EFM Fall 2014, Week 2: Introduction to Python

Jason Phang, Allan Zhang

October 16, 2014

Table of Contents

Table of Contents

Announcements

Diagnostic test results!

- Average number of questions completed: 4.1
- We'll see how we're doing at the end of the quarter
- Ask me for solutions!

Table of Contents

Python

Stuff we'll cover today

- Operations
- Lists
- Strings
- Basic control structures
- A little bit of File I/O

Running Python

Another way of running Python code saved in a file:
Save the following to a file:

my_python_code.py

```
for i in range(n):
  print "Print this " + str(n) + " times!"
```

And the run the following"

Commands

```
$ ipython

In [#]: %run file_name.py
```

Note:
$ indicates running in shell

In [#] indicates running in IPython

Python Basics

Some very basic Python:

Assignment of values

```
x = 10
print x
```

Basic mathematical operations

```
print x - 20
print x * 2
print x / 4
print x % 4
print x ** 3
```

Output (Collapsed)

```
-10, 20, 2, 2, 1000
```

Note that x here is an **Integer**, i.e. a whole number.

Python Basics

We can define y as a **float** or "floating-point number", i.e. a real number, with decimal places.

Assignment of values (Floats)

```
y = 10.0
print y + 17
print y - 20
print y * 2
print y / 4
print y % 4
print y**0.5
```

Which line causes an error?

Lists

Lists

- A **list** is an ordered sequence of elements.
- It is defined by a pair of square brackets [ ]

Lists (1)

```
list_1 = [1,2,3]
print list_1
```

Output

```
[1,2,3]
```

Lists

- We can **append** (add items) to lists

Lists (2)

```
list_1 = [1,2,3]
list_1.append(4)
print list_1
```

Output

```
[1,2,3,4]
```

Lists

- We can also add lists together

Lists (3)

```
list_1 = [1,2,3]
list_2 = [10,20,30]
print list_1 + list_2
```

Output

```
[1,2,3,10,20,30]
```

Lists

- We can also re-assign elements in a list

Lists (4)

```
list_1 = [1,2,3,4,5]
list_1[0] = 100
list_1[2:] = [0,0,0]
```

Output

```
[100, 2, 0, 0, 0]
```

Lists

- Other useful functions:

Lists (5)

```
list_1 = [1,2,3]
print len(list_1)
print sum(list_1)
```

Output

```
3
6
```

Indexing

Indexing

- To get items from our lists, we look up a specific **index**. We put our index in square brackets
- Note that indexing **starts from 0**
- We can also index starting from the back, with **-1**

Indexing (1)

```
list_1 = [1,2,3,4,5]
print list_1[0]
print list_1[1] + list_1[2]
print list_1[-2]
```

Output

```
1
5
4
```

Indexing

Indexing

- Besides indexing individual elements, we can also index (consecutive) segments of a list.
- We do so by providing starting and ending (exclusive) indices, separated by a colon **:**
- If you leave out either of the indices, it will default to the start/end of the list respectively

Indexing (2)

```
list_1 = [1,2,3,4,5,6,7,8,9,10]
print list_1[2:4]
print list_1[7:]
print list_1[:-2]
```

Output

```
[3, 4]
[8, 9, 10]
```

Indexing

- We can also index in intervals, by providing a third number after another colon.

Indexing (3)

```
list_1 = [1,2,3,4,5,6,7,8,9,10]
print list_1[::3]
```

Output

```
[1,4,7,10]
```

Range

Range

- We can quickly generate lists of numbers using the **range** function.
- This is often used in loops and iteration.

Range (1)

```
print range(10)
```

Output

```
[0,1,2,3,4,5,6,7,8,9]
```

- We can also supply a starting number.

Range (2)

```
print range(2,5)
```

Output

```
[2,3,4]
```

Strings

Strings

- Strings are the data structure for storing texts.
- They can be declared with single or double quotation marks.

Strings

```
s = "Hi bob"
print s
```

Output

```
Hi bob
```

Strings

Strings

- They share a lot of similarities with lists. (They are both "iterables".)
- In particular, you can index them.
- You can also add them (Can you do this with lists?)

Strings

```
s = "Hi bob"
print s[0:4]
print s[0:3] + "Sarah"
```

Output

```
Hi b
Hi Sarah
```

Strings

Strings

- Two particularly useful functions: **split** and **join**.

Strings

```
s = "The cat is fat"
list_1 = s.split(" ")
print list_1

s2 = '-'.join(list_1)
print s2
```

Output

```
['The', 'cat', 'is', 'fat']
The_cat_is_fat
```

- Notice the difference in syntax!

Dictionaries

Dictionaries

- Dictionaries are un-ordered, labeled sets of elements
- You can think of it as an collection of items that can be indexed by anything
- They are defined using curly brackets { }

Dictionaries (1)

```
heights = {}
heights["bob"] = 176
heights["ann"] = 208
print heights
print heights["ann"]
```

Output

```
{'bob': 176, 'ann': 208}
208
```

Dictionaries

- Note that the indices (keys) and the elements (values) can be of any time.

Dictionaries (2)

```
random_dict = {}
random_dict["bob"] = 1
random_dict[2] = "the number two"
print random_dict
```

Output

```
{2: 'the number two', 'bob': 1}
```

Dictionaries

Strings

- You can also pre-allocated your dictionary when you create it.
- Key-value pairs are denoted by colons, separated by commas

Dictionaries (3)

```
dict_1 = {1:"one", 2:"two"}
print dict_1[1]
```

Output

```
one
```

Dictionaries

Strings

- A few other useful functions

Dictionaries (4)

```
print random_dict.keys()
print len(random_dict)
```

Output

```
['bob', '2']
2
```

Control Structures

If

- **If** is a conditional.
- The identation (spacing) here is important! Python is "white-space" sensitive.

