

Web Science

Lecture 2

01/19/2017

CS 432/532 Spring 2017

Old Dominion University

Department of Computer Science

Sawood Alam <salam@cs.odu.edu>

Originally prepared by Hany SalahEldeen Khalil



Original Lectures

CS 495 Python and Web Mining

<http://www.cs.odu.edu/~hany/teaching/cs495-f12/>

By Hany SalahEldeen Khalil



Lecture Outline

Python Programming

- We will learn how to:
 - program in Python
 - write high quality code
 - utilize numerous libraries and APIs



Python

Taming the beast!



Python

- It's an open source programming language.
- Compiled and Interpreted.
- Slower than C/C++ but with the difference in speed is negligible for most applications.
- Developed in the late 1980s.



Why Python?

- It is a scripting language.
- Fast in development and prototyping.
- Fast in testing functionality.
- Pluggable to other C/C++/Java code.
- Object oriented.
- Has hundreds of libraries.
- Automatically convert variable types.
- Clean and easy to read as **white space is part of the syntax!**



Expression vs. Statement

Expression

- Represents something.
 - Python *Evaluates* it.
 - Results in a value.
-
- Example:
 - 5.6
 - $(5/3)+2.9$

Statement

- Does something.
 - Python *Executes* it.
 - Results in an action.
-
- Example:
 - `print "Barcelona FC is Awesome!"`
 - `import sys`



Similarity with C Syntax

- Mostly similar to C/C++ syntax but with several exceptions.
- Differences:
 - White spaces for indentation
 - No "{}" for blocks
 - Blocks begin with ":"
 - NO type declaration
 - No ++, -- operators
 - Keywords
 - No && and ||
 - No switch/case



Starting & Exiting Python REPL

```
[user@host ~]$ python
```

```
Python 2.6.5 (r265:79063, Jan 21 2011, 12:09:23)  
[GCC 4.4.4 20100726 (Red Hat 4.4.4-13)] on  
linux2
```

```
Type "help", "copyright", "credits" or "license" for  
more information.
```

```
>>> ctrl + D
```

```
[user@host ~]$
```



Our Hello World!

```
[user@host ~]$ python
```

```
Python 2.6.5 (r265:79063, Jan 21 2011, 12:09:23)  
[GCC 4.4.4 20100726 (Red Hat 4.4.4-13)] on  
linux2
```

```
Type "help", "copyright", "credits" or "license" for  
more information.
```

```
>>> print "hello world"  
hello world
```



Simple Data Types

Integer: 7

Float: 87.23

String: "abc", 'abc'

Boolean: False, True



Simple Data Types: String

- Concatenation: "Python" + "Rocks" → "PythonRocks"
- Repetition: "Python" * 2 → "PythonPython"
- Slicing: "Python"[2:3] → "th"
- Size: len("Python") → 6
- Index: "Python"[2] → 't'
- Search: "x" in "Python" → False
- Comparison: "Python" < "ZOO" → True
(lexicographically)



Compound Data Types: List

- The equivalent of array or vector in c++.
- `X = [0, 1, 2, 3, 4]`
 - Creates a pre-populated array of size 5.
- `Y = []`
- `X.append(5)`
 - X becomes [0, 1, 2, 3, 4, 5]
- `len(X)`
 - Gets the length of X which is 6



Compound Data Types: List

```
>>> mylist = [0, 'a', "hello", 1, 2, ['b', 'c', 'd']]  
>>> mylist [1]  
a  
>>> mylist [5][1]  
c  
>>> mylist[1:3]  
['a', "hello", 1]  
>>> mylist[:2]  
[0, 'a', "hello"]
```



Compound Data Types: List

```
>>> mylist = [0, 'a', "hello", 1, 2, ['b', 'c', 'd']]  
>>> mylist[3:]  
[1, 2, ['b', 'c', 'd']]  
>>> mylist.remove('a')  
>>> mylist  
[0, "hello", 1, 2, ['b', 'c', 'd']]
```



