Lecture 6: Classes

Adding a little class to your code.

Python Classes

A “class” is a blueprint for creating objects.

As an object-oriented programming language, virtually everything in Python is an object.

Python Classes

A “class” is a blueprint for creating objects.

As an object-oriented programming language, virtually everything in Python is an object.

Classes can have properties (variables)
Classes can have methods (functions)

Python Classes

A “class” is a blueprint for creating objects.

As an object-oriented programming language, virtually everything in Python is an object.

Classes can have properties (variables)
Classes can have methods (functions)

Creating an object using the class “blueprint” is called an “instance” of the class

Global/local review

```
a = 10

def func_one(b=20):
    c = 30
    def func_two(d=40):
        e = 50
def func_three(f=60):
    g = 70
```

Global/local review

```
a = 10

def func_one(b=20):
    c = 30
    def func_two(d=40):
        e = 50
def func_three(f=60):
    g = 70
```

Defined at the global level

Global/local review

```
a = 10

def func_one(b=20):
    c = 30
    def func_two(d=40):
        e = 50
def func_three(f=60):
    g = 70
```

Defined at the global level

Defined at the local level for func_one

Global/local review

```
a = 10

def func_one(b=20):
    c = 30
    def func_two(d=40):
        e = 50
def func_three(f=60):
    g = 70
```

Defined at the global level

Defined at the local level for func_one

Defined at the local level for func_two

Global/local review

```
a = 10

def func_one(b=20):
    c = 30
    def func_two(d=40):
        e = 50
def func_three(f=60):
    g = 70
```

Defined at the global level

Defined at the local level for func_three

Class with three properties

```
1   class MyClass():
2       a = 10
3       b = 20
4       x = a + b
5
6   mc_instance = MyClass()
```

```
In [4]: mc_instance.x
Out[4]: 30
```

Class with three properties

```
1  class MyClass():
2      a = 10
3      b = 20
4      x = a + b
5
6  mc_instance = MyClass()
```

Class properties

```
In [4]: mc_instance.x
Out[4]: 30
```

Is this class useful? Since it's all static class properties, each instance is identical.

Class with three properties

```
1  class MyClass():
2      a = 10
3      b = 20
4      x = a + b
5
6  mc_instance = MyClass()
```

Here is the same step with the standard string class

```
In [4]: mc_instance.x
Out[4]: 30
```

```
In [83]: my_string = str()

In [84]: my_string
Out[84]: ''
```

Customizing class instances

```
1   class MyClass():
2       a = 10
3       b = 20
4       x = a + b
5
6   mc_instance = MyClass()
```

```
10  class MyClass():
11      def __init__(self, a, b):
12          self.a = a
13          self.b = b
14          self.x = a + b
```

Customizing class instances

```
1   class MyClass():
2       a = 10
3       b = 20
4       x = a + b
5
6   mc_instance = MyClass()
```

```
10  class MyClass():
11      def __init__(self, a, b):
12          self.a = a
13          self.b = b
14          self.x = a + b
```

The leading and trailing double-underscore means this is a *reserved* method – Python classes know to look for this method and treat it a certain way.

Customizing class instances

```
1   class MyClass():
2       a = 10
3       b = 20
4       x = a + b
5
6   mc_instance = MyClass()
```

```
10  class MyClass():
11      def __init__(self, a, b):
12          self.a = a
13          self.b = b
14          self.x = a + b
```

Class methods always reserve the *first positional argument* for a pointer to itself.

It is named “self” purely by convention. It’s a normal positional argument that can technically be named anything.

Customizing class instances

```
10  class MyClass():
11      def __init__(self, a, b):
12          self.a = a
13          self.b = b
14          self.x = a + b
```

```
In [8]: mc_instance = MyClass(10, 20)
   ...: mc_instance.x
Out[8]: 30
```

Customizing class instances

```
10  class MyClass():
11      def __init__(self, a, b):
12          self.a = a
13          self.b = b
14          self.x = a + b
```

```
In [8]: mc_instance = MyClass(10, 20)
   ...: mc_instance.x
Out[8]: 30
```

```
In [9]: mc_instance_2 = MyClass(20, 20)
   ...: mc_instance_2.x
Out[9]: 40
```

Customizing class instances

```
10  class MyClass():
11      def __init__(self, a, b):
12          self.a = a
13          self.b = b
14          self.x = a + b
```

```
In [10]: type(mc_instance)
Out[10]: __main__.MyClass

In [11]: type(mc_instance_2)
Out[11]: __main__.MyClass
```

```
In [8]: mc_instance = MyClass(10, 20)
   ...: mc_instance.x
Out[8]: 30
```

```
In [9]: mc_instance_2 = MyClass(20, 20)
   ...: mc_instance_2.x
Out[9]: 40
```

Adding methods to a class

Create a class that starts with values for the number of bedrooms, bathrooms, and the square footage.

Add a method that uses these three values to calculate a home price.

