CNT 4603: System Administration
Spring 2012

Python – Part 1

Instructor: Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 4078-823-2790

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida
What Is Python?

• Python is an elegant and robust programming language that delivers both the power and general applicability of traditional compiled languages with the ease of use of simpler scripting and interpreted languages.

• Python originated in late 1989 at the National Research Institute for Mathematics and Computer Science in the Netherlands by Guido van Rossum, with its first public release coming in early 1991.

• Van Rossum was doing research in system administration at the time and found most conventional programming languages too cumbersome or incomplete to perform the work he envisioned. He was working on automating system administration tasks and thus needed access to the power of system level calls.
What Is Python?

• Van Rossum was working on an Amoeba distributed operating system at the time and gave serious consideration to developing an Ameoba-specific language to further his research. In the end he decided to go with a generalized language and Python was born.

• Although Python has now been around for about 20 years, many people feel that it is still relatively new to the general software development industry.

• The next few pages will illustrate some of the primary features of Python and why it has become a valuable tool for system administrators.
Main Features Of Python

• Python is a high level programming language. The hierarchy of languages takes you from very low-level machine languages through assembly languages to languages like Fortran, C, and Pascal. These latter three languages are primarily responsible for creating the software development industry. The language C was responsible for generating the modern version of compiled languages like C++ and Java. Even higher-level languages which are powerful system-accessible and interpreted languages like Python, Perl, and Ruby.

• These very high-level languages provide data structures that reduce the “framework” development time that was required in earlier languages. Useful types such as Python’s lists (resizable arrays), and dictionaries (hash tables) are built into the language.
Main Features Of Python

• Python is an object-oriented language, which adds another dimension to structured and procedural languages (like C) where data and logic are discrete elements of programming. OO allows for associating specific behaviors, characteristics, and/or capabilities with the data that they execute on or are representative of.

• Python is often compared to batch or Unix shell scripting languages. Unix shell scripts handle simple tasks and there is little chance for code reusability amongst scripts. However, this is not true of Python, which allows for easy code reuse. This makes Python a scalable language through modular design capabilities.
Main Features Of Python

• Python is highly portable. This is because Python is written in C and C is an extremely portable. Any platform with an ANSI C compiler is capable of running Python.

• Python is easy to learn as it has relatively few keywords, a simple structure, and a clearly defined syntax.

• The simple syntax enhances the readability of Python code which makes it easier to write initially, and to maintain over the long term.

• Python is a robust language in that it makes it easy to identify and catch errors in your software. This robustness helps not only the program developer but also the end user.
Downloading And Installing Python

• The most obvious place to get all Python-related software is at http://python.org.

• Right now there are two current production versions of Python available, versions 2.7.2 and 3.2.2. This is a common occurrence with Python and if in doubt about which version to download, the older version will almost always have more compatibility with third party software than the newer version which requires some lead time before third party vendors can catch up with new developments in the language.

• I have both versions on my systems at the moment so that I can point out a few of the relevant differences as we move through our look at Python.
Running Python

- Download your preferred version of Python and install it accordingly on your system.

- Once you’ve downloaded Python there are three different ways to start it running.

- The simplest way is to start the interpreter interactively, entering one line of Python at a time for execution. This is illustrated on page 9.

- The next way is to run a script written in Python by invoking the interpreter on the script. To do this, first create the script file using a text editor, then simply click on the script file to execute it. This is illustrated on page 10.
Running Python

From your Program listing you should find a Python command line interpreter once you’ve downloaded and installed Python. This is what is shown here.

Notice that in this environment the commands appear on a single line. If the command is separated across lines then it must be enclosed in parentheses or an error is generated.
Create the Python Script then click on the file name in the directory where the script is stored.

```python
# This is a Python script illustrating basic Python
# Script name: Python-Part1-page10
# Script created: November 16, 2011
# Script modified: April 5, 2012
# Script author: Mark Llewellyn

print(" ")
print("Hello World")
print(2+3)
print(8/4)

# the function call to os.system("pause") is here just to pause the window
# so that you can see the results in this environment.
print(" ")
import os
os.system("pause")

# end script
```
Running Python

• The final way to run Python is through an IDE.

