**Python Objects and Classes**

An **object** is a special datatype in programming that contains **properties** (key: value pairs) and **methods** (internal functions that can perform operations on the object itself but is not limited to just that). A **class** is a blueprint for objects that allow you to create multiple objects with different values for their properties. Think of them like a factory where you can feed in some variables and get an object at the end. Note that in Python properties are often to as attributes in objects.

You might have thought that dictionaries looked very similar to JavaScript objects, and you’d be right. They look and behave similarly however dictionaries are a lot more limited. For example, methods aren’t possible in dictionaries.

**JavaScript Object Literal**  **Python Dictionary**

**const myObj = { my\_dict = {**

**name: "James", "name": "James",**

**age: 12, "age": 12**

**sayName: function() { }**

**alert( this.name ); *#*** *no methods/functions*

**} *#*** *in dictionaries*

**}**

**myObj["name"]; my\_dict["name"] *#*** *‘James’*

**myObj["age"] = 6; my\_dict["age"] = 6 *#*** *updates age prop*

In JavaScript we could create an object literal, which is a basic object created without a class. Python is more limited however because we need a class to create an object. Even if we want to create just one.

**Classes – Blueprints for Objects**

Use the **class** keyword when creating a class. Like JavaScript, it is customary to capitalize the first letter of the class name to distinguish it as a class. You can add basic propertiesattributes just by creating simple variables inside of it. These attributes have their values fixed to the class when the object is created and are available to the object.

**class HumanClass:**

**species = "mammal"**

If you want to pass in specific values for properties we’ll need to use the **\_\_init\_\_()** function. Take note that **there are 2 underscores in front and behind** the **init** word here. This **\_\_init\_\_()** function is called immediately when the object is made and takes in parameters which you can assign as values to properties. Take note that the first parameter has to be **self** which is Python’s way of referring to the object itself. **self** acts the same as **this** in JavaScript objects and will be used quite a bit.

**self** is not assigned a value unlike the other parameters in the **\_\_init\_\_()** function as it represents the object. It still must be present, however.

**class HumanClass:**

**species = "mammal"**

**def \_\_init\_\_(self, name, age):**

**self.name = name**

**self.age = age**

**self.warm\_blooded = True**

Note that I also added an attribute in the **\_\_init\_\_()** function but assigned it a fixed value for **warm\_blooded**. This is perfectly fine too. This attribute won’t be assigned a custom value on creation. It is like **species** but differs in that it represents the object while **species** represents the class.

**Objects – Creating them from a Class**

Now our class is ready to create a human object with 4 attributes. 2 of these (name and age) we can assign our own values at the time of creation.

**human\_obj = HumanClass( "Bob", 45 )**

**Accessing attributes** in the object is done using the dot notation **human\_obj.name**. Note the square brackets system that is useable in JavaScript does not work here **~~human\_obj["name"]~~**

**print( human\_obj.name ) *#*** *‘Bob’*

**print( human\_obj.age ) *#*** *45*

**print( human\_obj.species ) *#*** *‘mammal’*

**print( human\_obj.warm\_blooded ) *#*** *True*

**Updating** and **deleting properties** in Python objects is very simple.

**human\_obj.name = "Kevin" *#*** *name is now ‘Kevin’*

**human\_obj.species = "Reptile" *#*** *species changed to reptile*

**del human\_obj.age *#*** *age is deleted from object*

**print(** **human\_obj.age ) *#*** AttributeError: '**human\_obj**' object has no attribute 'age'

**Objects object.\_\_dict\_\_**

Every object in Python has an attribute called **\_\_dict\_\_** which returns a dictionary of all the properties and their values from the object. This can be very handy to check and see what is in an object quickly. Methods are not contained in this however.

**print( human\_obj.\_\_dict\_\_ )**

***#*** *{‘name’: ‘Kevin’, ‘species’: ‘Reptile’, ‘warm\_blooded’: True}*

**Objects – Methods (functions belonging to an object)**

Methods are functions inside of objects which can perform on the object itself. They look the exact same as a regular function and are defined as such in the class. The big difference however is that they automatically have a first parameter called **self** which allows it to interact with the object just like in the **\_\_init\_\_()** function. Even if you don’t want the function to use **self**, ***all methods in an object have their first parameter assigned as* self**.

**class HumanClass:**

**def \_\_init\_\_( self, name, age ):**

**self.name = name**

**self.age = age**

**def my\_method( self ): *#*** *method referencing object*

**print( "My name is " + self.name )**

**def add\_one( self, num ): *#*** *method with own parameter*

**print( num + 1 )**

Let’s create a new object from the above class.

**human\_obj = HumanClass( "Laura", 15 ) *#*** *Create object from class*

When calling methods don’t forget to use the object’s name and dot notation followed by the method name.

**human\_obj.my\_method() *#*** *prints “My name is Laura”*

**human\_obj.add\_one ( 10 ) *#*** *prints 11, 10 is passed in for num parameter*

**human\_obj.add\_one ( 15 ) *#*** *prints 16, 15 is passed in for num parameter*

**NOTE:** Be careful when passing parameters into an object method. Remember that all object methods must have **self** as the first parameter. If you want the method to take in extra parameters, pass them in after **self**. Let’s take a look at the **add\_one()** method above once again.

**human\_obj.add\_one ( 10 ) *#*** *calling method* ***10*** *is passed to* ***num***

**{ …**

**def add\_one( self, num ): *# self****, the first parameter isn’t* ***10***

**print( num + 1 ) *#*** *this works*

**… }**

If we forget to add self when creating the method in the class we’ll get an error because Python grabs the first parameter and tries to use it as self. The below code will give an error indicating the number of parameters/arguments used is incorrect.

**human\_obj.add\_one ( 10 ) *#*** *causes an error*

**{ …**

**def add\_one( \_ num ): *# num*** *assumes the* ***self*** *value*

**print( num + 1 ) *#*** *ERROR 2 parameters when only should be 1*

**… }**