Chapter 02 - Objects In Python

April 4, 2021

0.1 Creating Python Classes

- creating an `instance` of a class is a simple matter of typing the class name, followed by a pair of parentheses

0.2 Adding Attibutes

- all we need to do to assign a value to an attribute on an object is use the `<object>.<attribute> = <value>` syntax
- the `<value>` can be anything including python primitive, a built-in data type or another object, function or another class

[5]:
```
class Point:
    pass

p1 = Point()
p2 = Point()

p1.x = 5
p1.y = 4

p2.x = 3
p2.y = 6

print(f"p1.x: {p1.x}, p1.y: {p1.y}")
print(f"p2.x: {p2.x}, p2.y: {p2.y}")
```

```
p1.x: 5, p1.y: 4
p2.x: 3, p2.y: 6
```

0.3 Making It Do Something

- we have data and we want to add behaviors to our classes

[7]:
```
class Point:
    def reset(self):
        self.x = 0
        self.y = 0

p = Point()
```

```
p.reset()
print(f"p.x: {p.x}, p.y: {p.y}")
```

```
p.x: 0, p.y: 0
```

0.4 Talking To Yourself

- methods have the `self` requirement that functions do not have
- pay attention to the difference between a `class` and an `object`
- we think of the `method` as a function attached to a class
- the `self` parameter is a specific instance of that class
- when you call the method on two different objects, you are calling the same method twice, but passing two different `objects` as the `self` paramater

0.5 More Arguments

- you can call a method with arguments as such `point1.move(5, 0)`

0.6 Initializing The Object

- most object-oriented programming languages have the concept of a `constructor`, a special method that creates and initializes the object when it is created
- python has both a `constructor`, which is not generally used and an initializer, called `__init__`
- the constructor is called `__new__` and you can use it to override behaviors

[9]:
```
class Point:
    def __init__(self, x=0, y=0):
        self.move(x, y)
```

0.7 Explaining Yourself

- a docstring should clearly and concisely summarize the purpose of the class or method it is describing
- it should explain any parameters whos usage is not immediately obvious
- it is a good place to include short examples of how to use the API

[10]:
```
import math

class Point:
    """Represents a point in two-dimensional geometric coordinates"""

    def __init__(self, x=0, y=0):
        """Initializes the position of a new point"""
        self.move(x, y)
```

0.8 Organizing Modules

- a `package` is a collection of modules in a folder

- the name of the package is the name of the folder
- to tell python that a folder is a package to distinguish it from other folders in the directory, we need to place a file in the folder named `__init__.py`
- if we forget this file, we wont be able to import modules form that folder

```
parent_directory/
    main.py
    ecommerce/
        __init__.py
        database.py
        products.py
        payments/
            __init__.py
            square.py
            stripe.py
```

0.9 Absolute Imports

```
import ecommerce.products
prodduct = ecommerce.products.Product()

// or

from eccommerce.products import Products
product = Product()

//or
from ecommerce import products
product = products.Product()
```

0.10 Relative Imports

- when working with related modules inside a package, it is redundant to specifiy the full path
- thus we can use relative imports
- `from .database import Database`
- the period indicates to use the database inside the current package
- we can use multiple periods to traverse further up the hierarchy

- we can also improt code directly from packages, as oppossed to just modules inside packages
- the `__init__.py` file can contain any variable or class declarations we like and they will be available as part of the package
- instead of having to do `import ecommerce.database.db` we could do `ecommerce.db`
- generally avoid putting stuff in the `__init___.py` file

0.11 Organizing module content

- use the keyword `global` to define variables one scope above it
- like in recursive tree traversing you can have variable `global output`

- always put your start code in a function

- then have `__name__` == `"__main__":`
- every module has a `__name__` special variable that specifies the name of the module when its imported
- classes can also be put inside functions and you should do that for one-off items