OK, now it’s your turn! Have a go at writing methods for a class. Create a Dog class with dog\_years based on the Piglet class shown before (one human year is about 7 dog years).

class Dog:

  years = 0

  def dog\_years(self):

    return self.years \* 7

fido=Dog()

fido.years=3

print(fido.dog\_years())

RunReset

Here is your output:

21

Awesome! You've now learned about writing your own methods!

What Is a Method?

Calling methods on objects executes functions that operate on attributes of a specific instance of the class. This means that calling a method on a list, for example, only modifies that instance of a list, and not all lists globally. We can define methods within a class by creating functions inside the class definition. These instance methods can take a parameter called **self** which represents the instance the method is being executed on. This will allow you to access attributes of the instance using dot notation, like **self.name**, which will access the name attribute of that specific instance of the class object. When you have variables that contain different values for different instances, these are called instance variables.

## Special Methods

Instead of creating classes with empty or default values, we can set these values when we create the instance. This ensures 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.

>>> class Apple:

...     def \_\_init\_\_(self, color, flavor):

...         self.color = color

...         self.flavor = flavor

When you call the name of a class, the constructor of that class is called. This constructor method is always named **\_\_init\_\_**. You might remember that special methods start and end with two underscore characters. In our example above, 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 now create a new instance of the Apple class and set the color and flavor values all in go:

>>> jonagold = Apple("red", "sweet")

>>> print(jonagold.color)

Red

In addition to the **\_\_init\_\_**constructor special method, there is also the **\_\_str\_\_** special method. This method allows us to define how an instance of an object will be printed when it’s passed to the print() function. If an object doesn’t have this special method defined, it will wind up using the default representation, which will print the position of the object in memory. Not super useful. Here is our Apple class, with the **\_\_str\_\_**method added:

class Apple:

...     def \_\_init\_\_(self, color, flavor):

...         self.color = color

...         self.flavor = flavor

...     def \_\_str\_\_(self):

...         return "This apple is {} and its flavor is {}".format(self.color, self.flavor)

...

Now, when we pass an Apple object to the print function, we get a nice formatted string:

>>> jonagold = Apple("red", "sweet")

>>> print(jonagold)

This apple is red and its flavor is sweet

This apple is red and its flavor is sweet

It's good practice to think about how your class might be used and to define a \_\_str\_\_ method when creating objects that you may want to print later.

## Documenting with Docstrings

The Python **help** function can be super helpful for easily pulling up documentation for classes and methods. We can call the **help** function on one of our classes, which will return some basic info about the methods defined in our class:

 >>> class Apple:

...     def \_\_init\_\_(self, color, flavor):

...         self.color = color

...         self.flavor = flavor

...     def \_\_str\_\_(self):

...         return "This apple is {} and its flavor is {}".format(self.color, self.flavor)

...

>>> help(Apple)

Help on class Apple in module \_\_main\_\_:

class Apple(builtins.object)

 |  Methods defined here:

 |

 |  \_\_init\_\_(self, color, flavor)

 |      Initialize self.  See help(type(self)) for accurate signature.

 |

 |  \_\_str\_\_(self)

 |      Return str(self).

 |

 |  ----------------------------------------------------------------------

 |  Data descriptors defined here:

 |

 |  \_\_dict\_\_

 |      dictionary for instance variables (if defined)

 |

 |  \_\_weakref\_\_

 |      list of weak references to the object (if defined)

We can add documentation to our own classes, methods, and functions using **docstrings**. A docstring is a short text explanation of what something does. You can add a docstring to a method, function, or class by first defining it, then adding a description inside triple quotes. Let's take the example of this function:

>>> def to\_seconds(hours, minutes, seconds):

...     """Returns the amount of seconds in the given hours, minutes and seconds."""

...     return hours\*3600+minutes\*60+seconds

...

We have our function called to\_seconds on the first line, followed by the docstring which is indented to the right and wrapped in triple quotes. Last up is the function body. Now, when we call the help function on our to\_seconds function, we get a handy description of what the function does:

help(to\_seconds)

Help on function to\_seconds in module \_\_main\_\_:

to\_seconds(hours, minutes, seconds)

    Returns the amount of seconds in the given hours, minutes and seconds.

Docstrings are super useful for documenting our custom classes, methods, and functions, but also when working with new libraries or functions. You'll be extremely grateful for docstrings when you have to work with code that someone else wrote!

## Classes and Methods Cheat Sheet (Optional)

## Classes and Methods Cheat Sheet

In the past few videos, we’ve seen how to define classes and methods in Python. Here, you’ll find a run-down of everything we’ve covered, so you can refer to it whenever you need a refresher.

### **Defining classes and methods**

class ClassName:

    def method\_name(self, other\_parameters):

        body\_of\_method

**Classes and Instances**

* Classes define the behavior of all instances of a specific class.
* Each variable of a specific class is an instance or object.
* Objects can have attributes, which store information about the object.
* You can make objects do work by calling their methods.
* The first parameter of the methods (self) represents the current instance.
* Methods are just like functions, but they can only be used through a class.

### **Special methods**

* Special methods start and end with \_\_.
* Special methods have specific names, like \_\_init\_\_ for the constructor or \_\_str\_\_ for the conversion to string.

### **Documenting classes, methods and functions**

* You can add documentation to classes, methods, and functions by using docstrings right after the definition. Like this:

class ClassName:

    """Documentation for the class."""

    def method\_name(self, other\_parameters):

        """Documentation for the method."""

        body\_of\_method

def function\_name(parameters):

    """Documentation for the function."""

    body\_of\_function

Help with Jupyter Notebooks (Optional)

Help with Jupyter Notebooks

We've aimed to make our Jupyter notebooks easy to use. But, if you get stuck, you can find more information [here](https://learner.coursera.help/hc/en-us/articles/360004995312-Solve-problems-with-Jupyter-Notebooks).

If you still need help, the discussion forums are a great place to find it! Use the forums to ask questions and source answers from your fellow learners.

If you want to learn more about Jupyter Notebooks as a technology, check out these resources:

* [Jupyter Notebook Tutorial](https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook), by datacamp.com
* [How to use Jupyter Notebooks](https://www.codecademy.com/articles/how-to-use-jupyter-notebooks), by codeacademy.com
* [Teaching and Learning with Jupyter](https://jupyter4edu.github.io/jupyter-edu-book/), by university professors using Jupyter

notebook link : <https://wtrkvaxqrwsfcctjfjqcog.coursera-apps.org/notebooks/C1M5L2_Methods_and_Classes_V3.ipynb>