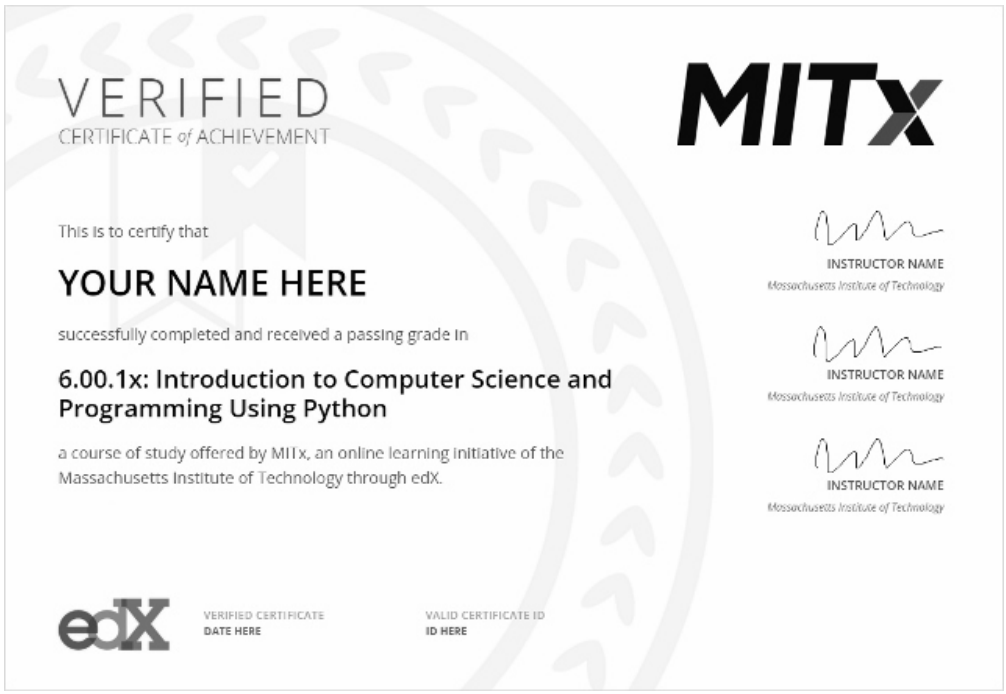
Verified Certificates



Interested in using an edX certificate to bolster a college application or to advance your career? With an edX verified certificate, you can demonstrate to colleges, employers, and colleagues that you successfully completed a challenging edX course, while helping to support the edX non-profit mission. You can register to earn a Verified Certificate directly here.

Video: Object Oriented Programming

Hide Discussion

Topic: Lecture 9 / Video: Object Oriented Programming

Add a Post

All Posts

[Tutorial] Object Oriented Programming: Methods BEHIND THE SCENES!

discussion posted 4 months ago by **Kiara-Elizabeth** (Community TA)

Object Oriented Programming: Methods

Hi! Welcome to this second part of the Object Oriented Programming Tutorial.

In this section we are going to study very powerful concepts in Object Oriented Programming. **METHODS!**

Let's get started!

Remember the example we worked with in the first part of the tutorial? If not, don't worry, here we have the code to refresh your memory.

- We covered naming the class, the `__init__( )` function and what instances are and how to create them.
- Now we will discuss what **METHODS** are.

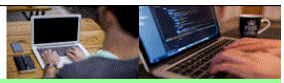
The functions below `__init__( )` must be familiar to you, they are exactly like function we have been working with, BUT notice that now, these functions are inside what we call a class.

This changes the scenario a little bit, but you must remember that they are still functions that can be called with an input, if necessary.

Let's dive into METHODS!

- When you create an instance, that instance will have access to all the methods the class has. The keyword `self` will store a reference to the instance so the instance will be able to call these functions and these functions can act on the its own individual attributes to modify them.

