

Introduction to Python (2)

John Kelleher

Dublin Institute of Technology

Acknowledgements

- These notes are intended to provide an introduction to Python, they are not meant to be an exhaustive explanation of the language.
- Python is a very popular language and there are a lot of resources online for learning Python. I encourage you to use them. Indeed, I drew on several online Python tutorials when preparing these notes.
- If you are looking for a good introductory text to Python I would also recommend the book `Python Programming: An Introduction to Computer Science` 2nd Edition by John Zelle, I found it very useful in preparing these notes.

- There are two versions of Python that are currently available Python 2.x and Python 3.x. These notes assume you are using Python 3.x. But if you are using 2.x you should be fine once you keep an eye out for the following differences:
 - 1 division works slightly differently in the different versions of Python (in 2.x division rounds and in 3.x it doesn't).
 - 2 in 2.x you don't need to put brackets around parameter lists in function calls (e.g., *print x*) in 3.x you do (e.g., *print(x)*).

- 1 Control Structures
 - Code Blocks
 - Conditionals
 - Iterators
- 2 Functions in Python
 - Lambda Functions
- 3 Object Oriented Python
- 4 Handling Files
- 5 Summary

Definition

In python, blocks are created by the use of a colon, followed by an indented section of text

Example

```
if foo==bar:
    do something
    do another thing
    a final thing
do this regardless
```

Definition

Conditional can be used to control the flow of a program. Code inside a conditional will only be executed when the specified condition is met.

Example

```
temp = 50
if temp < 55:
    print("I should wear a coat today")
```

Table : Numerical Comparison Operators

Operator	Relationship
<	less than
<=	less than or equal to
==	equal to (note this is two "=" signs, not one)
!=	not equal to
>	greater than
>=	greater than or equal to

Example

```
# Initialize two variables x,y to 3,4
x = 3
y = 4
# if x and y are equal to each other,
# print "equal", otherwise "unequal"
if x == y:
    print("equal")
else:
    print("unequal")
```

Note: when we use the Python interpreter we have to add an extra blank line in order for it to detect that the next block is complete.

Example

```
# Initialize two variables x,y to 3,4
x = 3
y = 4
# if x and y are equal to each other,
# print "equal", otherwise if
# x is less than y, print "x is less than y"
# otherwise, print "x is greater than y"
if x == y:
    print("equal")
elif x < y:
    print("x is less than y")
else:
    print("x is greater than y")
```

Table : Some Word Comparison Operators

Function	Meaning
s.startswith(t)	test if s starts with t
s.endswith(t)	test if s ends with t
t in s	test if t is contained inside s
s.islower()	test if all cased characters in s are lowercase
s.isupper()	test if all cased characters in s are uppercase
s.isalpha()	test if all characters in s are alphabetic
s.isalnum()	test if all characters in s are alphanumeric
s.isdigit()	test if all characters in s are digits
s.istitle()	test if s is titlecased (all words in s have have initial capitals)

Truth values

Nothing is false. That is:

`False`

`None`

`0`

`[]` (empty list)

`{}` (empty dict)

`''` (empty string)

`()` (empty tuple)

Booleans

You can combine conditions with *and* and *or*, and negate with *not*

```
>>> y=[0,4,7,9]
>>> x = 8
>>> if 5 < x < 10 and x not in y:
...     print("x is between 5 and 10,\
and is not in the list y")
```

```
x is between 5 and 10, and is not in the list y
>>>
```

For Loops

Iterate: to do (something) over again or repeatedly. It is easy to iterate over lists with **for loops**

Example

```
# Multiply each item in mylist by 2,  
# and print the result (don't change the  
# values of mylist)  
>>> mylist = [2,4,6,8]  
>>> for item in mylist:  
...     print (item*2),
```

Notice the comma at the end of the print statement, which tells Python to produce its output on a single line.

