

CNT 4603: System Administration Spring 2011

Python – Part 1

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 4078-823-2790
<http://www.cs.ucf.edu/courses/cnt4603/spr2011>

Department of Electrical Engineering and Computer Science
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 2.7.1 and 3.2. 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

C:\Python31\python.exe

```
Python 3.1.3 (r313:86834, Nov 27 2010, 18:30:53) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> print ("Hello World")
```

```
Hello World
```

```
>>> 2+3
```

```
5
```

```
>>> 8/4
```

```
2.0
```

```
>>> (2
```

```
... +
```

```
... 4)
```

```
6
```

```
>>> 2 +
```

```
File "<stdin>", line 1
```

```
2 +
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>>
```

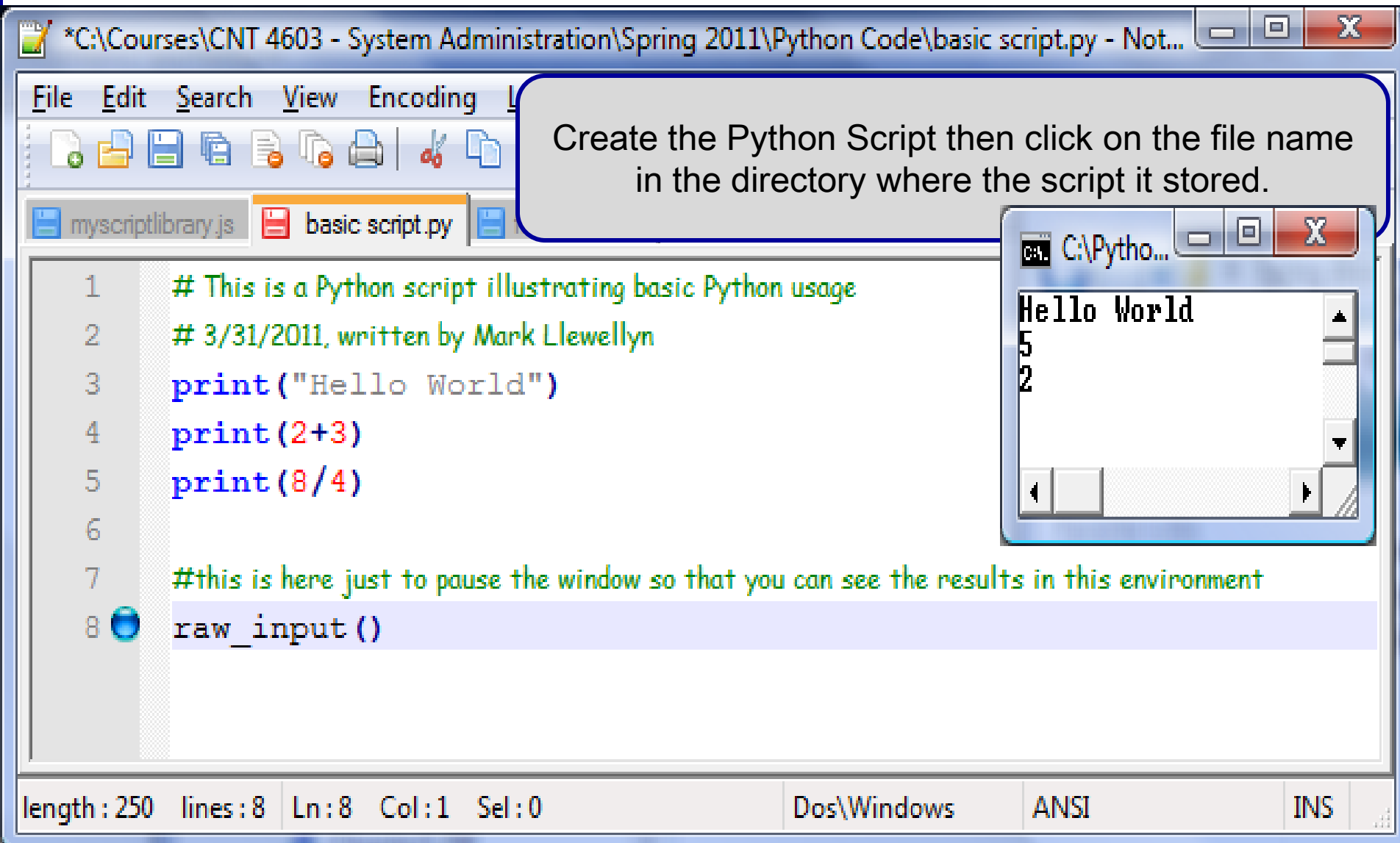
Running Python interactively

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.