Example we'll be working with!



```
1  class House(object):
2      def __init__(self, street, rooms, bathrooms):
3          self.street = street
4          self.rooms = rooms
5          self.bathrooms = bathrooms
6          # Not provided as input
7          self.clean = True
8
9      def cleanHouse(self):
10         if not self.clean:
11             self.clean = True
12             print("This house is now clean")
13         else:
14             print("This house is already clean")
15
16     def unCleanHouse(self):
17         if self.clean:
18             self.clean = False
19             print("This house is now dirty")
20         else:
21             print("This house was dirty already")
22
23     def talk(self, phrase):
24         print(phrase)
25
26 house1 = House(35, 15, 16)
```

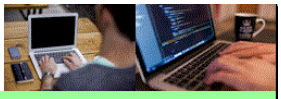
Name and Inheritance

__init__()

Methods

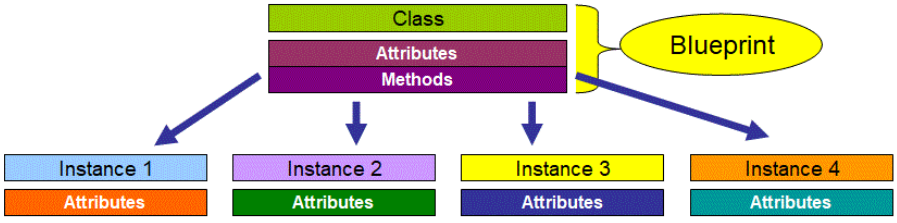
Instance

What are methods?



Concept

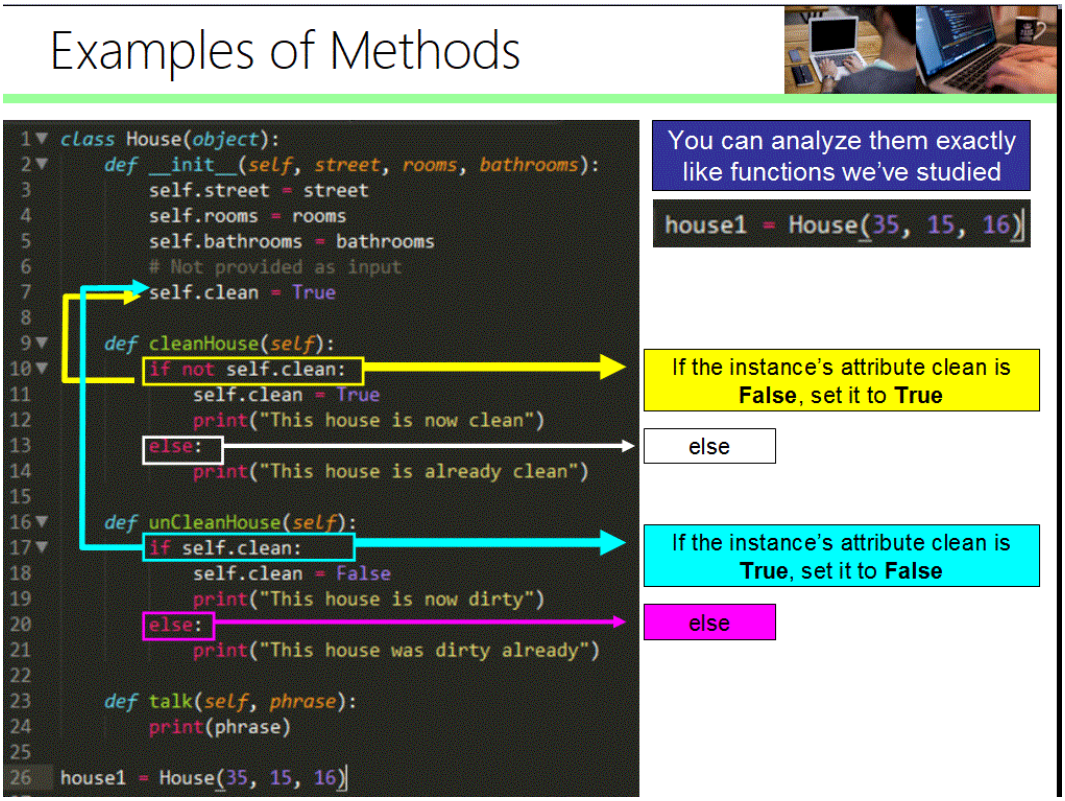
- You can think of methods as functions that an instance can call.
- In the real world, If you're an instance of a Class "Human", one of your methods could be "breath", for example.
- They are "Actions" or "functions" that belong to each instance and they can act or modify the instance's attributes
- Methods are part of the initial blueprint for instances. **All instances of a class have access to its methods**



BUT if you call a method for a specific instance and that method modifies an attribute, **it will only modify the instance's attribute**, not all the instances attribute.