List Expressions

A *list comprehension* is a concise way to operate on every element of a list

Example

```
# Create a new list which, in which
# each value of the origlist is doubled
>>> origlist = [1,2,3,4]
>>> newlist = [item * 2 for item in origlist]
>>> for item in newlist:
...     print item,
```

List Expressions

A *list comprehension* is a concise way to operate on every element of a list

Your turn:

For each item in *mylist*, multiply by 2 and add 3, (*do* change the values of *mylist*)

List Expressions

A *list comprehension* is a concise way to operate on every element of a list

Example

```
>>> mylist=[x * 2 + 3 for x in mylist]
```


We can also use loops and conditionals together, which is very powerful

Example

```
>>> mylist = [4,50,8,9,20]
>>> for item in mylist:
...     if item % 4 == 0:
...         print(str(item) + " is divisible
...               by 4")
```

Your turn:

Create a list called *ab* and put all words from list of words into it which start with 'ab'

Iterators

```
>>> s = 'absolutely this is some absolutely  
      abominable text'  
>>> text1 = s.split()  
>>> text1  
['absolutely', 'this', 'is', 'some',  
 'absolutely', 'abominable', 'text']  
>>> ab = []  
>>> for word in set(text1):  
...     if word.startswith('ab'):  
...         ab.append(word)  
  
>>> print(ab)  
['abominable', 'absolutely']  
>>>
```

While loop example

- 1 Create a list *myList* from 1:10
- 2 Choose a random item from this list until the item you choose is your lucky number (your choice of number from 1 to 10 (inclusive)).
- 3 Count how many times did your program have to choose a number? (Use the `random.choice` function)

Example

```
>>> import random
>>> mycount=0
>>> myList=range(1,11)
>>> myNumber=3
>>> while random.choice(myList) != myNumber:
...     mycount+=1

>>> print("It took " + str(mycount) + "
        times")
```

Your Turn:

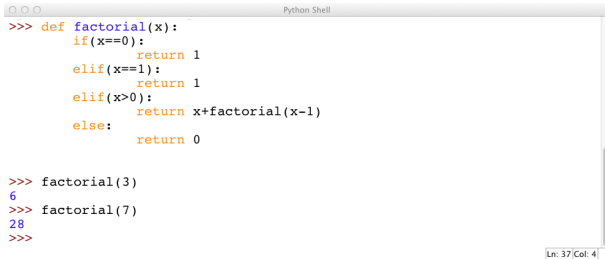
- 1 Define `sent` to be the list of words `['she', 'sells', 'sea', 'shells', 'by', 'the', 'sea', 'shore']`.
- 2 Now write code to perform the following tasks:
 - 1 Print all words beginning with `sh`
 - 2 Print all words longer than four characters

How do you define a function in Python?

- The keyword `def` tells the interpreter that we are defining a function, this is followed by the function name and the parameter list and the line is ended with a colon (:)
- The following lines are indented to show that they are part of the function.
- The blank line at the end is obtained by hitting the ENTER key twice and lets Python know that the definition is finished.

```
>>> def percentage(count, total):  
...     return 100 * count/total  
  
>>> percentage(25,100)  
25.0  
>>> percentage(3,8)  
37.5  
>>>
```


- **Your turn:** Define a function that returns the factorial of a number and use it to compute the factorial of 3 and of 7.



```
>>> def factorial(x):  
    if(x==0):  
        return 1  
    elif(x==1):  
        return 1  
    elif(x>0):  
        return x+factorial(x-1)  
    else:  
        return 0  
  
>>> factorial(3)  
6  
>>> factorial(7)  
28  
>>>
```

Ln: 37 Col: 4

- One problem with entering functions interactively into a Python shell is that the definitions are lost when we quit the shell.
- Programs are usually created by typing definitions into a separate file called a module or script. This file is save on a disk so that it can be used over and over again.
- Let's illustrate the use of a module file by writing and running a couple program.
- Our program will illustrate a mathematical concept known as chaos.