Compound Data Types: List

- >>> mylist.reverse() → Reverse elements in list
- >>> mylist.append(x) → Add element to end of list
- >>> mylist.sort() → Sort elements in list ascending
- >>> mylist.index('a') → Find first occurrence of 'a'
- >>> mylist.pop() → Removes last element in list



Compound Data Types: Tuple

- $X = (0, 1, 2, 3, 4)$
 - Creates a pre-populated array of **fixed** size 5.
 - `print X[3]` → prints 2



Compound Data Types: Tuple vs. List

- Lists are mutable, tuples are immutable.
- Lists can be resized, tuples can't.
- Tuples are slightly faster than lists.



Compound Data Types: Dictionary

- An array indexed by a string.
- Denoted by { }

```
>>> marks = { "science": 90, "art": 25 }
```

```
>>> print(marks["art"])
```

```
25
```

```
>>> marks["chemistry"] = 75
```

```
>>> print(marks.keys())
```

```
["science", "art", "chemistry"]
```



Compound Data Types: Dictionary

- `dict = { "fish": 12, "cat": 7}`
- `dict.has_key('dog')` → `False` (To check if the dictionary has 'dog' as a key)
- `dict.keys()` (Gets a list of all keys)
- `dict.values()` (Gets a list of all values)
- `dict.items()` (Gets a list of all key-value pairs)
- `dict["fish"] = 14` → Assignment



Variables

- Everything is an object.
- No need to declare.
- No need to assign.
- Not strongly typed.
- Assignment = reference
 - Ex:

```
>>> X = ['a', 'b', 'c']  
>>> Y = X  
>>> Y.append('d')  
>>> print(X)  
['a', 'b', 'c', 'd']
```



Input / Output

- **Input:**
 - **Without a Message:**

```
>>> x = input()
```

```
3
```

```
>>> x
```

```
3
```

- **With a Message:**

```
>>> x = input('Enter the number: ')
```

```
Enter the number: 3
```

```
>>> x
```

```
3
```



Input / Output

- Input:
 - Without evaluation:

```
>>> x = input()
```

```
3+4
```

```
>>> x
```

```
"3+4"
```



File: Read

- **Input:**

- `>>> f = open("input_file.txt", "r")`
 - File handle
 - Name of the file
 - Mode
- `>>> line = f.readline()`
 - Read one line at a time
- `>>> f.close()`
 - Stop using this file and close



File: Write

- Output:

- ```
>>> f = open ("output_file.txt", "w")
```

File handle      Name of the file      Mode
- ```
>>> line = f.write("Hello how are you?")
```

Write a string to the file
- ```
>>> f.close()
```

Stop using this file and close



# Control Flow

- Conditions:
  - if
  - if / else
  - if / elif / else
- Loops:
  - while
  - for
  - for loop in file iterations



# Conditions

- The condition must be terminated with a colon ":"
- Scope of the loop is the following indented section

```
>>> if score == 100:
 print("You scored a hundred!")
elif score > 80:
 print("You are an awesome student!")
else:
 print("Go and study!")
```



# Loops: while

```
>>> i = 0
>>> while i < 10:
 print(i)
 i = i + 1
```

- Do not forget the : at the end of the condition line!



# Loops: for

```
>>> for i in range(10):
 print(i)
```

```
>>> myList = ['hany', 'john', 'smith', 'aly', 'max']
>>> for name in myList:
 print(name)
```

- Do not forget the : at the end of the condition line!



# Loops: Inside vs. Outside

```
for i in range(3):
 print("Iteration {}".format(i))
print("Done!")
```

Iteration 0  
Done!  
Iteration 1  
Done!  
Iteration 2  
Done!

```
for i in range(3):
 print("Iteration {}".format(i))
print("Done!")
```

Iteration 0  
Iteration 1  
Iteration 2  
Done!



# Loops: for in File Iterations

```
>>> f = open ("my_file.txt", "r")
>>> for line in f:
 print(line)
```



# Control Flow Keywords: pass

- It means do nothing

- >>> **if** x > 80:  
    **pass**  
  **else:**  
    **print("You are less than 80!")**



# Control Flow Keywords: break

- It means **quit** the loop
- >>> **for** name in myList:  
    **if** name == "aly":  
        **break**  
    **else**:  
        print(name)

→ This will print all names before “aly”



# Control Flow Keywords: continue

- It means skip this iteration of the loop

- >>> **for** name in myList:  
    **if** name == "aly":  
        **continue**  
    **else**:  
        print(name)

→ This will print all names except “aly”



Now, let's dig some more  
into *Python* ...



# Functions

- So far you have learned how to write regular small code in python.
- Code for finding the biggest number in a list:

```
mylist = [2,5,3,7,1,8,12,4]
maxnum = 0
for num in mylist:
 if (num>maxnum):
 maxnum = num
print "The biggest number is: {}".format(maxnum)
```



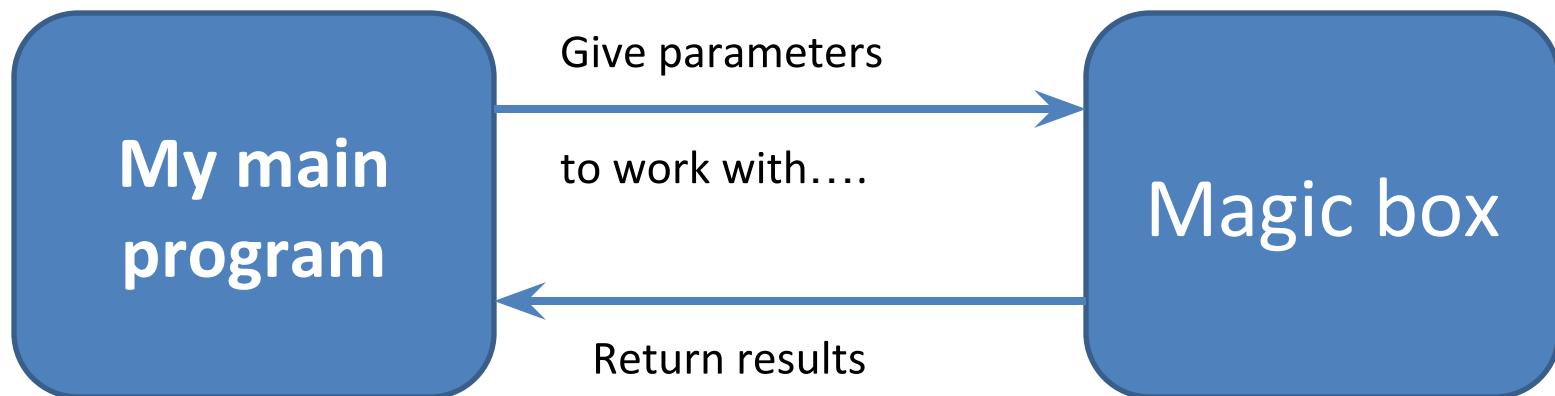
# Functions

- But what if the code is a bit more complicated and long?
- Writing the code as one blob is bad!
  - Harder to read and comprehend
  - Harder to debug
  - Rigid
  - Non-reusable



# Functions

```
def my_funtion(parameters):
 do stuff
```



# Functions

- Back to our example:

```
mylist = [2,5,3,7,1,8,12,4]
```

```
maxnum = getMaxNumber(mylist)
```

```
print "The biggest number is: {}".format(maxnum)
```



# Functions

- While you can make the function getMaxNumber as you wish

```
def getMaxNumber(list_x):
 maxnum = 0
 for num in list_x:
 if (num>maxnum):
 maxnum = num
 return maxnum
```



