**Lesson Outline**

* Object-oriented programming syntax
  + procedural vs object-oriented programming
  + classes, objects, methods and attributes
  + coding a class
  + magic methods
  + inheritance
* Using object-oriented programming to make a Python package
  + making a package
  + tour of scikit-learn source code
  + putting your package on PyPi

**Why Object-Oriented Programming?**

Object-oriented programming has a few benefits over procedural programming, which is the programming style you most likely first learned. As you'll see in this lesson,

* object-oriented programming allows you to create large, modular programs that can easily expand over time;
* object-oriented programs hide the implementation from the end-user.

Consider Python packages like **[Scikit-learn](https://github.com/scikit-learn/scikit-learn" \t "_blank)**, [**pandas**](https://pandas.pydata.org/), and **[NumPy](http://www.numpy.org/" \t "_blank)**. These are all Python packages built with object-oriented programming. Scikit-learn, for example, is a relatively large and complex package built with object-oriented programming. This package has expanded over the years with new functionality and new algorithms.

When you train a machine learning algorithm with Scikit-learn, you don't have to know anything about how the algorithms work or how they were coded. You can focus directly on the modeling.

Here's an example taken from the **[Scikit-learn website](http://scikit-learn.org/stable/modules/svm.html" \t "_blank)**:

from sklearn import svm

X = [[0, 0], [1, 1]]

y = [0, 1]

clf = svm.SVC()

clf.fit(X, y)

How does Scikit-learn train the SVM model? You don't need to know because the implementation is hidden with object-oriented programming. If the implementation changes, you as a user of Scikit-learn might not ever find out. Whether or not you SHOULD understand how SVM works is a different question.

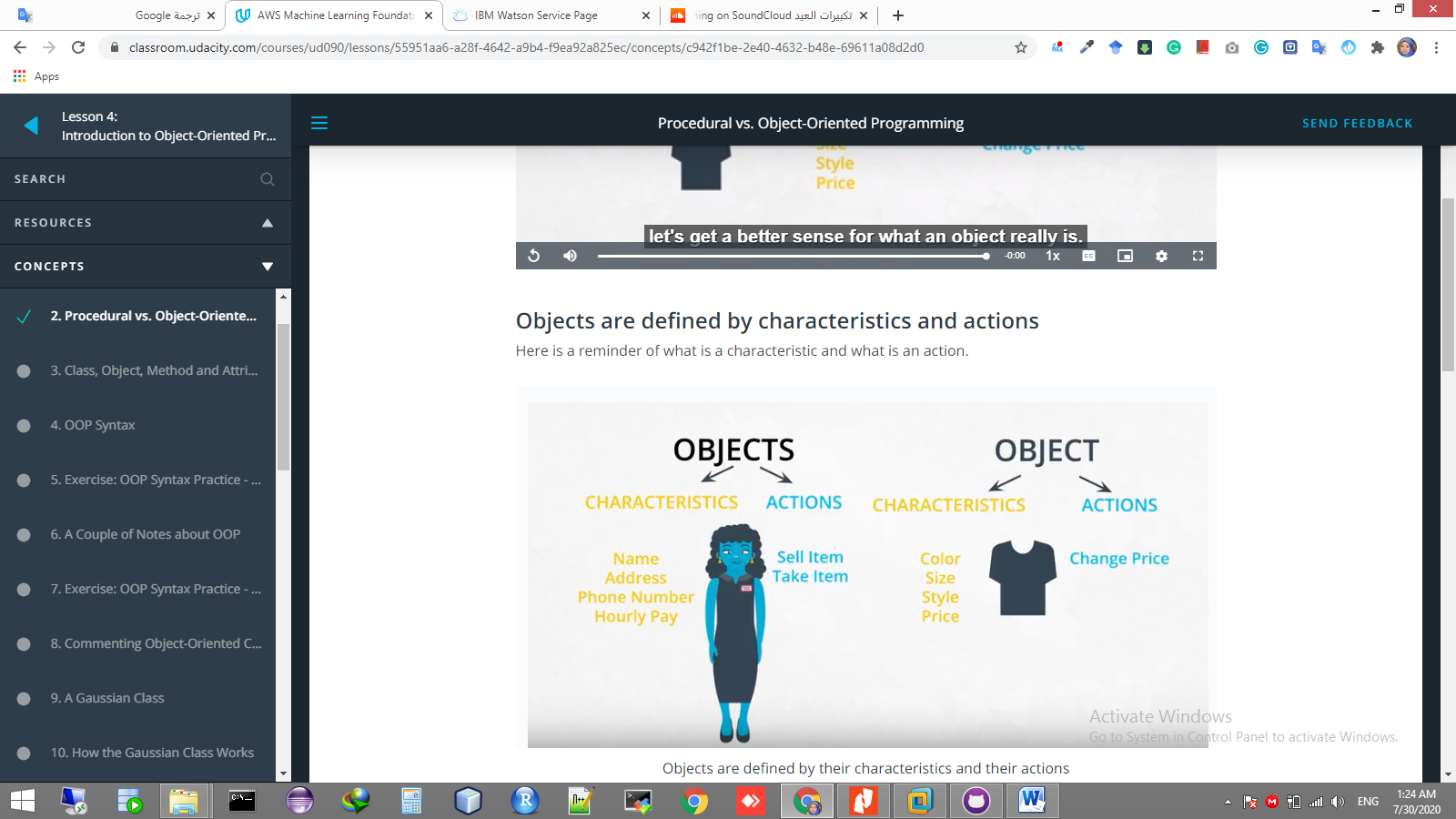
In this lesson, you'll practice the fundamentals of object-oriented programming. By the end of the lesson, you'll have built a Python package using object-oriented programming.

