ECON 370 Quantitative Economics with Python

Lecture 5: Python Fundamentals (Part 3)

Zhen Huo [zh335@nyu.edu]
Matthew McKay [mm8390@nyu.edu]
Thomas Sargent [thomas.sargent@nyu.edu]

Spring 2016

Agenda

Python Programming Fundamentals

1. Review + Questions
2. Syntax
    - Whitespace
    - Line Continuation
    - Commenting
3. Mutable and Immutable Objects
4. String Formatting
5. Conditional Logic
6. Iteration
7. List Comprehensions
8. Functions
9. Recursion
10. Exceptions

Review and Questions

Numerics (integers, floats), Strings, Lists, Tuples, Dictionaries

Questions?

Using Python

`python` Base python interpreter

`ipython` More powerful REPL, Jupyter Kernel, high performance tools for parallel computing ...

`Jupyter` A web-based environment that interacts with an IPython kernel.

These **all** come with Anaconda

Python Fundamentals

See **intro-to-python-two.ipynb**

Syntax - Whitespace

Python uses whitespace as a **delimiter** for code blocks

```
for␣i␣in␣range(0,4,2):
␣␣␣␣print(”i␣=␣%s”%i)␣␣␣#This␣code␣belongs␣to␣first␣loop
␣␣␣␣for␣j␣in␣range(2):
␣␣␣␣␣␣␣␣print(”j␣=␣%s”%j)␣␣␣␣␣␣␣␣␣␣#This␣code␣belongs␣to␣second␣l
␣␣␣␣␣␣␣␣print(”i+j␣=␣%s”%(i+j))
```

and is therefore very important.

Syntax - Whitespace

Whitespace is however ignored in parentheses, brackets, and simple expressions

```
>>>m2 = [[1,2,3],
          [4,5,6],
          [7,8,9]]
>>>m2
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Syntax - Line Continuation

```
>>> a = 1 + 2 + 3 + 4 + 5 + 6 + 7
    + 8 + 9 + 10 + 11 + 12 + 13
  File ''<ipython-input-73-24d78ad3af91>'', line 3
    + 8 + 9 + 10 + 11 + 12 + 13
    ^
IndentationError: unexpected indent
```

A line can continue using a **backslash**

```
>>> a = 1 + 2 + 3 + 4 + 5 + 6 + 7 \
    + 8 + 9 + 10 + 11 + 12 + 13
>>> a
91
```

Syntax - Comments

Line Comments

```
# This is a Line Comment, Anything written here is ignored
a = 2
print(a) # This prints variable a
```

Block Comments(?)

```
"""
This is technically a docstring.
Anything written between these are ignored by the python interpre
but it is really a docstring!
"""
```

String Formatting

How do you construct strings in Python using variables?

```
>>> name = "Matt"
>>> greeting = "Hello " + name + "! Nice to meet you."
>>> print(greeting)
Hello Matt! Nice to meet you.
```

This works - but can be more concise to use the % operator

String Formatting

Better to use string formatting

```
>>> name = "Matt"
>>> print("Hello %s! Nice to meet you."%name)
Hello Matt! Nice to meet you.
```

Pass a **tuple** for multiple arguments and will be unpacked across the string

```
>>> name = "Matt"
>>> day = "Tuesday"
>>> print("Hello %s! Nice to meet you.\nToday is %s"%(name,day))
Hello Matt! Nice to meet you.
Today is Tuesday
```

String Formatters

The following are the **basic** string formatters

%s	Format as a string. All types match a %s target
%d	Format as a Decimal (base-10 integer).
%f	Format as a Floating Point

There are others but these basics go a long way

Advanced String Formatters

Advanced String Formatters are available and can be very useful when working with **templates** etc.

```
>>> #Example Dictionary Substitutions
>>> reply = """
    Greetings...
    Hello %(name)s!
    Your age is %(age)s
    """
>>> values = {'name': 'Matt', 'age': 103}
>>> print(reply % values)
Greetings...
Hello Matt!
Your age is 103
```

Mutable and Immutable Objects

Mutable objects in python are those whose state can change

1. lists
2. dictionaries

Immutable objects in python are those whose state cannot be changed without the creation of a new object

1. numbers
2. strings
3. tuples

Conditional Logic

Using Boolean expressions to control the flow of a program

Relational Operators

```
x == y    # x is equal to y
x != y    # x is not equal to y
x > y     # x is greater than y
x < y     # x is less than y
x >= y    # x is greater than or equal to y
x <= y    # x is less than or equal to y
```

Reference: http://quant-econ.net/py/python_essentials.html#comparisons-and-logical-operators

Conditional Statements

There are three main ways to write conditional logic expressions

```
if x > 0:
    print(”x is > 0”)
```

```
if x > 0:
    print(”x is > 0”)
else:
    print(”x is <= 0”)
```

```
if x > 0:
    print(”x is > 0”)
elif x == 0:
    print(”x is = 0”)
else:
    print(”x is < 0”)
```

Combining Conditions

Conditional statements can be combined using

1. **and**
2. **or**
3. **not**

```
if x >= 0:
    if x <= 10:
        print(”X is greater than 0 AND less than or equal to 10”)
```

can be written

```
x >= 0 and x <= 10
```

Iteration

The while loop:

```
#Collatz Conjecture
while n != 1:
    print(n)
    if n%2 == 0:
        n = n/2
    else:
        n = n*3+1
```

Iteration

The for loop:

```
for i in [1,'A',2,'B']:
    print(i)
    print(type(i))
```

List Comprehensions

See **intro-to-python-two.ipynb**

Functions

Functions are very useful for collecting a sequence of instructions

```
def collatz(n):
    seq = []
    while n != 1:
        seq.append(n)
        if n%2 == 0:
            n = n/2
        else:
            n = n*3+1
    return seq
```

Functions

Why use functions?

1. Incredibly useful to reuse code when performing the same operation many times
2. Can make your program much easier to read by breaking big tasks into many small tasks
3. Easier code to read is easier to debug
4. Can import your functions into other programs without rewriting them.

Functions: Basic Syntax

See **intro-to-python-two.ipynb**

Recursion

Functions and **Conditional Statements** can be combined to produce recursive loops when a function calls itself.

See intro-to-python-two.ipynb

References:

`http://openbookproject.net/thinkcs/python/english3e/recursion.html`