Theory: Class instances

28 minutes 8 / 8 problems solved

Start practicing

6951 users solved this topic. Latest completion was **about 2 hours ago**.

By now, you already know what classes are and how they're created and used in Python. Now let's get into the details about **class instances.**

A class instance is an object of the class. If, for example, there was a class **River,** we could create such instances as Volga, Seine, and Nile. They would all have the same structure and share all class attributes defined within the class River.

However, initially, all instances of the class would be identical to one another. Most of the time that is not what we want. To customize the initial state of an instance, the `__init__` method is used.

§1. def __init__()

The `__init__` method is a **constructor**. Constructors are a concept from the object-oriented programming. A class can have one and only one constructor. If `__init__` is defined within a class, it is automatically invoked when we create a new class instance. Take our class River as an example:

```
class River:
    # list of all rivers
    all_rivers = []

    def __init__(self, name, length):
        self.name = name
        self.length = length
        # add current river to the list of all rivers
        River.all_rivers.append(self)

volga = River("Volga", 3530)
seine = River("Seine", 776)
nile = River("Nile", 6852)

# print all river names
for river in River.all_rivers:
    print(river.name)
# Output:
# Volga
# Seine
# Nile
```

We created three instances (or objects) of the class River: `volga`, `seine`, and `nile`. Since we defined **name** and **length** parameters for the `__init__`, they must be explicitly passed when creating new instances. So something like `volga = River()` would cause an error.

The `__init__` method specifies what attributes we want the instances of our class to have from the very beginning. In our example, they are **name** and **length.**

§2. self

Current topic:

✓ Class instances Stage 1 3★ ···

Topic depends on:

✓ Scopes Stage 1 5★ ···

✓ Class Stage 1 3★ ···

Topic is required for:

✓ Class vs instance 3★ ···

✓ Methods Stage 1 3★ ···

Datetime module Stage 2 ···

Table of contents:

You may have noticed that our `__init__` method had another argument besides name and length: `self`. The `self` argument represents a particular instance of the class and allows us to access its attributes and methods. In the example with `__init__`, we basically create attributes for the particular instance and assign the values of method arguments to them. It is important to use the `self` parameter inside the method if we want to save the values of the instance for later use.

Most of the time we also need to write the self parameter in other methods because when the method is called the first argument that is passed to the method is the object itself. Let's add a method to our River class and see how it works. The syntax of the methods is not of importance at the moment, just pay attention to the use of the `self`:

```
class River:
    all_rivers = []

    def __init__(self, name, length):
        self.name = name
        self.length = length
        River.all_rivers.append(self)

    def get_info(self):
        print("The length of the {0} is {1} km".format(self.name, self.length))
```

Now if we call this method with the objects we've created we will get this:

```
volga.get_info()
# The length of the Volga is 3530 km
seine.get_info()
# The length of the Seine is 776 km
nile.get_info()
# The length of the Nile is 6852 km
```

As you can see, for each object the `get_info()` method printed its particular values and that was possible because we used the `self` keyword in the method.

Note that when we actually call an object's method we don't write the `self` argument in the brackets. The `self` parameter (that represents a particular instance of the class) is passed to the instance method **implicitly** when it is called. So there are actually two ways to call an instance method: `self.method()` or `class.method(self)`. In our example it would look like this:

```
# self.method()
volga.get_info()
# The length of the Volga is 3530 km

# class.method(self)
River.get_info(volga)
# The length of the Volga is 3530 km
```

§3. Instance attributes

Classes in Python have two types of attributes: class attributes and instance attributes. You should already know what class attributes are so here we'll focus on the instance attributes. **Instance attributes** are defined within methods and they store instance-specific information.

In the class River, the attributes **name** and **length** are instance attributes since they are defined within a method (`__init__`) and have `self` before them. Usually, instance attributes are created within the `__init__` method since it's the constructor, but you can define instance attributes in other methods as well. However, it's not recommended so we advise you to stick to the `__init__`.

Instance attributes are available only from the scope of the object which is why this code will produce a mistake:

```
print(River.name)  # AttributeError
```

Instance attributes, naturally, are used to distinguish objects: their values are different for different instances.

```
volga.name  # "Volga"
seine.name  # "Seine"
nile.name   # "Nile"
```

So when deciding which attributes to choose in your program, you should first decide whether you want it to store values unique to each object of the class or, on the contrary, the ones shared by all instances.

§4. Summary

In this topic, you've learned about class instances.

If classes are an abstraction, a template for similar objects, a **class instance** is a sort of example of that class, a particular object that follows the structure outlined in the class. In your program, you can create as many objects of your class as you need.

To create objects with different initial states, classes have a constructor `__init__` that allows us to define necessary parameters. Reference to a particular instance within methods is done through the `self` keyword. Within the `__init__` method, we define instance attributes that are different for all instances.

Most of the time in our programs we'll deal not with the classes directly, but rather with their instances, so knowing how to create them and work with them is very important!

Report a typo

Thanks for your feedback!

Start practicing

This content was created over 1 year ago and updated 6 days ago. Share your feedback below in comments to help us improve it!

Comments (17) Hints (1) Useful links (1) Show discussion