Then add a method that randomly picks a markup for the neighborhood and modifies the result

First, plan your class out!

```
class HouseValues():
    def __init__(self):
        # needs three values: num bedrooms, num bathrooms, sqft
        pass
```

First, plan your class out!

```
class HouseValues():
    def __init__(self):
        # needs three values: num bedrooms, num bathrooms, sqft
        pass

    def estimate_value(self):
        # use an equation of questionable accuracy that I found on a random
        # website to estimate the value based on those three parameters
        pass
```

First, plan your class out!

```
class HouseValues():
    def __init__(self):
        # needs three values: num bedrooms, num bathrooms, sqft
        pass

    def estimate_value(self):
        # use an equation of questionable accuracy that I found on a random
        # website to estimate the value based on those three parameters
        pass

    def pick_a_neighborhood(self):
        # randomly pick a modifier to multiple the value estimate by
        pass
```

Adding methods to a class

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
```

Adding methods to a class

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
28
29      def estimate_value(self):
30          #add 10% per num of bedrooms over 1
31          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
32
33          #add 5% per num of baths over 1
34          bath_mod = ((self.num_baths - 1) * 0.05) + 1
35
36          self.value = (self.sqft * 400) * bedroom_mod * bath_mod
37          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

Adding methods to a class

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
28
29      def estimate_value(self):
30          #add 10% per num of bedrooms over 1
31          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
32
33          #add 5% per num of baths over 1
34          bath_mod = ((self.num_baths - 1) * 0.05) + 1
35
36          self.value = (self.sqft * 400) * bedroom_mod * bath_mod
37          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

```
39  house1 = HouseValues(2, 2, 950)
40  house2 = HouseValues(1, 1, 700)
```

Follow the variable

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
28
29      def estimate_value(self):
30          #add 10% per num of bedrooms over 1
31          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
32
33          #add 5% per num of baths over 1
34          bath_mod = ((self.num_baths - 1) * 0.05) + 1
35
36          self.value = (self.sqft * 400) * bedroom_mod * bath_mod
37          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

```
39  house1 = HouseValues(2, 2, 950)
40  house2 = HouseValues(1, 1, 700)
```

Follow the variable

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
28
29      def estimate_value(self):
30          #add 10% per num of bedrooms over 1
31          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
32
33          #add 5% per num of baths over 1
34          bath_mod = ((self.num_baths - 1) * 0.05) + 1
35
36          self.value = (self.sqft * 400) * bedroom_mod * bath_mod
37          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

```
39  house1 = HouseValues(2, 2, 950)
40  house2 = HouseValues(1, 1, 700)
```

Follow the variable

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
28
29      def estimate_value(self):
30          #add 10% per num of bedrooms over 1
31          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
32
33          #add 5% per num of baths over 1
34          bath_mod = ((self.num_baths - 1) * 0.05) + 1
35
36          self.value = (self.sqft * 400) * bedroom_mod * bath_mod
37          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

```
39  house1 = HouseValues(2, 2, 950)
40  house2 = HouseValues(1, 1, 700)
```

Adding methods to a class

```
23  class HouseValues():
24      def __init__(self, num_bedrooms, num_baths, sqft):
25          self.num_bedrooms = num_bedrooms
26          self.num_baths = num_baths
27          self.sqft = sqft
28
29      def estimate_value(self):
30          #add 10% per num of bedrooms over 1
31          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
32
33          #add 5% per num of baths over 1
34          bath_mod = ((self.num_baths - 1) * 0.05) + 1
35
36          self.value = (self.sqft * 400) * bedroom_mod * bath_mod
37          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

```
39  house1 = HouseValues(2, 2, 950)
40  house2 = HouseValues(1, 1, 700)
```

```
In [23]: house1.estimate_value()
I estimate this house will be worth $438900.0

In [24]: house2.estimate_value()
I estimate this house will be worth $280000.0
```

Adding methods to a class

```
45  from numpy import random
```

?

```
53      def pick_a_neighborhood(self):
54          value = random.normal(1, 0.1)
55          if value > 1.2:
56              print('Whoa, you got an expensive neighborhood!')
57          elif value > 1:
58              print('Fairly pricy neighborhood.')
59          elif value < 1:
60              print('Maybe not the nicest neighborhood.')
61          return value
```

Adding methods to a class

```
63      def estimate_value(self):
64          #add 10% per num of bedrooms over 1
65          bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1
66
67          #add 5% per num of baths over 1
68          bath_mod = ((self.num_baths - 1) * 0.05) + 1
69
70          n_mod = self.pick_a_neighborhood()
71
72          self.value = (self.sqft * 400) * bedroom_mod * bath_mod * n_mod
73          print(f'I estimate this house will be worth ${round(self.value, 2)}')
```

```
75  house1 = HouseValues(2, 2, 950)
76  house2 = HouseValues(1, 1, 700)
```

```
In [28]: house1.estimate_value()
Fairly pricy neighborhood.
I estimate this house will be worth $513187.73

In [29]: house2.estimate_value()
Maybe not the nicest neighborhood.
I estimate this house will be worth $260774.86
```

```
class HouseValues():
    def __init__(self, num_bedrooms, num_baths, sqft):
        self.num_bedrooms = num_bedrooms
        self.num_baths = num_baths
        self.sqft = sqft

    def pick_a_neighborhood(self):
        value = random.normal(1, 0.1)
        if value > 1.2:
            print('Whoa, you got an expensive neighborhood!')
        elif value > 1:
            print('Fairly pricy neighborhood.')
        elif value < 1:
            print('Maybe not the nicest neighborhood.')
        return value

    def estimate_value(self):
        #add 10% per num of bedrooms over 1
        bedroom_mod = ((self.num_bedrooms - 1) * 0.1) + 1

        #add 5% per num of baths over 1
        bath_mod = ((self.num_baths - 1) * 0.05) + 1

        n_mod = self.pick_a_neighborhood()

        self.value = (self.sqft * 400) * bedroom_mod * bath_mod * n_mod
        print(f'I estimate this house will be worth ${round(self.value, 2)}')
```