Running Python



The screenshot shows a Notepad window titled "*C:\Courses\CNT 4603 - System Administration\Spring 2011\Python Code\basic script.py - Not...". The menu bar includes File, Edit, Search, View, and Encoding. The toolbar contains icons for file operations. The file explorer shows two files: "myscriptlibrary.js" and "basic script.py". The main text area contains the following Python code:

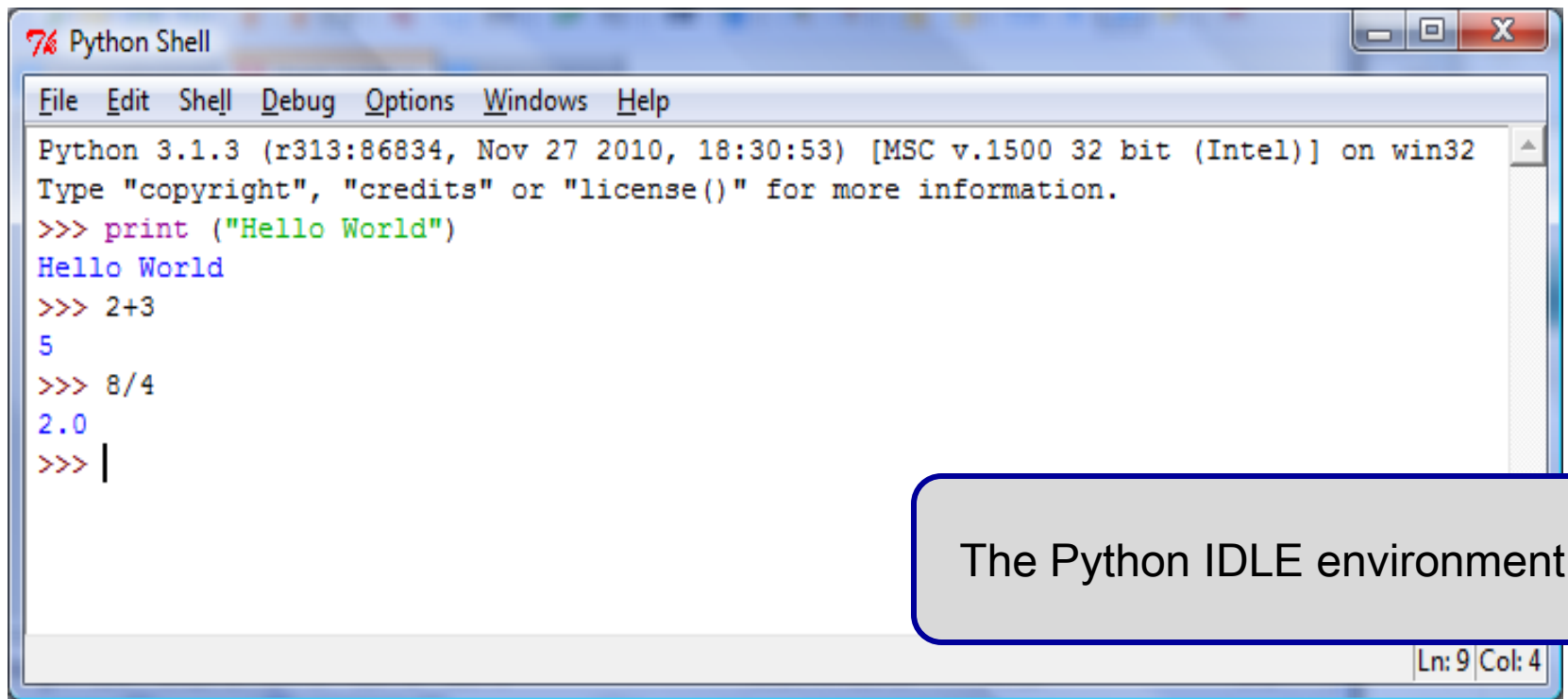
```
1 # This is a Python script illustrating basic Python usage
2 # 3/31/2011, written by Mark Llewellyn
3 print("Hello World")
4 print(2+3)
5 print(8/4)
6
7 #this is here just to pause the window so that you can see the results in this environment
8 raw_input()
```

A callout box with a blue border contains the text: "Create the Python Script then click on the file name in the directory where the script it stored." A smaller window titled "C:\Pytho..." is overlaid on the right, showing the output of the script: "Hello World", "5", and "2". The status bar at the bottom of the Notepad window displays "length: 250 lines: 8 Ln: 8 Col: 1 Sel: 0", "Dos\Windows", "ANSI", and "INS".



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:



The screenshot shows a window titled "Python Shell" with a menu bar containing "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main area displays the following text:

```
Python 3.1.3 (r313:86834, Nov 27 2010, 18:30:53) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ("Hello World")
Hello World
>>> 2+3
5
>>> 8/4
2.0
>>> |
```

A callout box on the right side of the window contains the text: "The Python IDLE environment." The status bar at the bottom right of the window shows "Ln: 9 | Col: 4".