You can read the code inside methods exactly like you would read normal functions code, the only difference is that their functionality changes a little bit because they belong an instance of a class.

We can access and change an instance's attribute with a method by referring to it using *self.attributeName*



Calling Methods

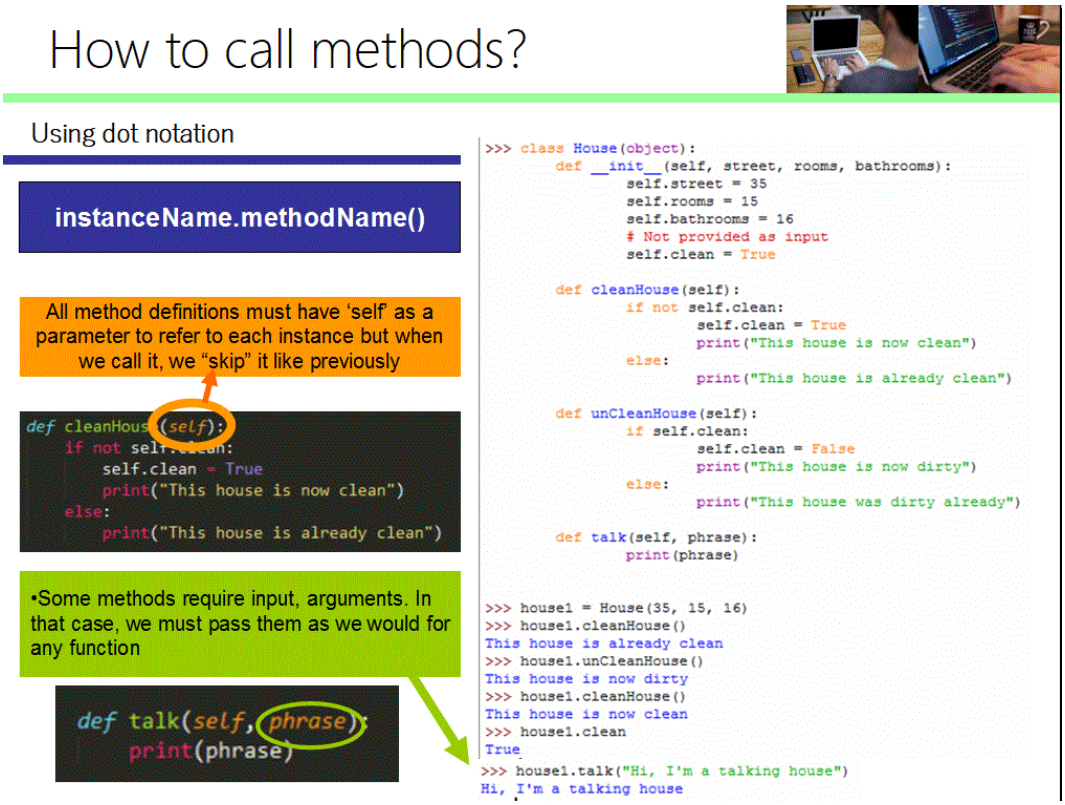
Now, let's discuss how to **call methods**.

Remember that they are functions? Well, functions are not very useful if we can't call them, so in this case we have a way of calling functions that belong to a class.

- We use **instanceName.methodName( )**

The parenthesis is **VERY IMPORTANT** because it means that we want to RUN the method instead of returning the method itself, the method definition.

NOTE: Yes, If you've noticed we have a talking house, you're not crazy, we do have a talking house hahaha (one gets really creative with these examples, right? :-))



NOTE: The values of the attributes have been hard-coded, the will initially be the same for all instances, no matter what parameters we pass. If you want to assign custom values, you must assign the parameter to `self.<attribute>`.

