GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 2: ETL AND DATA STORAGE

AGENDA

I. INTRO TO PYTHON

II. PYTHON HANDOUT WALKTHROUGH

III. ETL HANDOUT WALKTHROUGH

I. INTRO TO PYTHON

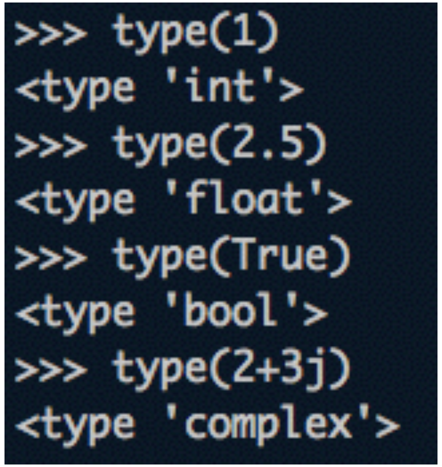
SETTING UP VARIABLES

- Python shell is just a complex calculator:
  - 10 * 15

  - x = 5
  - x #prints 5
  - x^2 #prints 25

The most basic data structure is the **None** *type. This is the equivalent of NULL in other languages.*

There are four basic numeric types: **int, float, bool, complex, string**



```
>>> type(1)
<type 'int'>
>>> type(2.5)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type(2+3j)
<type 'complex'>
```

DATA TYPES

- Lists:
  - l = [1, 2, 3]
  - l = [‘happy’, ‘sad’, ‘indifferent’]

- Dictionaries (Maps):
  - Key-Value datastructure
  - d = { ‘first_name’ : ‘Arun’, ‘last_name’: ‘Ahuja’}

IF/ELSE STATEMENTS

- If/Else statements allow us to take different paths through depending on some condition:
- x = 5
- if x > 4:
    - print “This number was less than 4”

LOOPING

- Looping allows us to pass through some set of values and perform an operation on each

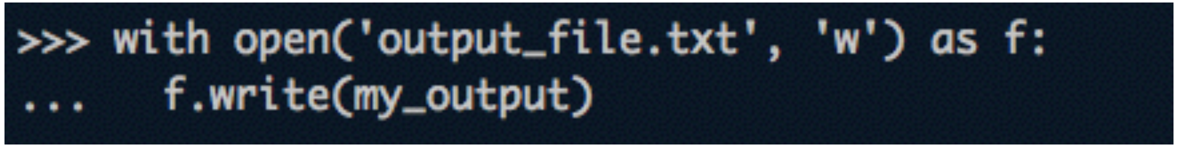
- l = [“happy”, “sad”, “don’t care”]
- for x in l:
    - print x
    - if x == ‘happy’:

FUNCTIONS

- Functions allow us to save some piece of functionality to reuse later

- def func(x):
    - if x > 4:
        - print “This number is less than 4

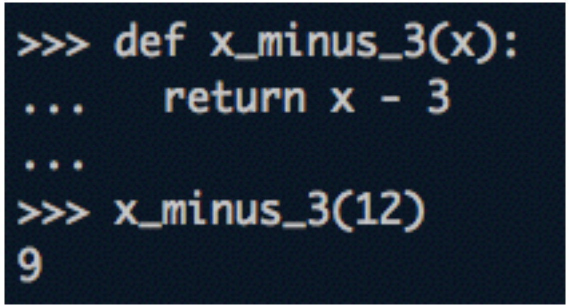
Our final example of a data type is the Python **file object***. This represents an open connection to a file (eg) on your laptop.*



```
>>> with open('output_file.txt', 'w') as f:
...     f.write(my_output)
```

These are particularly easy to use in Python, especially using the with statement context manager, which automatically closes the file handle when it goes out of scope.

Python allows you to define custom **functions** *as you would expect:*

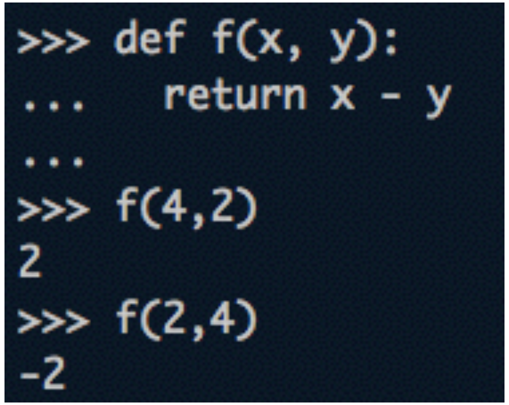


```
>>> def x_minus_3(x):
...     return x - 3
...
>>> x_minus_3(12)
9
```

Functions can optionally return a value with a **return statement** *(as this example does).*

Functions can take a number of **arguments** *as inputs, and these arguments can be specified in two ways:*