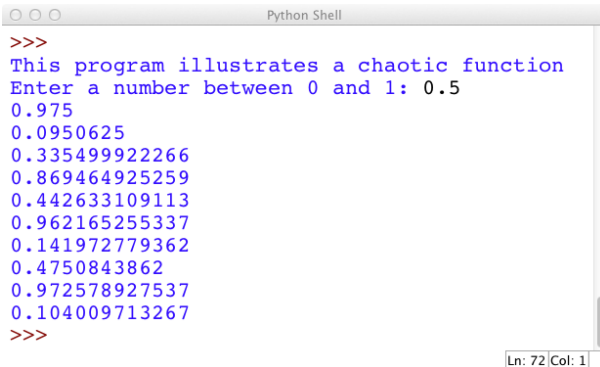
- 1 In the IDLE environment go to **File→New Window**
- 2 In the new window that opens up, enter the Python program listed on the next slide, don't forget the **:s**

```
# File: chaos.py
# A simple program illustrating chaotic
  behaviour.

def main():
    print("This program illustrates a
          chaotic function")
    num = input("Enter a number between 0
                and 1: ")
    x = float(num)
    for i in range(10):
        x = 3.9 * float(x) * (1-float(x))
        print(x)

main()
```

- You can see that this examples defines a function called main.
 - By convention programs are often placed in a function called main.
 - The last line of the example, that is not indented and hence is not inside the scope of the function, is the command to invoke the function.
- 1 Once you have entered the Python code go to **File→Save** and save the script with the name **chaos.py** in the directory that you are using.
 - 2 To run your script: **Run→Run Module**; you should see something similar to what the output illustrated on the next slide.



```
Python Shell

>>>
This program illustrates a chaotic function
Enter a number between 0 and 1: 0.5
0.975
0.0950625
0.335499922266
0.869464925259
0.442633109113
0.962165255337
0.141972779362
0.4750843862
0.972578927537
0.104009713267
>>>
```

Ln: 72 Col: 1

- Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called "lambda".

- This piece of code shows the difference between a normal function definition ("f") and a lambda function ("g"):
- As you can see, f() and g() do exactly the same and can be used in the same ways.

```
>>> def f(x): return x**2
>>> print(f(8))
64
>>> g = lambda x: x**2
>>> print(g(8))
64
>>>
```

- Note that the lambda definition does not include a "return" statement – it always contains an expression which is returned.
- Also note that you can put a lambda definition anywhere a function is expected, and you don't have to assign it to a variable at all.

- The following code fragments demonstrate the use of lambda functions.

```
>>> def make_incrementor (n): return lambda  
    x: x+n  
>>> f = make_incrementor(2)  
>>> g = make_incrementor(6)  
>>> print (f(42))  
44  
>>> print (g(42))  
48  
>>> print (make_incrementor(22) (33))  
55  
>>>
```

- The code on the last slide defines a function "make_incrementor" that creates an anonymous function on the fly and returns it.
- The returned function increments its argument by the value that was specified when it was created.
- You can now create multiple different incrementor functions and assign them to variables, then use them independent from each other.
- As the last statement demonstrates, you don't even have to assign the function anywhere – you can just use it instantly and forget it when it's not needed anymore.

- Python is fully object-oriented:
 - 1 you can define your own classes,
 - 2 instantiate the classes you've defined,
 - 3 inherit from you own or built in classes.

- Defining a class in Python is simple: a class is a collection of methods and methods are just functions, which we already know how to define.
- We use the keyword **class** followed by the class name and a colon **:** to signal the start of a class definition:

```
class <class-name>:  
    <method-definition>
```

- Some things that you should be aware of when defining a class:
 - In Python indentation is part of the language, so it is the indentation of the method definition that tells the Python interpreter which methods in the module belong to a class.
 - The `__init__` method is called when a class is instantiated.
 - The `self` parameter (similar to `this` in Java): whenever a method is called, a reference to the main object is passed as the first argument, by convention you always call this first argument to your methods `self`.
 - Class variables can be declared inside any method by assigning a value to them; however, you need to ensure that the method declaring the variable is always called before a method that uses that variable.

```
class MyClass:

    def __init__(self, arg1, arg2):
        self.att1 = arg1
        self.att2 = arg2
        self.att3 = "default"

    def sumAtt1Att2(self):
        return self.att1 + self.att2

    def getAtt3(self):
        return self.att3

    def setAtt3(self, newVal):
        self.att3 = newVal
```



```
def main():  
    instance = MyClass(10, 15)  
    print(instance.sumAtt1Att2())  
    print(instance.getAtt3())  
    instance.setAtt3("HELLO WORLD")  
    print(instance.getAtt3())  
  
main()
```