Examples

To demonstrate what I mean with every instance having access to the class methods but that methods act on the instance's individual attributes, here we have two instances of the class House.

```
>>> house1 = House(35, 15, 16)
>>> house2 = House(50, 25, 86)
>>> house1.clean
True
>>> house2.clean
True
>>> house1.unCleanHouse()
This house is now dirty
>>> house1.clean
False
>>> house2.clean
True
```

- We first check to see that their attribute clean is the same, since all houses are initially clean according to our class definition.
- Then, we call the instance house1 calls the method unCleanHouse. The method changes the attribute self.clean to the opposite Boolean

- Now we check the `clean` attribute again on both instances, and now we notice that **the attribute has changed for the instance** that called the method unCleanHouse.

```
def __init__(self, street, rooms, bathrooms):
    self.street = 35
    self.rooms = 15
    self.bathrooms = 16
    # Not provided as input
    self.clean = True
```

```
def cleanHouse(self):
    if not self.clean:
        self.clean = True
        print("This house is now clean")
    else:
        print("This house is already clean")

def unCleanHouse(self):
    if self.clean:
        self.clean = False
        print("This house is now dirty")
    else:
        print("This house was dirty already")
```

```
>>> house1 = House(35, 15, 16)
>>> house2 = House(50, 25, 86)
>>> house1.clean
True
>>> house2.clean
True
>>> house1.unCleanHouse()
This house is now dirty
>>> house1.clean
False
>>> house2.clean
True
```

BUT that same attribute for house2 has remained intact.

Calling the method on house1 had no effect on house2 because each one has access to methods that act on their own individual attributes and don't affect any other instances.

There are other types of methods such as **Static Methods** and **Class Methods**. If you'd like to learn more about them:

- https://realpython.com/blog/python/instance-class-and-static-methods-demystified/

- http://stupidpythonideas.blogspot.com/2013/06/how-methods-work.html (thanks to: Kiwitrader)
- https://docs.python.org/3.6/howto/descriptor.html (thanks to: Kiwitrader)

Hope it helps!

If you have any questions, don't hesitate to ask, community TAs and your classmates will be there to help you in the forums.

Estefania.

This post is visible to everyone.

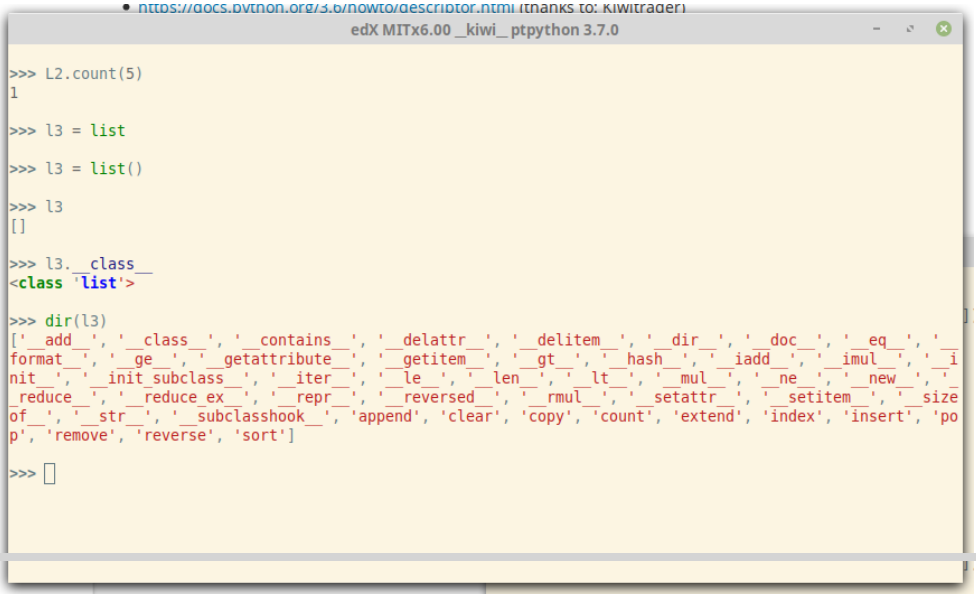
Add a Response

2 responses

jlessey
4 months ago

Great Tutorials!

Is there a way to see, for example, all methods that a **list** or a **dictionary** has, but the same way that is presented in this tutorial? Like seeing all the information of the type "List", from its definition, the very initial part of the creation of that type, through all its methods as functions.



posted 4 months ago by **kiwitrader** (Community TA)

Ok cool, a list of methods.

posted 4 months ago by **jlessey**

help(list)

posted 3 months ago by **LarryW69**

Add a comment

bsoe8019
3 months ago

Thank you. You're amazing.

Thank you very much for your very kind words. I'm very glad it helped 😊

Estefania.

posted 3 months ago by **Kiara-Elizabeth** (Community TA)

Add a comment

Showing all responses

Add a response: