Introduction to Web Science

Lecture 2
09-05-13

Old Dominion University
Department of Computer Science
CS 495/595 Fall 2013

Hany SalahEldeen Khalil  hany@cs.odu.edu
Introductions

Name: Hany SalahEldeen
Email: hany@cs.odu.edu
Office: ECSB Third Floor 3102
Cell: 757-648-9944

Original lectures: cs495 Python and Web Mining
http://www.cs.odu.edu/~hany/teaching/cs495-f12/

Any question about python...you know where to find me!
Lecture Outline

Python Programming

• We will learn how to:
  • program in Python.
  • write high quality code.
  • utilize hundreds of 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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represents something.</td>
<td>Does something.</td>
</tr>
<tr>
<td>Python <em>Evaluates</em> it.</td>
<td>Python <em>Executes</em> it.</td>
</tr>
<tr>
<td>Results in a value.</td>
<td>Results in an action.</td>
</tr>
</tbody>
</table>

**Example:**
- 5.6
- \((5/3)+2.9\)

**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 {}
  • NO type declaration.
  • No ++, -- operators.
  • Keywords.
  • Loops & Conditions end with :
  • No && and ||
Starting & Exiting Python

```
[hany@ws-dl-01 ~]$ 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
```

```
[hsany@ws-dl-01 ~]$ 
```
Our Hello World!

```
[hany@ws-dl-01 ~]$ 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

int: 7
float: 87.23
str: “abc”, ‘abc’
Booleans: False, True
Strings:

- **Concatenation**: “Python” + “Rocks” → “PythonRocks”
- **Repetition**: “Python”*2 → “PythonPython”
- **Cutting**: “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

• Lists:
  • The equivalent of array or vector in c++.
  • X = [0, 1, 2, 3, 4]
    • Creates an array of size 5 having those numbers.
  • 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

• Lists:
  >>> 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

- Lists:

```python
>>> 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

• Lists:
  >>> mylist.reverse() → Reverse elements in list
  >>> mylist.append(x) → Add element to end of list
  >>> mylist.sort() → Sort elements in list ascendingly
  >>> mylist.index('a') → Find first occurrence of ‘a’
  >>> mylist.pop() → Removes last element in list
• **Tuples:**
  • \( X = (0, 1, 2, 3, 4) \)
    • Creates an array of **fixed** size 5 having those numbers.
  • \( \text{print } X[3] \rightarrow \text{prints 2} \)
Compound Data types

- **Tuples Vs. Lists:**
  - 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 { }

```python
>>> Hany = { "science": 90, "art": 25 }
>>> print Hany["art"]
25

>>> Hany["chemistry"] = 75
>>> print Hany.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:**
    ```python
    >>> x = raw_input()
    3+4
    >>> x
    "3+4"
    ```
File Input / Output

• Input:
  • >>> F = open ("input_file.txt", "r")
    Variable name Name of the file Read directive
  • >>> line = F.readline()
    Read a line at a time
  • >>> F.close()
    Stop using this file and close
File Input / Output

- **Output:**
  - ```
  >>> F = open ("output_file.txt", "w")
  >>> line = F.write("Hello how are you?")
  >>> F.close()
  ```
  - Variable name
  - Name of the file
  - Write directive
  - Write a string to the file
  - 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

```python
>>> 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 < 100 :
    print i
    i = i + 1

• Do not forget the • at the end of the condition line!
Loops

• for:

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

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

• Do not forget the `:` at the end of the condition line!
Loops

- **for** in file iterations:

```python
>>> F = open(“my_file.txt”, “r”)
>>> for line in F:
    print line
```
Control Flow Keywords

- **pass**
  - It means do nothing or “skip”
  - `>>> if x > 80 :`
    ```python
    pass
    ```
    ```python
    else:
    print "You are less than 80!"
    ```
- **break**
  - It means quit the loop
  - `>>> for name in myList:
    if name == "Aly":`
    ```python
    break
    ```
    ```python
    else:
    print name
    ```
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:

```python
mylist = [2,5,3,7,1,8,12,4]
max_number = 0
for num in mylist:
    if ( num>max_number):
        max_number = num
print "The biggest number is: " + str(max_number)
```
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
def my_function (parameters):
do stuff

My main program

Give parameters
to work with....

Magic box

Return results
Functions

- Back to our example:

```python
mylist = [2,5,3,7,1,8,12,4]
max_number = getMaxNumber( mylist )
print "The biggest number is: " + str(max_number)
```
Functions

• While you can make the function `getMaxNumber` as you wish

```python
def getMaxNumber(list_x):
    max_number = 0
    for num in list_x:
        if num > max_number:
            max_number = num
    return max_number
```
Functions

• Or...

```python
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

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

• And you call it like this:

```python
mylist = [2,5,3,7,1,8,12,4]
max_number, index = getMaxNumberAndIndex(mylist)
print "The biggest number is: " + str(max_number)
print "It’s index is: " + str(index)
```
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=lbIqL-IN1B4&feature=player_detailpage#t=77s
Writing Clean Code

• Comments start with a `#` and end at the end of the line.

```python
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.
max_number, index = getMaxNumberAndIndex( mylist )
print “The biggest number is: “ + str(max_number)
print “It’s index is: “ + str(index)
```
Creating Python Files

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

```python
>>> python myprogram.py
```
Creating Python Files

• To make the file “a script” you add this in the beginning:

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

L = [1,2,4]
....bla bla bla
```
Building on the shoulders of giants!

• You don’t have to reinvent the wheel.....someone has already done it better!
• 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?
• 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.

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

Or

```python
>>> from math import sqrt
>>> x = sqrt(9.0)
```
• To import all functions in one library we use the joker card: *

```python
>>> 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

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

→ Error! Divide by Zero
```
Bad Scenario

```python
>>> 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:

```python
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:

```python
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:

```python
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
```
Python Tips and Tricks

• range(start, end, increment)

→ You can design a specific loop with that
Python Tips and Tricks

• in operator:
  • In loops: for line in lines:
  • In conditions: if item in list
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: urllib2

• **urllib2** is a Python module that can be used for fetching URLs.

• **urllib2** can accept a Request object to set the headers for a URL request, **urllib** accepts only a URL.

• Example:

```python
import urllib2
response = urllib2.urlopen('http://www.cs.odu.edu/~hany')
html = response.read()
# do something
response.close()
```
import urllib2
response = urllib2.urlopen('http://python.org/')
print "Response:", response

print "The URL is: ", response.geturl()
print "The response code: ", response.code
print "The Headers are: ", response.info()
print "The Date is: ", response.info()['date']
print "The Server is: ", response.info()['server']
html = response.read()
print "Get all data: ", html
URLlib2 requests

```python
url = 'http://www.www.cs.odu.edu'

# This puts the request together
request = urllib2.Request(url)

# Sends the request and catches the response
response = urllib2.urlopen(request)

# Extracts the response
html = response.read()
```
URLlib2 request parameters

# Prepare the data
query_args = { 'q':'query string', 'foo':'bar' }

# This urlencodes your data (that's why we need
# to import urllib at the top)
data = urllib.urlencode(query_args)

# Send HTTP POST request
request = urllib2.Request(url, data)

response = urllib2.urlopen(request)

html = response.read()
What happens when the server tells you can’t get this page?
import urllib2

req = urllib2.Request('http://192.168.1.2/')
req.add_header('User-agent', 'Mozilla 5.10')
res = urllib2.urlopen(req)
html = res.read()
```python
from bs4 import BeautifulSoup
import urllib2

redditFile = urllib2.urlopen("http://www.reddit.com")
redditHtml = redditFile.read()
redditFile.close()

soup = BeautifulSoup(redditHtml)
for links in soup.find_all('a'):
    print (links.get('href'))
```
References

- http://www.cs.cornell.edu/courses/cs1110/2012fa/
- http://www.cs.cornell.edu/courses/cs2043/2012sp/
- **Python in a Nutshell, 2nd Edition** By Alex Martelli
- http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/