If (1)

```
n = 10
if n < 5:
    print "Too small!"
```

Output

Control Structures

If

- Sometimes **if**s come with **else** or even **elif** clauses. Both are optional.

If (2)

```
n = 10
if n < 5:
    print "Too small!"
elif n > 15:
    print "Too large!"
else:
    print "Just right :)"
```

Output

```
Just right :)
```

Control Structures

while

- **while** is a loop, that allows you to perform an action repeatedly.
- Notice the identation again.

While

```
n = 0
while n < 3:
    print n,
    n = n + 1
```

Output

```
0 1 2
```

- Be very careful of never-ending loops! (If ever in trouble, try to hit Control-C)

Control Structures

for

- **for** is another loop structure, that allows you to *iterate* over a list (or iterable) of elements

For

```
for n in range(10):
    print n,
```

Output

```
0 1 2 3 4 5 6 7 8 9
```

- No risk of never-ending loops here!

Control Structures

for

- When you iterate over a dictionary using a **for** loop, it iterates over the keys!

For

```
dict_1 = {1:100, 2:200}
for key in dict_1:
    print key
```

Output

```
1
2
```

Control Structures

for vs **while**

- When do we use each one?
- Generally, use **for** if you have a fixed list or iterable to go over, or you want to do something specific for a fixed number of times
- Use **while** if you want to keep doing something (you might not know how many times beforehand) until a condition is attained.

Example

- There is something called the Collatz Conjecture, whereby given a starting n you divide by 2 if n is even, or multiply by 3 and add 1 if n is odd.
- The conjecture states that if you keep going, no matter that n you start with, you will always end at 1.
- The proof of the conjecture is still an open problem, but it's a good example of a case where you don't know how many times you need to loop before you hit your condition.

Control Structures

Collatz Conjecture

```
n = 10    ## Replace this number
while n != 1:
    if n%2 == 0:
        n = n/2
    else:
        n = 3*n + 1
print n,
```

Output

```
5 16 8 4 2 1
```

Comments

- Comments are non-program pieces of text you add to your code to explain things or make your code more readable
- In Python, you can use # for single-line comments. All code after a # on the line is not run.

For

```
## Second half of the next line isn't run
x = 1  # + 2
print x
```

Output

```
1
```

Table of Contents

Reading Data

How do we get data into your Python code?

- Key in manually
- Data files (Good)
  - Text-based (e.g. CSV, JSON, XML)
  - Other formats (Images, binary formats)
- Raw files (Bad)
  - E.g. Web-scraping, text-processing
- Databases
  - SQL (MySQL, SQLite), NoSQL, etc
- APIs (Application Programming Interface)

Opening a text file

- We can load a text file into a giant string!
- First, create your own text file, and put it in the same folder as your code

For

```
## Open the file, as a file object in "read" (r) mode
raw_file = open("my_text","r")

## Read the file into a string
file_string = raw_file.read()

## Close the file
raw_file.close()

print file_string
```

Writing to a text file

- We can also write to a file in a very similar way!

For

```
## Open the file, as a file object, in "read/write" (rw) mode
raw_file = open("my_text","rw")

## Read the file into a string
file_string = raw_file.read()

## Add something on the end
file_string += "\n New line!"

## Write the file using the file object
raw_file.write(file_string)

## Close the file
raw_file.close()
```

Appending to a text file

- We can directly add to a file, without reading from it

For

```
## Open the file, as a file object, in "append" (a) mode
raw_file = open("my_text","a")

## New line to add
new_text = "\n New line!"

## Write to the file using the file object
raw_file.write(new_text)

## Close the file
raw_file.close()
```

Text-based Formats

For data, we normally don't need to manually read and parse the files (e.g. splitting by lines, etc)
Instead, there are a couple common formats

- CSV (Comma-separated values)
  - Lines of data, separated by commas for columns
  - Use **csv** library
- JSON (JavaScript Object Notation)
  - JavaScript associative arrays (dictionaries)
  - Use **json** library
- XML (eXtensible Markup Language
  - Looks like HTML
  - Use **beautifulsoup** or something similar

CSV Example

- Go to: https://www.quandl.com/FRED/GDP-Gross-Domestic-Product-1-Decimal
- Click "Download" and select CSV
- This downloads a .csv file to your computer

CSV Example

CSV Example

```
## Import a library called "CSV"
import csv

## Open file and read into the raw text
raw_text = open('FRED-UNRATE.csv', 'r')

## Create our two lists
dates = []
gdp = []

## Create a reader object using the CSV library. Skip the first line.
reader = csv.reader(raw_text, delimiter=',')
reader.next()

## Iterate over each line in the CSV file using the reader
for line in reader:
    ## Save the date
    dates.append(line[0])

    ## Save the GDP value, which we first convert to a floating-point number (real number)
    gdp.append(float(line[1]))
```

Bye!

To-do!

- New assignment for the week!
- Again, you shouldn't need to spend too long on it. If you're stuck, just shoot me/Allan an email!

Next Week

- We got through a lot of the dry stuff this time. Look forward to:
  - Functions
  - Statistics
  - Plotting
  - Using IPython Notebook and never looking at a terminal again (except when you really need to)