# Testing

```
def getMaxNumber(list_x):
```

```
 """
```

Returns the maximum number from the supplied list

```
>>> getMaxNumber([4, 7, 2, 5])
```

```
7
```

```
>>> getMaxNumber([-3, 9, 2])
```

```
9
```

```
>>> getMaxNumber([-3, -7, -1])
```

```
-1
```

```
"""
```

```
maxnum = 0
```

```
for num in list_x:
```

```
 if (num>maxnum):
```

```
 maxnum = num
```

```
return maxnum
```

```
if __name__ == '__main__':
```

```
 import doctest
```

```
 doctest.testmod()
```



# Testing

```
def getMaxNumber(list_x):
```

```
 """
```

Returns the maximum number from the supplied list

```
>>> getMaxNumber([4, 7, 2, 5])
```

```
7
```

```
>>> getMaxNumber([-3, 9, 2])
```

```
9
```

```
>>> getMaxNumber([-3, -7, -1])
```

```
-1
```

```
"""
```

```
maxnum = 0
```

```
for num in list_x:
```

```
 if (num>maxnum):
```

```
 maxnum = num
```

```
return maxnum
```

```
if __name__ == '__main__':
```

```
 import doctest
```

```
 doctest.testmod()
```

```
$ python max_num.py
```

```

```

```
File "max_num.py", line 8, in __main__.getMaxNumber
```

```
Failed example:
```

```
 getMaxNumber([-3, -7, -1])
```

```
Expected:
```

```
 -1
```

```
Got:
```

```
 0
```

```

```

```
1 items had failures:
```

```
 1 of 3 in __main__.getMaxNumber
```

```
Test Failed 1 failures.
```



# Functions

- Or...

```
def getMaxNumber(list_x):
 return max(list_x)
```



# Functions

- Remember:
  - All arguments are passed by value
  - All variables are local unless specified as global
  - Functions in python can have several arguments or none
  - Functions in python can return several results or none



# Functions

- Remember:
  - All arguments are passed by value
  - All variables are local unless specified as global
  - Functions in python can have several arguments or none
  - Functions in python can return several results or none
    - This is *AWESOME!*



# Functions

- Example of returning several values

```
def getMaxNumberAndIndex(list_x):
 maxnum = 0
 index = -1
 i = 0
 for num in list_x :
 if (num>maxnum):
 maxnum = num
 index = i
 i = i + 1
 return maxnum, index
```



# Functions

- And you call it like this:

```
mylist = [2,5,3,7,1,8,12,4]
```

```
maxnum, idx = getMaxNumberAndIndex(mylist)
```

```
print("The biggest number is: {}".format(maxnum))
```

```
print("It's index is: {}".format(idx))
```



# Class

```
class Student:
 count = 0 # class variable
 def __init__(self, name): # Initializer
 self.name = name
 self.grade = None
 Student.count += 1
 def updateGrade(self, grade): # Instance method
 self.grade = grade
if __name__ == "__main__": # Execute only if script
 s = Student("John Doe")
 s.updateGrade("A+")
 s.grade #=> "A+"
 Student.count #=> 1
```



# Writing Clean Code

- Programmers have a terrible short term memory



# Writing Clean Code

- Programmers have a terrible short term memory

You will have to learn  
to live with it!



# Writing Clean Code

- To fix that we need to write clean readable code with a lot of comments.



# Writing Clean Code

- To fix that we need to write clean readable code with a lot of comments.
- You are the narrator of your own code, so make it interesting!
- Ex: Morgan freeman



[http://www.youtube.com/watch?v=lblql-IN1B4&feature=player\\_detailpage#t=77s](http://www.youtube.com/watch?v=lblql-IN1B4&feature=player_detailpage#t=77s)



# Writing Clean Code

- Comments start with a **#** and end at the end of the line.

```
mylist = [2,5,3,7,1,8,12,4]
```

```
The function getMaxNumberAndIndex will be
called next to retrieve
```

```
the biggest number in list "mylist" and the
index of that number.
```

```
maxnum, idx = getMaxNumberAndIndex(mylist)
print "The biggest number is: " + str(maxnum)
print "It's index is: " + str(idx)
```