• Current versions of Python come with an IDE called IDLE (Python GUI), which looks like:

![Python IDLE environment]

```python
>>> print("Hello World")
Hello World
>>> 2+3
5
>>> 8/4
2.0
>>> |
```
The Python IDLE can be found in your PythonXX/Lib/idlelib folder as shown below.

Double click on the batch file to run the IDE.
Running Python

• Most of the more commonly used IDE also can be set-up for incorporating Python development.

• I use Eclipse mostly for Python development and I’ve put together another set of notes (available on the course website) that will step you through the process of configuring Eclipse for Python. There are several steps that need to be done, but it’s a fairly straightforward process and Eclipse ultimately provides a good development environment for Python scripting.

• The next page illustrates the Python perspective in Eclipse.
The Eclipse Python perspective.
Python Basics

• Now that you’ve set-up Python and tried out the various ways to run Python, it's time to learn the basics of the language so that you can write Python programs or scripts to accomplish some task.

• Comments in Python are delimited by the hash mark (#) (pound sign). Any text following a # is ignored by the interpreter.

• Multiple line comments will require a # at the start of each line.

• There are also special comments called documentation strings or “doc strings” for short that typically appear at the beginning of a Python module, a feature which should be familiar to Java programmers.

  – Unlike regular comments, doc strings are accessible at runtime and are used to automatically generate documentation.
The standard mathematical operators that you are familiar with work the same way in Python as in most other languages.

Python supports: +, -, *, /, //, %, and ** (the double slash division operator is \textit{floor division}, in which the division result is rounded down to the nearest whole number).

Operator precedence is also what you would normally expect.

Python also supports the normal set of comparison operators: <, <=, >, >=, ==, and != which all return boolean values.

The following example illustrates a few of these, but look closely at the very last example, which is a Python-specific shorthand that would be equivalent to 3 < 4 and 4 < 5.
Created on Nov 21, 2011

@author: Mark Llewellyn

```
# = 3**2 = 9, -2 * 4 = -8, -8 + 9 = 1
print("1.", -2 * 4 + 3 ** 2)
# = 14/3 = 4.667, rounded down = 4
print("2.", 14 // 3)
print("3.", 2 < 4)
print("4.", 6.2 <= 6)
print("5.", 2 < 4 and 2 == 4)
print("6.", not 6.2 <= 6)
# equivalent to (3 < 4) and (4 < 5)
print("7.", 3 < 4 and 4 < 5)
```
Python Basics - Variables

• The rules for variable naming in Python are much the same as in most other high-level languages inspired by C.

• The variable name must begin with a letter or underscore character and can include any number of letters, digits, or underscores.

• In particular no spaces are allowed in a variable name.

• Python is case sensitive.

• The assignment operator is =.
Python Basics – Operators And Data Types

• Unlike many high-level languages, Python does not support prefix and postfix increment operators, e.g., \( ++n \) or \( n++ \). This is because + and – are also unary operators.

• Python would interpret \( --n \) and \(-(-n) = n\).

• Python is dynamically typed, meaning that no pre-declaration of a variable or its type in necessary. The type and value are initialized on assignment.

• Python supports five basic numeric types:
  
  - int, long, bool
  
  - float, complex
Python Basics - Strings

• Strings in Python are identified as a contiguous set of characters in between quotation marks. Python allows for either pairs of single or double quotes.

• Triple quotes (three consecutive single or double quotes) can be used to escape special characters.

• Subsets of strings can be taken using the `index([ ] )` and `slice([ : ] )` operators, which work with indices starting at 0 in the beginning of the string and working their way from -1 at the end.

• The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

• Examples of these are shown on the next page.
```python
@Author: Mark Llewellyn

```
Python Basics - Lists

• Lists and tuples can be thought of as generic arrays which hold an arbitrary number of arbitrary Python objects.

