Google

# Reading: More about object-oriented programming

**Note:** This reading contains only a  brief introduction to object-oriented programming. A more detailed discussion about the nuances of object oriented programming is beyond the scope of this course.

Previously, we identified object-oriented programming as a programming paradigm that is based around objects, which can contain both data and code that manipulates that data. You may recall that a class is an object's data type that bundles data and functionality together, and you've encountered some examples of this class-specific functionality in the form of methods and attributes. In this reading, you're going to learn more about object-oriented programming and how it works. Although this certificate program will not require you to define your own classes, having a basic understanding of how this process works will be very helpful when you encounter these concepts along your learning journey.

## Review: Attributes and methods

Python classes are powerful and convenient because they come with built-in features that simplify common data analysis tasks. These features are known as attributes and methods.

- **Attribute**: A value associated with an object or class which is referenced by name using dot notation.
- **Method**: A function that belongs to a class and typically performs an action or operation.

A simpler way of thinking about the distinction between attributes and methods is to remember that attributes are *characteristics* of the object, while methods are *actions* or *operations*.

For example, if the class were Spaceship, then attributes might be:

**name**

**kind**

**speed**

**tractor_beam**

These attributes could be accessed by typing:

**Spaceship.name**

`Spaceship.kind`

`Spaceship.speed`

`Spaceship.tractor_beam`

Notice that these characteristics are accessed using only a dot.

On the other hand, methods of the Spaceship class might be:

`warp()`

`tractor()`

These methods could be used by typing:

`Spaceship.warp()`

`Spaceship.tractor()`

Notice that methods are followed by parentheses, and it's possible for them to take arguments. For example, `Spaceship.warp(7)` could change the speed of the ship to warp seven.

## Defining classes with unique attributes and methods

Python lets you define your own classes, each with their own special attributes and methods. This helps all different kinds of programmers to build reusable code that makes their work more efficient. You can even build the Spaceship class mentioned previously. The example, here, demonstrates how to do this.

**Note:** The following code block is not interactive.

```
class Spaceship:
  # Class attribute
  tractor_beam = 'off'


  # Instance attributes
  def __init__(self, name, kind):
      self.name = name
      self.kind = kind
      self.speed = None
```

```
 # Instance methods
  def warp(self, warp):
     self.speed = warp
     print(f'Warp {warp}, engage!')


  def tractor(self):
     if self.tractor_beam == 'off':
        self.tractor_beam = 'on'
        print('Tractor beam on.')
     else:
        self.tractor_beam = 'off'
        print('Tractor beam off')
```

For this course you don't have to learn the syntax to create classes. Just notice that the class itself is defined first, and then indented beneath it are the attributes and methods. This is what it means when an attribute or method "belongs to a class." Attributes and methods are defined in the code for that class.

A class is like a blueprint for all things that share characteristics and behaviors. In this case, the class is Spaceship. There can be all different kinds of spaceships. They can have different names and different purposes. Whenever you create an object of a given class, you're creating an **instance** of that class. This is also known as **instantiating** the class. In the code above, every time you instantiate an object of the Spaceship class it will start with its tractor beam set to off. The tractor beam is a class attribute. All instances of the Spaceship class have one. There are also instance attributes. These are attributes that you can assign when you instantiate the object.

```
# Create an instance of the Spaceship class (i.e. "instantiate")
ship = Spaceship('Mockingbird','rescue frigate')

# Check ship's name
print(ship.name)

# Check what kind of ship it is
print(ship.kind)

# Check tractor beam status
print(ship.tractor_beam)
```

**Output:**

```
Mockingbird
rescue frigate
off
```

The next block of code uses the **warp()** method to set the warp speed to seven. Then it checks the current speed of the ship using the speed attribute.

```
# Set warp speed
ship.warp(7)

# Check speed
ship.speed
```

**Output:**

```
Warp 7, engage!
7
```

This final block of code uses the **tractor()** method to toggle the tractor beam. Then it checks the current status of the tractor beam using the **tractor_beam** attribute.

```
# Toggle tractor beam
ship.tractor()

# Check tractor beam status
print(ship.tractor_beam)
```

**Output:**

```
Tractor beam on.
on
```

This is just a basic example meant to demonstrate some of the fundamental ways that classes, attributes, and methods work and how they relate to each other, but classes can be very complex and have many attributes and methods. Depending on the work you're doing as a data professional, knowledge about object-oriented programming will be helpful as you define your own classes, attributes, and methods to investigate the patterns, relationships, and meaning data holds.

## Key takeaways

Classes comprise the core objects of Python, which is why Python is known as an object-oriented language. Class objects are powerful because they contain unique tools designed specifically for that class packaged within them. Methods are functions that belong to a class; they perform actions or operations, and they use parentheses. Attributes are values or characteristics associated with a class or class instance; they do not use parentheses. And, while there are many classes, attributes, and methods pre-built into Python, there is a high level of customization offered in the object-oriented programming paradigm.

---