**Lesson Files**

This lesson uses classroom workspaces that contain all of the files and functionality you will need. You can also find the files in the [**data scientist nanodegree term 2 GitHub repo**](https://github.com/udacity/DSND_Term2/tree/master/lessons/ObjectOrientedProgramming).

# Objects are defined by characteristics and actions

Here is a reminder of what is a characteristic and what is an action.



# Characteristics and Actions in English Grammar

Another way to think about characteristics and actions is in terms of English grammar. A characteristic would be a noun. On the other hand, an action would be a verb.

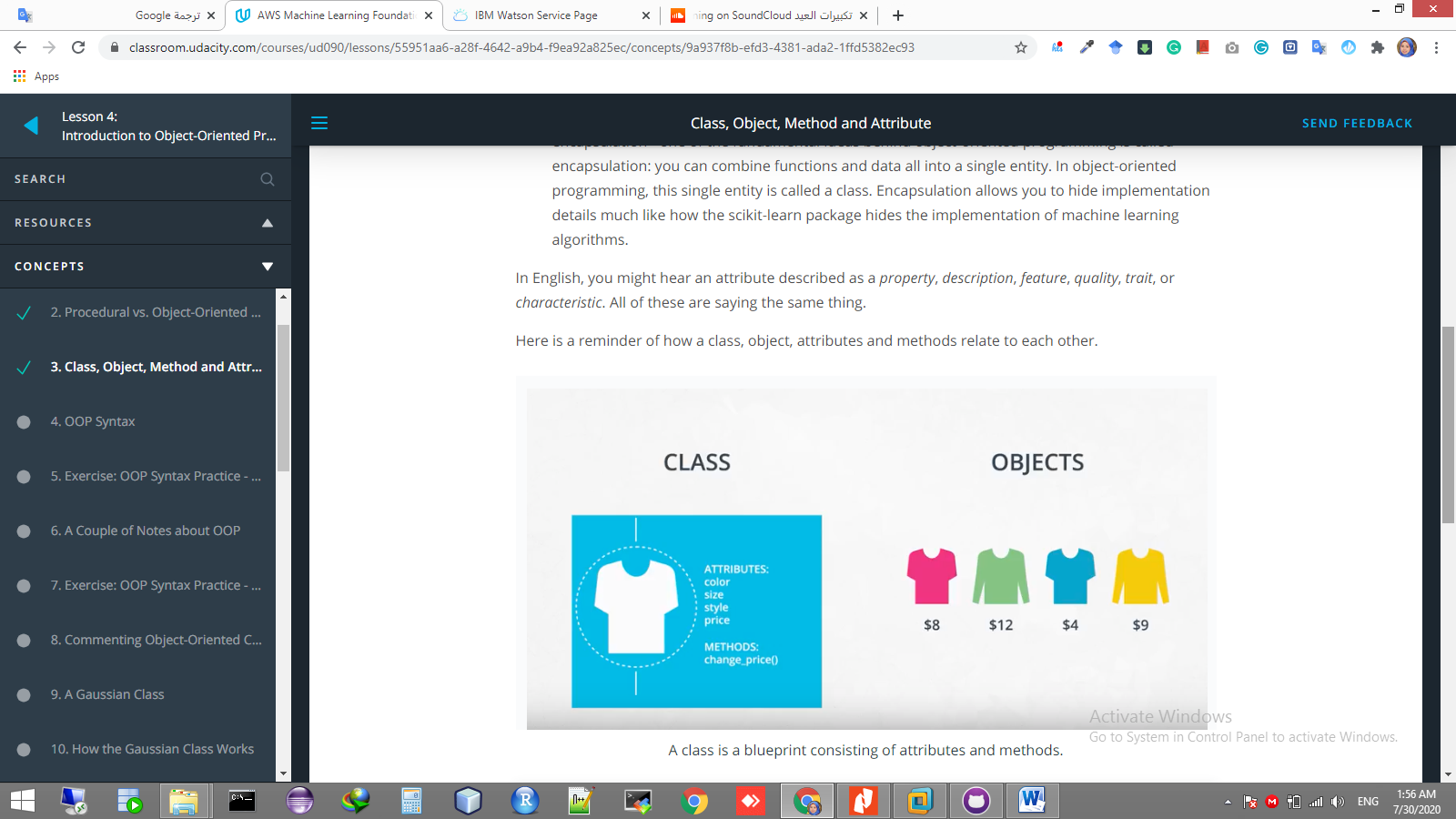
Let's pick something from the real-world: a dog. A few characteristics could be the dog's weight, color, breed, and height. These are all nouns. What actions would a dog take? A dog can bark, run, bite and eat. These are all verbs.

# Object-Oriented Programming (OOP) Vocabulary

* class - a blueprint consisting of methods and attributes
* object - an *instance* of a class. It can help to think of objects as something in the real world like a yellow pencil, a small dog, a blue shirt, etc. However, as you'll see later in the lesson, objects can be more abstract.
* attribute - a descriptor or characteristic. Examples would be color, length, size, etc. These attributes can take on specific values like blue, 3 inches, large, etc.
* method - an action that a class or object could take
* OOP - a commonly used abbreviation for object-oriented programming