• The items are ordered and accessed via index offsets, similar to arrays, except that lists and tuples can store different types of objects.

• Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed.

• Tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Think of a tuple as a “read-only” list.

• Subsets can be taken with the index and slice operator in the same manner as for strings. The example on the next page illustrates both lists and tuples.
```python
# create a list object
aList = [1, 2, 3, 4, 5];
print(aList);
print(aList[0]);
print(aList[2:]);
print(aList[:3]);
aList[4] = 16;
print(aList);

# create a tuple
aTUPLE = (1, "hi", 4, "bye", 6, 8);
print(aTUPLE);
print(aTUPLE[3:]);
print(aTUPLE[4:]);

# uncomment next line: error * tuple cannot be assigned
aTUPLE[4] = 12;
```

```
[1, 2, 3, 4, 5]
1
[3, 4, 5]
[1, 2, 3]
[1, 2, 3, 4, 16]
(1, 'hi', 4, 'bye', 6, 8)
(1, 'hi', 4)
(6, 8)
```
Dictionaries (or “dicts” for short) are Python’s mapping type and work like associative arrays or hashes found in Perl.

They are made up of key-value pairs. Keys can be almost any Python type, but are most commonly numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dicts are enclosed by curly braces ( { } ).

The next page shows an example that uses dicts.
Created on Nov 21, 2011

@author: Mark Llewellyn

```python
# create a dict
aDict = {
    "host": "Earth", "port": 8080
};
print("1. ", aDict);
# add an element to the dict
aDict['Hans'] = "Solo";
print("2. ", aDict);
# print all the keys in the dict
print("3. ", aDict.keys());
print("4. ", aDict["host"], ",\n\n");
for key in aDict:
    print(key, = ", aDict[key]);
```

```bash
<terminated> C:/Users/Administrator/workspace/Python/Part 1 Notes/dictExample.py
1. {'host': 'Earth', 'port': '8080'}
2. {'Hans': 'Solo', 'host': 'Earth', 'port': '8080'}
3. dict_keys(['Hans', 'host', 'port'])
4. Earth

Hans = Solo
host = Earth
port = 8080
```
Python Basics – Code Blocks

• Unlike many high-level languages which use a variety of curly braces or straight braces to denote code blocks, Python uses only indentation. The lack of extra symbols makes Python code easier to read and thus manage.

• While code blocks typically consist of many statements, recall that it is also possible for them to consist of a single statement. Indentation must be used in the single statement case as well.
Python Basics – if statement

• The standard if conditional statement has the following syntax in Python:

```python
if expression:
    if_suite
```

• If the expression is non-zero or True, then the statement `if_suite` is executed; otherwise, execution continues at the first statement after the if statement.

• Suite is the term used in Python to refer to a sub-block of code.
Python Basics – if-else statement

• Python also supports an else statement that is used with an if statement using the following syntax:

```python
if expression:
    if_suite
else:
    else_suite
```

• If the expression is non-zero or true, then the statement if_suite is executed; otherwise, the else_suite is executed. In both cases execution continues at the first statement after the if-else statement.
Python Basics – `elif` statement (else-if)

- Python also supports an “else-if” statement spelled as `elif` using the following syntax:

```python
if expression1:
    if_suite
elif expression2:
    elif_suite
else:
    else_suite
```
Python Basics – while loop

• Python also a while loop statement which executes as you would expect. The syntax of the while statement is:

```python
while expression:
    while_suite
```

• The statement(s) in the `while_suite` are executed continuously until the expression becomes zero or false; execution then continues with the first statement after the `while` statement.

• The next page shows a simple `while` statement example.
Created on Nov 21, 2011

@author: Mark Llewellyn

# while loop example
counter = 0;
while counter < 3:
    print('Loop #{}d' % (counter))
    counter += 1

Loop #0
Loop #1
Loop #2
Python Basics – for loop

• The `for` loop in Python is more like a `foreach` iterative-type loop in a shell scripting language than a traditional `for` conditional loop that works like a counter.

