self.sides) + ' sides.'
```

```
16
17 # Ex: Instantiate Obj
18 # use_dice.py
19 from dice import Dice
20
21 def play():
22     # create new dice obj
23     my_die = Dice() # calls __init__ method
24
25     # roll my dice 3 times
26     print('Roll 1:', my_die.roll()) # use dot operator on obj to invoke method
27     print('Roll 2:', my_die.roll())
28     print('Roll 3:', my_die.roll())
29
30     # display obj
31     print(my_die) # calls __str__ method
32
33 if __name__ == "__main__":
34     play()
35
36 # Output
37 Roll 1: 3
38 Roll 2: 4
39 Roll 3: 1
40 Die has 6 sides.
```

2.1 `__init__()`

- Special method; typically used to **initialize attributes** for new obj that's created
- Automatically called when an obj is instantiated (i.e., when name of class is called)
- Aka constructor method (not quite accurate name; see here)

2.2 `__str__()` & `__repr__()`

- Both used to rep obj
- Good idea to define at least one
- `__str__` returns **informal** str rep of instance
- `__str__` is called by built-in fns `str()` & `print()`
- `__repr__` returns **official** str rep of instance
- `__repr__` called by built-in fn `repr()`

2.3 `self` Parameter

- 1st parameter in EVERY class method
- Refers to obj itself
- Don't include as argument when invoking method of obj → self passed implicitly when using dot operator on obj

```
1  # dice2.py
2  import random
3
4  class Dice:
5      def __init__(self, howMany): # pass in additional value(s) to initialize attribute(s)
6          self.sides = howMany
7
8      def roll(self):
9          return random.randint(1, self.sides)
10
11     def __str__(self):
12         return 'Die has ' + str(self.sides) + ' sides.'
13
14 # use_dice2.py
```

```
from dice2 import Dice

def play():
    # create new dice objs
    cube_die = Dice(6)
    icosahedron_die = Dice(20)

    # roll dice
    print('Cube roll:', cube_die.roll())
    print('Icosahedron roll:', icosahedron_die.roll())

    # display objs
    print(cube_die)
    print(icosahedron_die)

if __name__ = "__main__":
    play()

# Output
Cube roll: 1
Icosahdron roll: 11
Die has 6 sides.
Die has 20 sides.
```

3 Encapsulation

- A class wraps up/**encapsulates** its attributes & methods
  - Ensures all data related to an obj is contained in single struc
- Attributes can be made **private** to prevent them being accessed directly by outside programs
  - Def attribute name starting w/ '`__`'
  - Ex: `self.__sides`
- Implemment setter & getter methods to change & access attributes
  - Ctrl HOW attribute values can be changed & seen
  - Form public interface btw program & obj