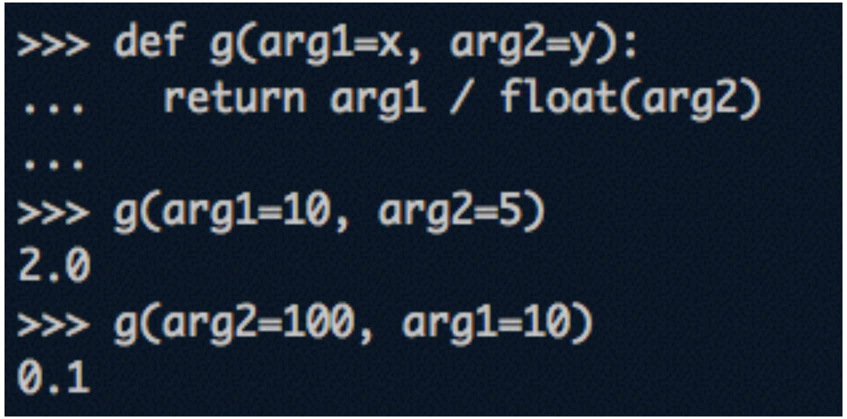
As **positional arguments***:*



```
>>> def f(x, y):
...     return x - y
...
>>> f(4,2)
2
>>> f(2,4)
-2
```

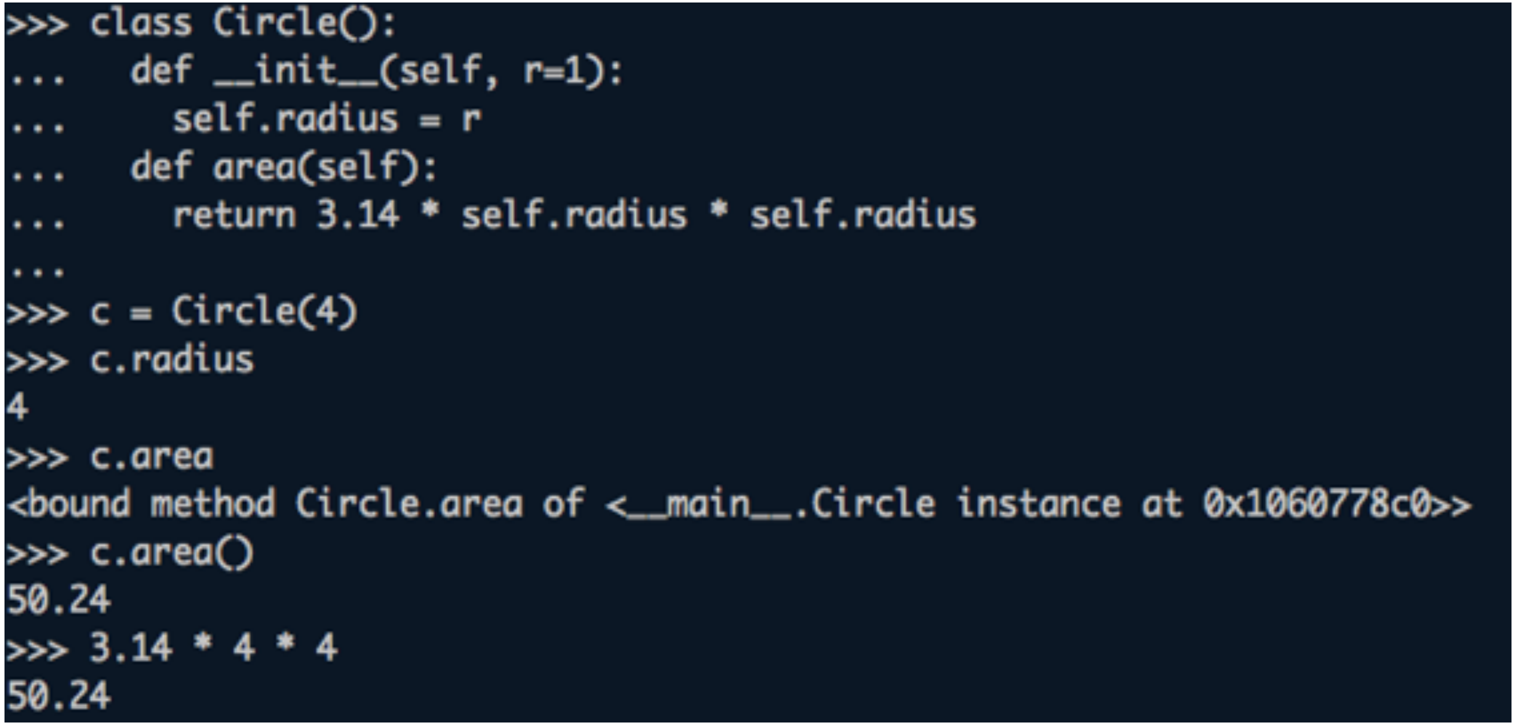
Functions can take a number of **arguments** *as inputs, and these arguments can be specified in two ways:*

Or as **keyword arguments***:*



```
>>> def g(arg1=x, arg2=y):
...     return arg1 / float(arg2)
...
>>> g(arg1=10, arg2=5)
2.0
>>> g(arg2=100, arg1=10)
0.1
```

Python supports **classes** *with member attributes and functions:*



```
>>> class Circle():
...   def __init__(self, r=1):
...     self.radius = r
...   def area(self):
...     return 3.14 * self.radius * self.radius
...
>>> c = Circle(4)
>>> c.radius
4
>>> c.area
<bound method Circle.area of <__main__.Circle instance at 0x1060778c0>>
>>> c.area()
50.24
>>> 3.14 * 4 * 4
50.24
```