



Object-oriented Programming (Optional)

- ✓ **Video:** OOP Introduction (Optional)
1 min
- ✓ **Video:** What is Object-oriented programming? (Optional)
3 min
- ✓ **Reading:** Object-Oriented Programming Defined
10 min
- ✓ **Video:** Classes and Objects in Python (Optional)
4 min
- ✓ **Reading:** Classes and Objects in Detail
10 min
- ✓ **Video:** Defining New Classes (Optional)
4 min
- ✓ **Reading:** Defining Classes (Optional)
10 min
- ✓ **Practice Quiz:** Practice Quiz: Object-oriented Programming (Optional)



Special Methods

Instead of creating classes with empty or default values, we can set these values when we create the instance that we don't miss an important value and avoids a lot of unnecessary lines of code. To do this, we use a special method called a **constructor**. Below is an example of an Apple class with a constructor method defined.

```
4 ... self.flavor = flavor
```

When you call the name of a class, the constructor of that class is called. This constructor method is always called. You might remember that special methods start and end with two underscore characters. In our example, the constructor method takes the self variable, which represents the instance, as well as color and flavor parameters. These parameters are then used by the constructor method to set the values for the current instance. So we can create an instance of the Apple class and set the color and flavor values all in one go:

```
1 >>> jonagold = Apple("red", "sweet")
2 >>> print(jonagold.color)
3 Red
```

In addition to the `__init__` constructor special method, there is also the `__str__` special method. This method defines how an instance of an object will be printed when it's passed to the `print()` function. If an object does not have a special method defined, it will wind up using the default representation, which will print the position of the object.