* encapsulation - one of the fundamental ideas behind object-oriented programming is called encapsulation: you can combine functions and data all into a single entity. In object-oriented programming, this single entity is called a class. Encapsulation allows you to hide implementation details much like how the scikit-learn package hides the implementation of machine learning algorithms.

In English, you might hear an attribute described as a *property*, *description*, *feature*, *quality*, *trait*, or *characteristic*. All of these are saying the same thing.

Here is a reminder of how a class, object, attributes and methods relate to each other.



A class is a blueprint consisting of attributes and methods.

A function and a method look very similar. They both use the def keyword. They also have inputs and return outputs. The difference is that a method is inside of a class whereas a function is outside of a class.

# What is self?

If you instantiate two objects, how does Python differentiate between these two objects?

shirt\_one = Shirt('red', 'S', 'short-sleeve', 15)

short\_two = Shirt('yellow', 'M', 'long-sleeve', 20)

That's where self comes into play. If you call the change\_price method on shirt\_one, how does Python know to change the price of shirt\_one and not of shirt\_two?

shirt\_one.change\_price(12)

Behind the scenes, Python is calling the change\_price method:

**def** **change\_price**(self, new\_price):

self.price = new\_price

Self tells Python where to look in the computer's memory for the shirt\_one object. And then Python changes the price of the shirt\_one object. When you call the change\_price method, shirt\_one.change\_price(12), self is implicitly passed in.

The word self is just a convention. You could actually use any other name as long as you are consistent; however, you should always use self rather than some other word or else you might confuse people.