04-Functions and Using Modules

Unknown Author

August 20, 2013

1 Functions

Functions are the basic building blocks that we use to store chunks of code we'll want to use again later. The details are pretty simple, but this is one of those ideas where it's good to get lots of practice!

```
In [1]: def simple_function(x):
            print x + 1

        simple_function(2)
```

```
3
```

Note that our function might not work for everything we pass in (you should get an error on this one):

```
In [2]: simple_function('2')
```

```
        ---------------------------------------------------------------
-------------
    TypeError                                   Traceback (most recent
call last)



        <ipython-input-2-d20a2ba40f15> in <module>()
    ----> 1 simple_function('2')




        <ipython-input-1-f7f6e7cb5905> in simple_function(x)
          1 def simple_function(x):
    ----> 2     print x + 1
          3
          4 simple_function(2)




        TypeError: cannot concatenate 'str' and 'int' objects
```

Functions can take any number of arguments

```
In [3]: def less_simple(a, b, c):
            print a + b + c

        less_simple(1, 2, 3)
```

```
6
```

And now we can pass in strings (as long as they're all strings):

```
In [4]: less_simple('These ', 'should ', 'concatenate')
```

```
These should concatenate
```

You can also use named arguments instead of positional ones. Note how the printed order is still "abc" even though we pass in "acb."

```
In [5]: less_simple(a='first', c=' third', b=' second')
```

```
first second third
```

1.1 Variables are created and destroyed in a function call

When you execute a previously-defined function, like `simple_function(3)`, we say that you "called" the function. When you call a function, a temporary workspace is set up that will be destroyed when the function returns by:

1. getting to the end, or
2. explicity by a `return` statement

In this temporary environment, the variables in the parameter list (in parentheses in the definition) are set to the values passed in. For example, in `simple_function(3)`, `x` gets set to `3`. Afterwards, you can't access these variables!

```
In [6]: # Check out the def of simple_function above - we're setting x to 3!
        simple_function(3)
        # x is no longer defined because simple_function returned (i.e., finished)!
        x
```

```
        ---------------------------------------------------------------
-------------
    NameError                                 Traceback (most recent
call last)



        <ipython-input-6-b7f603e5e74f> in <module>()
          2 simple_function(3)
          3 # x is no longer defined because simple_function returned
(i.e., finished)!
    ----> 4 x
```

```
NameError: name 'x' is not defined
```

```
4
```

Things can get confusing when you use the same names for variables both inside and outside a function. Check out this example:

In [7]:
```
night = 'night'
day = 'day'

# If you were just reading through, it would be easy to think
# that 'night' in this function corresponds to 'night' above!
def confused_by_names(night, day):
    print 'night is', night
    print 'day is', day


confused_by_names(day, night)
```

```
night is day
day is night
```

So, to avoid confusion, *use different variable names in every context!*

In general, Python has very rich options for defining and calling functions. But we're already getting ahead of Codecademy, so we'll stop here for now.

2 Modules

Python has an extensive standard library that is always included with the python interpreter. You can read about it here:

http://docs.python.org/2/library/

It can seem overwhelming! I've been using it for years and still don't know everything in there!

Another very useful module is pandas. It provides a DataFrame structure that's a lot like a spreadsheet:

In [8]:
```
import pandas

df = pandas.DataFrame({'Day': [1, 2, 3, 4], 'Score': [88, 90, 76, 43]})
```

The IPython notebook will print these very nicely:

In [9]:
```
df
```

Out [9]:
```
   Day  Score
0    1     88
1    2     90
2    3     76
3    4     43
```

And we can select columns or rows (or both):

```
In [10]: df.Score # This only works if the column is a valid python variable name!
```

```
Out [10]:
0    88
1    90
2    76
3    43
Name: Score, dtype: int64
```

```
In [11]: df.iloc[1]
```

```
Out [11]:
Day       2
Score    90
Name: 1, dtype: int64
```

```
In [12]: df.Score[1]
```

```
Out [12]:
90
```

Pandas is very powerful. Even this one library will take time to master. You can read more about it here:

http://pandas.pydata.org/