Lecture 4 Notes

February 6, 2020

Lecture 4: Functions and Object Oriented Programming

Today we will cover two of the most important concepts in programming which are functions and objects.

Functions

Functions are named sections of a program that performs a specific task. In this sense, a function is a type of procedure or routine. This routine performs a tasks and in some cases returns a value. This is how you create a function in python:

```
def hello():
    print("Hello buddy!")
```

The keyword `def` signifies the definition of a function, then follows the name of the function, in this case `hello`. Then the parenthesis, which we will discuss after too, and at the end : which always signifies the end of a statement in Python.

To call the function we created we simply use:

```
hello()
```

The parenthesis contain the **arguments** which are parsed in the function. Arguments are variables which you want the function to work with. A simple example of this is the function `print` which we have covered already:

```
print("Hello buddy!")
```

`print` is the name of the function, and the string `"Hello Buddy"` is the argument parsed into the function.

Here is an example of how arguments could be used in your own functions:

```
def addition(x, y):
    print(x+y)

addition(1, 2)
```

Here we are parsing the two integers `1` and `2`, the function `addition` is taking them and printing the sum. Now we will cover the return of a function.

Return

A return is a value that a function returns to the calling script or function when it completes its task. You will be using this a lot if you keep programming. Let's look at and example:

```
def operation(x, y):
    result = (x+y)*2
    return result

r = opeation(3, 6)
print(r)
```

We are parsing the integers `3` and `6` into the function `operation`, the variable `result` which is defined within the function, is returned. This means that the variable `r` will have the value of whatever is returned from `operation`

There is also the possibility to nest functions, which is a very neat way to clean up your code. Don't overdo it or it will look like crap though! :)

```
def operation(x, y):
    result = (x+y)*2
    return result
```

```
print(opeation(3, 6))
```

It is important to know that once a function returns, nothing else within that function will execute. (*yield*)

When would you use this?

Functions are great to split up your program into parts and run the same operations multiple times without having to re-write it.

Objects

Another concept implemented in programming languages like python, is Object Oriented Programming (OOP). Objects can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures which we can call functions or methods.

```
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

    def bark(self):
        print("Bau Bau")

    def sit(self):
        print(self.name+" sits")

    def talk(self, message):
        print(self.name+" says: "+message)

    def __str__(self):
        return self.name+":"+self.breed+":"+str(self.age)
```

We define the object by using the syntax `class` followed by the name of the object, in this case `Dog` and then the `:` since this is a statement.

The Constructor

The constructor of an object is a subroutine which creates the object. In python we define the constructor of the object using the syntax `def __init__(self):`. **self** is a special keyword which represents the object itself. It is used to set fields (attributes) of the object. The keyword `self` must be parsed to the methods of the object which require access to the object's fields.

In our examples we are setting the fields `self.name`, `self.age` and `self.breed` which will be avaiable to all the methods using the keyword `self`.

Other than `self` we can parse other arguments in the constructor, usually they are used to set the fields of the object.

Methods

Methods are just like functions, and you define them inside the object. These methods can take the arguments `self` but also any other argument you want your method to use. An example using the `Dog` object is:

```python
def talk(self, message):
    print(self.name+" says: "+message)
```

We are accessing the field `name` and printing it along with the message parsed in the method.

Using the object

Now we have defined an object, it's time to use it.

```
d = Dog("NotAPlanet", 6, "French Bulldog")
print(d.name)
print(d.breed)
print(d.age)
d.bark()
d.talk("Hello, I'm not supposed to talk")
d.sit()
```

As you can see we simply create our object by calling its name `Dog` with brackets and inside the arguments to parse to the constructor. The variable `d` will now be an object of "type" `Dog`. We can access the object's fields and methods just by calling the object name `.` field/method.

Representations

Representations are really fun. They allow you to assign a type representation to an object. In our case it is the `string` type representation, which is defined using the name `__str__` as the name of the method. We can get the "stringfied-dog" using:

```
print(str(d))
```

We can use this to create more complex programs, for example if we had an array of dogs, we could set the `int` type representation to be their age using `__int__`. This would allow us to for example sort the dogs by their age.

Recap

So today we covered a lot of stuff so here is a little summary:

- Functions:
  - Arguments: variables given to the function to work with
  - Return: values returned to the function call when all operations are finished
  - E.g.
    ```
    def operation(x, y):
        result = (x+y)*2
        return result
    ```

```
    r = opeation(3, 6)
    print(r)
```

That' all there is to functions.

- Objects:
  - Constructor: creates the object and initially sets the object's fields
  - Methods: functions exclusive to the object
  - Self: the object itself, which is used within to call fields or methods
  - Representations: the type representation of an object
  - E.g.

```
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

    def bark(self):
        print("Bau Bau")

    def sit(self):
        print(self.name+" sits")

    def talk(self, message):
        print(self.name+" says: "+message)

    def __str__(self):
        return self.name+":"+self.breed+":"+str(self.age)
```

Challenges

1) Create a Person object which has at least the following fields:
    - name
    - age (int)

- job
- motto

Then have methods which allow the Person object to introduce himself and add a string represenation of the object which returns the motto.

2) Create an array of Person object, then have a function which takes the array as an argument and returns the oldest Person in the array. (You will have to use loops to find it out)