Output:

```
>>>  
25  
default  
HELLO WORLD  
>>>
```

Your Turn:

- Using the code example on the previous slide for reference, create a Python module that:
 - 1 defines a class,
 - 2 defines a main() method that creates an instance of your class and invokes some of its methods,
 - 3 invokes your main() method.

- Inheritance is simple in Python: you simply list the classes you want your new class to extend from after the class name in the definition.

```
class SubClassName (ParentClass1,  
    ParentClass2, ...):  
    <method-definition>
```

```
class Parent:
    def __init__(self):
        print "Parent Constructor"

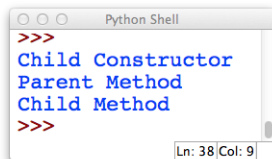
    def parentMethod(self):
        print "Parent Method"

class Child(Parent):
    def __init__(self):
        print "Child Constructor"

    def childMethod(self):
        print "Child Method"

def main():
    childInstance = Child()
    childInstance.parentMethod()
    childInstance.childMethod()

main()
```



Your Turn:

- Using the code example on the previous slide for reference, extend your Python module to include a class that inherits from your original class.
- Experiment with your code to find out whether or not the parent init method is invoked when an instance of your child class is created. If it isn't, how can you get it to be invoked?

- An important part of any programming language is to be able to interact with files on the system.
- The next two slides list programs that illustrate the most popular operations that you need to interact with files in Python.
- Study these two programs so that you understand how they work.

```
*filehandling1.py - /Users/jkelleher/teaching/ai/ai2/practical/2011/Labs/week1/examples/filehandling1.py*

#Create a file of usernames from a file of names
from __future__ import print_function
def main():
    #get the file names
    infilename = input("What file are the names in? ")
    outfilename = input("What file should the usernames go in? ")
    #open the files
    infile = open(infilename, "r")
    outfile = open(outfilename, "w")
    #process each line of the input file
    for line in infile:
        #get the first and last names from line
        first, last = line.split()
        #create the username
        uname = (first[0]+last[:7]).lower()
        #write it to the output file
        print(uname, file=outfile)

    #close both files
    infile.close()
    outfile.close()
    print("Usernames have been created")

main()
```

```
Python Shell

>>>
What file are the names in? "names.txt"
What file should the usernames go in? "unames.txt"
Usernames have been created
>>>

Ln: 60 Col: 4 18
```

```
processdir.py - /Users/jkelleher/teaching/ai/ai2/practical/2011/Labs/week1/examples/processdir.py
#iterate over a directory of files and print out the file names
import os
def process_dir():
    flist = os.listdir("./mydir/")
    print(flist)
    for f in flist:
        fPath = os.path.join("./mydir/" + f)
        infile = open(fPath, 'r')
        for line in infile:
            print(line)

        infile.close()
    print("Finished Processing Dir")

process_dir()
```

```
Python Shell
>>>
['file1.txt', 'file2.txt']
line 1 file1.txt
line 1 file2.txt
Finished Processing Dir
>>>
```

Ln: 75 Col: 4

Ln: 13 Col: 38

Your Turn:

- ❶ Make a directory called *mydata* and create some text files in the directory, put a couple of lines of random text in each file.
- ❷ Write a program that will iterate over the files in the directory and for each file:
 - ❶ print out the first line of each file to the screen
 - ❷ create a copy of the file with the name *filenamecopy.txt*, where *filename* is the name of the original file.

- 1 Control Structures
 - Code Blocks
 - Conditionals
 - Iterators
- 2 Functions in Python
 - Lambda Functions
- 3 Object Oriented Python
- 4 Handling Files
- 5 Summary