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 provides a good development environment.
- The next page illustrates the Python perspective in Eclipse.



Pydev - Python Project/src/HelloWorld.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

DisplayBikes.java ServerTest.java ClientTest.java SimpleJdbc.java HelloWorld

```
'''
Created on Mar 31, 2011

@author: Mark Llewellyn
'''
print ('Hello World');
print (2+3)
print (8/4)
```

Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\HelloWorld.py

Hello World
5
2.0

Writable Insert 9:1

The Eclipse Python perspective.



Python Basics

- Now that you've set-up Python and tried out the various ways to run Python, its 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.



Python Basics

- 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 *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`.



Pydev Package Explorer

- Morse Code Project
- Program Four
- Program Four - CNT 4714 - Fall 2010
- Program One
- Program One - CNT 4714 - Fall 2010
- Program Two
 - bin
 - src
 - SQL_Client_GUI.java
- Python Project
 - src
 - basicScript2.py
 - HelloWorld.py
 - Python31 (C:\Python31\python.e
- Servlets
- Socket File Server
 - bin
 - src
 - Client.java
 - ClientTest.java
 - Server.java
 - ServerTest.java
- temp
- Threading Examples
- TicTacToe
- WaterTreatmentPlant

```

Created on Mar 31, 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 < 5)

```

Problems Console

```

<terminated> C:\Users\Lei Wei\
1. 1
2. 4
3. True
4. False
5. False
6. True
7. True

```



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 is 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.



Pydev - Python Project/src/stringExamples.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

DisplayBikes.java SimpleJdbc.java HelloWorld basicScript2 stringExamples

```
@author: Mark Llewellyn
'''
string1 = "Python is neat!";
string2 = "Python is cool!";
#index 0 in string1 is character "P"
print("1. ", string1[0]);
#slice of string1 starting at position 10 and ending
#at position 14 is = "neat"
print("2. ", string1[10:14]);
#index last position in string1 is character "!"
print("3. ", string1[-1]);
#concatenate the two strings together
print("4. ", string1 + string2);
print("5. ", string1 + " " + string2);
#repetition operator
print("6. ", string1 * 3);
```

Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\stringExamples.py

```
1. P
2. neat
3. !
4. Python is neat!Python is cool!
5. Python is neat! Python is cool!
6. Python is neat!Python is neat!Python is neat!
```

Writable Insert 19:27



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.



Pydev - Python Project/src/listExample.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

SimpleJdbc.java HelloWorld basicScript2 stringExamples *listExample x 6

```
#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;
```

Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\listExample.py

```
[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)
```

Writable Insert 19:55



Python Basics - Dictionaries

- 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.



Pydev - Python Project/src/dictExample.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

basicScript2 stringExamples listExample dictExample

```
@author: Mark Llewellyn
'''
#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");
for key in aDict:
    print(key, " = ", aDict[key]);
```

Problems Console

```
<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\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
```

Writable Insert 14 :



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:

```
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:

```
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:

```
if expression1:
```

```
    if_suite
```

```
elif expression2:
```

```
    elif_suite
```

```
else:
```

```
    else_suite
```



Python Basics – `while` loop

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

```
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.



Pydev - Python Project/src/whileLoopExample.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

listExample dictExample whileLoopExample x 10

```
'''
Created on Apr 4, 2011

@author: Mark Llewellyn
'''
# while loop example
counter = 0;
while counter < 3:
    print('Loop #%d' % (counter))
    counter += 1
```

Problems Console x

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\whileLoopEx

```
Loop #0
Loop #1
Loop #2
```

Writable Insert 10 : 17



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.



Pydev - Python Project/src/forLoopExample1.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

listExample dictExample whileLoopExample forLoopExample1

```
'''
Created on Apr 4, 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)
```

Problems Console

```
<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\forLoopExample1.py
I like to use the Internet for:
e-mail
net-surfing
news
shopping
```

Writable Insert



Pydev - Python Project/src/forLoopExample2.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

whileLoopExample forLoopExample1 forLoopExample2 >>12

```
'''
Created on Apr 4, 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=" ")
```

Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\forLoopExample2.py
I like to use the Internet for: e-mail net-surfing news shopping

Writable Insert



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.



Pydev - Python Project/src/forLoopExample3.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

Pydev Java Browsing

forLoopExample1 forLoopExample2 forLoopExample3 13

```
'''
Created on Apr 4, 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=" ")
```

Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\forLoopExample3.py

```
0 1 2 3 4

4 3 1 5 6 7 9 4 5
```

Writable Insert



Pydev - Python Project/src/forLoopExample4.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

Pydev Java Browsing

forLoopExample2 forLoopExample3 forLoopExample4 »14

