Classes and Objects

Tuesday, December 21, 2021 3:42 PM

CLASSES

Classes are a widely used tool in any high level programming language.

A Class is thought of as a template for creating objects and functions. Compared other languages, creating classes in Python is simpler.

To create a class in Python we simply type class in undercase and followed by the class name and a colon, note the class name should follow pascalcase meaning it starts with a capital letter according to PEP 8 standards. In the following example, we will create a class called User which will simply do nothing and thus we type the word pass.

This is because classes need at least one line when defined:

```
In [1]: class User:
            pass
```

With the class defined, we can add different users to it, we do this by typing out an instance of the user class, in this case user1 followed by the assignment operator = ,the name of the class followed by parenthesis, much like how we call a method.

User1 is also called an object, see example:

```
In [3]: class User:
            pass

In [4]: user1 = User()
```

We can then go further and attach data to this object. The data attached to an object is known as a field. Field names should follow snake case which means it should be under case. Additionally if there is more than one word for the field name, each word should be separated by an underscore. This is done by first typing the name of the object followed by a full stop (period), followed by the variable name, the assignment operator = and the value, see example:

```
In [ ]: class User:
            pass

        user1 = User()
        user1.first_name = "Joe"
        user1.sur_name = "Saund"
```

To see if this data exists we can simply use a `print()` method. In the parenthesis we put the object name followed by a period and the field name, see the example:

```
In [6]: class User:
            pass

        user1 = User()
        user1.first_name = "Joe"
        user1.sur_name = "Saund"

In [7]: print(user1.first_name)
Out[9]: Joe
```

Fields however in Python do not need to be assigned to an object and can be made to be stand alone. We can do this by simply typing the field name followed by a period then the value name. These values can be seen when printing the field names without an object name. These values are kept sperate to user1 because user1 already has values assigned to its fields, see example:

```
In [11]: first_name = "Jason"
         sur_name = "Statham"
```

```
print(first_name, sur_name)
```

```
Out[12]: ('Jason', 'Statham')
```

Additional fields can be attached to objects and can be of different data types. We can create a second user called user2 and assign two different fields to both it and user1 which neither object has. For example, user1 has an age field while user2 has a mobile_phone field:

```
In [14]: user2 = User()
         user2.first_name = "Rukia"
         user2.sur_name = "Kuchiki"

In [15]: user1.age = 33
         user2.phone_number = 1234

In [16]: print(user1.age)

Out[12]: 33

In [17]: print(user2.age)

Out[12]: ---------------------------------------------------------------------------
         AttributeError                            Traceback (most recent call last)
         ~\AppData\Local\Temp/ipykernel_4880/984494274.py in <module>
         ----> 1 print(user2.age)

         AttributeError: 'User' object has no attribute 'age'
```

Both of the new fields are integers, however when we try to print the age for user2 which was not assigned, the Python interpreter returns an attribute error. It is key to make sure each user is assigned each field.

METHODS

Adding methods we can make a class more efficient and effective than say another data structure such as a dictionary.

A function inside a class is called a method. The first method is called an initialization method or an init method. This init method is also known as a constructor.

To write a init method we use def followed by init, init which has two underscores before and after it `__init__`, then followed by an open parenthesis. This method gets called each time a new instance of the class is called, see example:

```
In [ ]: class User:
            def __init__(
            )
```

The first argument to this method is self which references the new object that is being created. Other arguments can also be added to the parenthesis, in this case full name and date of birth. After which a colon goes at the end.

```
In [ ]: class User:
            def __init__(self, full_name, d_o_b):
```

These arguments can be stored to fields in the object. This is done by typing self, followed by a period, the field name and then assigning it to a value. In this example, full name is stored in a field called fullname and d_o_b will be stored in a field called birthday:

```
In [ ]: class User:
            def __init__(self, full_name, d_o_b):
                self.fullname = full_name
                self.birthday = d_o_b
```

A user can now be created to utilize the init method. This difference here is that because we are using the init method in which we provided two arguments, it expects two values which have to be typed into the parenthesis which in this example is the name and the date of birth, see example:

```
In [20]: class User:
             def __init__(self, full_name, d_o_b):
                 self.fullname = full_name
                 self.birthday = d_o_b

         user = User("Joseph Saund", "19880816")

         print(user.fullname, user.birthday)
```

```
Joseph Saund 19880816
```

We can print both fields to see if the data exists. Note that Python sees date in a year/month/day format, so is typed as yyyymmdd.

Another feature can be added to the init method which can split both the first and last name form the full_name field. This is done using the `split()` method. Within the methods parenthesis we can put a space `" "`. This cuts a string whenever it encounters a space. The pieces of the full name will be stored in an array.

A new field called name needs to be created followed by an assignment operator `=` the fullname field, a period and an argument in the parenthesis. Next we type self, period the firstname field, an assignment operator `=` then open and close square brackets with an index of 0 due to it being an array.

Another field for last name needs to be created in the same way, except the index is –1 due to it appearing last in the array, note that it is important to include the use of self. Otherwise am attribute error will be returned by the Python interpertor:

Finally we can print the new fields for the separated first and surname to confirm that it works.

In [23]:
```
class User:
    def __init__(self, full_name, d_o_b):
        self.fullname = full_name
        self.birthday = d_o_b

        name = full_name.split(" ")
        self.first_name = name[0]
        self.sur_name = name[-1]

user = User("Joseph Saund", "19880816")

print(user.first_name, user.sur_name, user.birthday)
```

```
Joseph Saund 19880816
```

As you can see it works, we can see both the first and last names by calling them as sperate fields in the `print()` method.

With the addition of a doc string, the `help()` function can also be used to enhance a class further. This can be called by using the `help()` method with the class name placed within the parenthesis. The doc string is displayed as a summary as well as the arguments that are expected in the class.

It is highly recommended to include a doc string into classes and init methods.