• Python’s `for` takes an `iterable` (such as a sequence or iterator) and traverses each element once.

• The `for` loop can take on several different variants in Python when augmented with functions. In particular the built-in function `range()` is often used with the `for` statement to make it act more like a traditional counted loop.

• The example on the next page, illustrates a typical Python `for` loop acting much like a `foreach` loop.
Created on Nov 21, 2011

@author: Mark Llewellyn

# for loop example 1
print("I like to use the Internet for: ")
for item in ["e-mail", "net-surfing", "news", "shopping"]:
    print(item)
```python
Created on Nov 21, 2011

@author: Mark Llewellyn

# for loop example 2

print ("I like to use the Internet for:", end="")
for item in ["e-mail", "net-surfing", "news", "shopping"]:
    print (item, end=" ")
```

```
I like to use the Internet for: e-mail net-surfing news shopping
```
Python Basics – for loop

• The following two examples illustrate the for loop in Python acting more like a traditional counted loop.

• The first example explicitly specifies the iterable set of values that drive the loop.

• The second example uses the built-in function range() to generate the upper end of the iterable set.
```python
Created on Nov 21, 2011

@author: Mark Llewellyn

# for loop example 3
for eachNum in [0, 1, 2, 3, 4]:
    print (eachNum, end=" ")
print("\n")
for eachNum in [4, 3, 1, 5, 6, 7, 9, 4, 5]:
    print(eachNum, end=" ")
```

```
0 1 2 3 4
4 3 1 5 6 7 9 4 5
```
Created on Nov 21, 2011

@author: Mark Llewellyn

# for loop example 4
for eachNum in range(6):
    print(eachNum)
print("\n")

<terminated> C:\Users\Administrator\workspace\Python\Part 1 Notes\forLoopExample4.py

0
1
2
3
4
5
Slightly different use of `range()` function to create values outside of the for loop and then use those range values (albeit somewhat modified, e.g. squared) in the subsequent for statement.
Python Basics – Files

- Once you are comfortable with a new language’s syntax; one of the more important aspects of the language that you need to learn is how to access files. Persistent storage really allows you to get some work done.

- Python includes a function `open()` with the following syntax for opening files:

  ```python
  handle = open(filename, mode)
  ```

- There are a couple of lesser used variants of this function, but this one will suffice for now. The function returns a file object (a handle to the file) specified by the `filename` argument and is opened according to the `mode` specified. The `mode` options are: `r` for reading (the default case), `w` for writing, `a` for appending, and `r+` for reading and writing. Files are opened by default in text mode using a UTF-8 encoding.
Created on Nov 21, 2011

@author: Mark Llewellyn

#fileName = input('Please enter the file name: ')
fileObject = open('rawtext.txt', 'r')
for eachLine in fileObject:
    print(eachLine)
fileObject.close()

This is the text file that will be opened
by the Python openFileExample1 test program.

This file was created on November 21, 2011
by Mark Llewellyn
on TestBedServer in the workspace: Python\Part 1 Notes
An alternative approach to the previous example. In this case the file is opened, all lines read, the file closed, and then the lines from the file are printed.
Python Basics – Files

• The following example shows the creation of a file using the w mode for writing to a file. By default, the file is created in the default directory if it does not exist, and is overwritten if it already exists.

• This simple example simply takes input from the console provided by the user, one line at a time and enters it into the file.

• You’ll need to be able to both write and read files for an upcoming Python scripting project, so you might want to try both of these examples on your system.
```python
import os

fileObj = open("output.txt", 'w')

while True:
    aLine = input("Enter a line ('#' to quit): ")
    if aLine != "#":
        fileObj.write("%s\n" % (aLine))
    else:
        break

fileObj.close()
```
This file was created on November 21, 2011 by MJL

Text line 1.
Text line 2.
Text line 3.