# Creating Python Files

- Python files end with ".py"
- To execute a python file you write:

```
>>> python myprogram.py
```



# Creating Python Files

- To make the file “a script”, set the file permission to be executable and add this shebang in the beginning:

```
#!/usr/bin/python
```



The path to Python installation

or better yet

```
#!/usr/bin/env python
```



# Building on the Shoulders of Giants!

- You don't have to reinvent the wheel.....  
*someone has already done it better!*



# Modules

- Let's say you have this awesome idea for a program, will you spend all your time trying to figure out the **square root** and how it could be implemented and utilized?



# Modules

- Let's say you have this awesome idea for a program, will you spend all your time trying to figure out the **square root** and how it could be implemented and utilized?

No!



# Modules

- We just call the math library that has the perfect implementation of square root.

```
>>> import math
>>> x = math.sqrt(9.0)
```

*Or*

```
>>> from math import sqrt
>>> x = sqrt(9.0)
```



# Modules

- To import all functions in a library we use the wildcard: \*

```
>>> from string import *
```

**Note:** Be careful upon importing "from" several files, there might be two modules named the same in different libraries.



# Your Programs are Your Butlers!

- You are Batman! Your programs are your Alfreds!

- *Send them work:*



# Command-Line Arguments

- To get the command line arguments:
- *>>> import sys*
- The arguments are in *sys.argv* as a *list*



# What Happens When Your Program Goes

*Kaboom!?*



# Bad Scenario

```
>>> sum_grades = 300
>>> number_of_students = input()
>>> average = sum_grades / number_of_students
```

→ What if the user wrote 0?



# Bad Scenario

```
>>> sum_grades = 300
>>> number_of_students = input()
0
>>> average = sum_grades / number_of_students
```

→ Error! Divide by Zero



# Bad Scenario

```
>>> sum_grades = 300
>>> number_of_students = input()
0
>>> average = sum_grades / number_of_students
```

→ Error! Divide by Zero

Remember: User input is evil!



# Precautions: Exception Handling

You can just say:

**try:**

```
 average = sum_grades / number_of_students
```

**except:**

```
 # this catches if something wrong happens
```

```
 print "Something wrong happened, please check it!"
```

```
 average = 0
```



# Precautions: Exception Handling

Or if you have an idea what exception could it be:

**try:**

```
 average = sum_grades / number_of_students
```

**except ZeroDivisionError:**

```
 # this catches if a number was divided by zero
```

```
 print "You Evil User!.....you inserted a zero!"
```

```
 average = 0
```



# Precautions: Exception Handling

Or several exceptions you are afraid of:

**try:**

```
average = sum_grades / number_of_students
```

**except ZeroDivisionError:**

```
this catches if a number was divided by zero
print "You Evil User!.....you inserted a zero!"
```

```
average = 0
```

**except IOError:**

```
this catches errors happening in the input process
print "Something went wrong with how you enter words"
average = 0
```



# Generators

```
def fib():
 a = b = 1
 while True:
 yield a
 a, b = b, a + b
```

```
f = fib()
print(next(f)) #=> 1
print(next(f)) #=> 1
print(next(f)) #=> 2
print(next(f)) #=> 3
print(next(f)) #=> 5
```



# Python Tips and Tricks

- `range(start, end, increment)`
  - You can design a specific loop with that
- Swap variable values using multiple assignment
  - `a, b = b, a`



# Python Tips and Tricks

“in” and “not in” operators

- In loops
  - `for line in lines`
  - `for line not in lines`
- In conditions
  - `if item in list`
  - `if item not in list`



# Python Tips and Tricks

## List comprehensions

```
squares = []
for x in range(10):
 squares.append(x**2)
```

# Can be written like this

```
squares = [x**2 for x in range(10)]
```

# A more complex example

```
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```



# Python Tips and Tricks

- Manipulating files:

- `readline()` → reads a line from file
- `readlines()` → reads all the file as a list of lines
- `read()` → reads all the file as one string.
- `seek(offset, start)` → start could be:
  - 0 → beginning
  - 1 → current location
  - 2 → end of file



# Python Libraries: urllib

- **urllib** is a Python module that can be used for interacting with remote resources

```
import urllib.request
with urllib.request.urlopen('http://www.cs.odu.edu/') as res:
 html = res.read()
 # do something
```



# urllib Response Headers

```
import urllib.request
with urllib.request.urlopen('http://python.org/') as res:
 print("URL: {}".format(res.geturl()))
 print("Response code: {}".format(res.code))
 print("Date: {}".format(res.info()['date']))
 print("Server: {}".format(res.info()['server']))
 print("Headers: {}".format(res.info()))
```



# urllib Requests

```
import urllib.request
url = 'http://www.cs.odu.edu/'

This puts the request together
req = urllib.request.Request(url)

Sends the request and catches the response
with urllib.request.urlopen(req) as res:
 # Extracts the response
 html = res.read()
```



# urllib Request Parameters

```
import urllib.request
import urllib.parse

url = 'http://www.cs.odu.edu/'
query_args = {'q': 'query string', 'foo':'bar'}

data = urllib.parse.urlencode(query_args).encode('ascii')

req = urllib.request.Request(url, data)

with urllib.request.urlopen(req) as res:
 # Extracts the response
 html = res.read()
```



# What Happens When the Server Tells, “You Can't Get This Page!”



# URLlib2 Request Headers

```
import urllib.request
import urllib.parse

url = 'http://www.cs.odu.edu/'
query_args = {'q': 'query string', 'foo':'bar'}

headers = {'User-Agent': 'Mozilla 5.10'}

data = urllib.parse.urlencode(query_args).encode('ascii')

req = urllib.request.Request(url, data, headers)

with urllib.request.urlopen(req) as res:
 # Extracts the response
 html = res.read()

Try a nicer third-party HTTP library named 'requests'
```



# Beautiful Soup: HTML/XML Parser

```
Installation is needed before you could use any third-party library
$ pip install beautifulsoup4
```

```
from bs4 import BeautifulSoup
import urllib.request

with urllib.request.urlopen('http://www.reddit.com') as res:
 redditHtml = res.read()
 soup = BeautifulSoup(redditHtml)
 for links in soup.find_all('a'):br/> print(links.get('href'))
```



# Jupyter Notebook

The screenshot shows the Jupyter Notebook interface running on a Mac OS X system. The title bar reads "jupyter Running Code Last Checkpoint: 9 hours ago (unsaved changes)" and the address bar says "localhost examples/Notebook/". The toolbar includes standard file operations like File, Edit, View, Insert, Cell, Kernel, Help, and various cell-related icons. A Python 3 kernel icon is visible in the top right. The main content area displays a section titled "Running Code" with the following text:

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefore runs Python code.

**Code cells allow you to enter and run code**

Run a code cell using Shift-Enter or pressing the button in the toolbar above:

```
In [1]: a = 10
In [2]: print(a)
10
```

There are two other keyboard shortcuts for running code:

- Alt-Enter runs the current cell and inserts a new one below.
- Ctrl-Enter runs the current cell and enters command mode.

**Managing the Kernel**

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted. Try running the following cell and then hit the button in the toolbar above.

```
In []: import time
```



# References

- <http://introtopython.org/>
- <http://www.cs.cornell.edu/courses/cs1110/2012fa/>
- <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2011/lectures/>
- <http://courses.cms.caltech.edu/cs11/material/python/index.html>
- <http://www.cs.cornell.edu/courses/cs2043/2012sp/>
- <http://www-cs-faculty.stanford.edu/~nick/python-in-one-easy-lesson/L>
- <http://www.pythonforbeginners.com/python-on-the-web/how-to-use-urllib2-in-python/>
- <http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/>
- **Python in a Nutshell, 2nd Edition** By Alex Martelli