```
'''
Created on Apr 4, 2011

@author: Mark Llewellyn
'''
#for loop example 4
for eachNum in range(6):
    print (eachNum)
print (" \n")
```

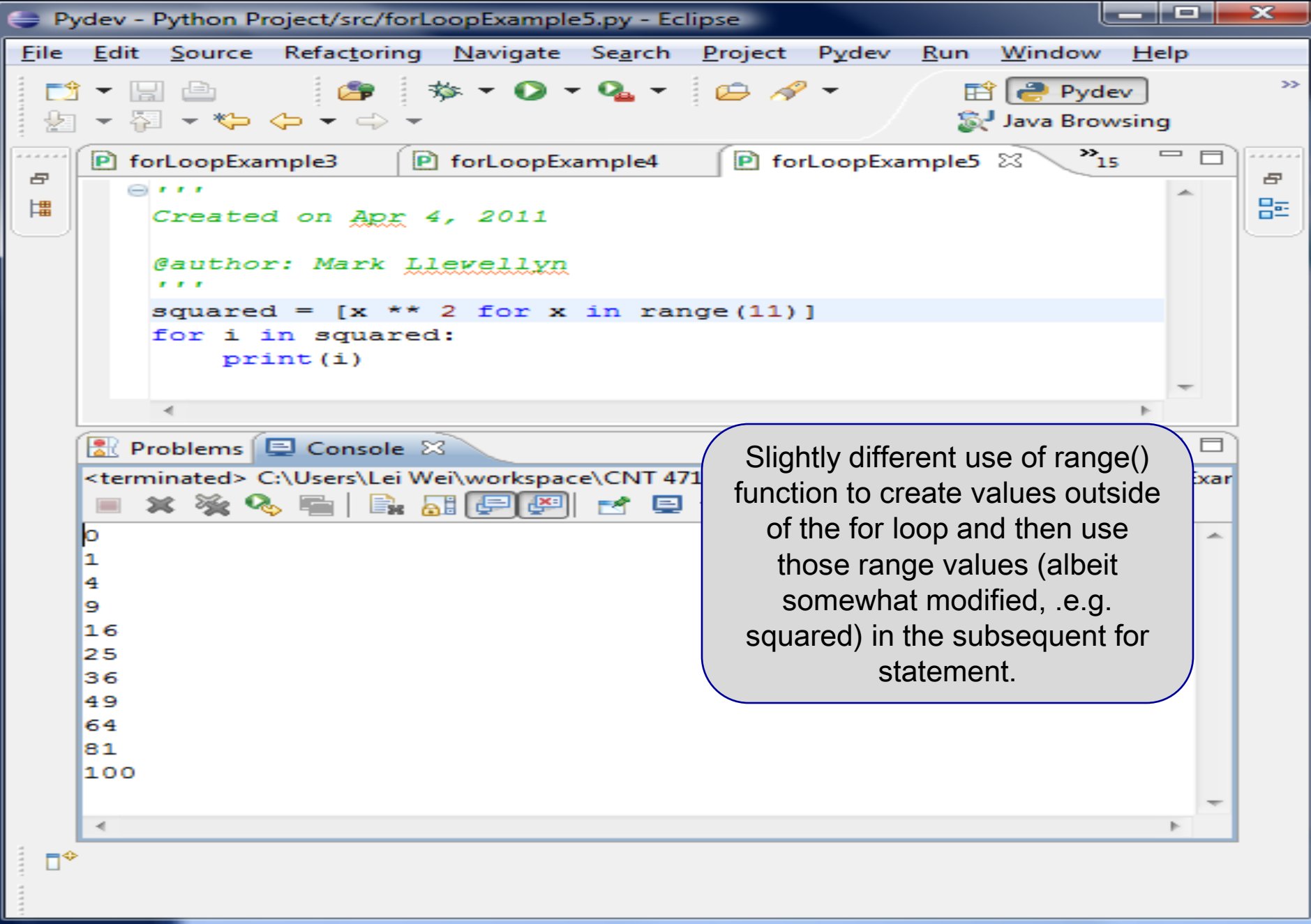
Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\forLoopExamp

```
0
1
2
3
4
5
```

Writable Insert 8 : 19





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:

```
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.



Pydev - Python Project/src/openFileExample.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

forLoopExample2 forLoopExample3 forLoopExample4 forLoopExample5 openFileExample 14

```
'''
Created on Apr 4, 2011

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

Problems Console

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\openFileExample.py

This is the text file that will be opened

by the Python openFileExample text program.

This file was created on 4/4/2011.

C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\rawte...

File Edit Search View Encoding Language Settings Macro Run TextFX
Plugins Window ?

prog4rootcommands.sql xhtml strict template.html rawtext.txt

```
1 This is the text file that will be opened
2 by the Python openFileExample text program.
3 This file was created on 4/4/2011.
```

ler Ln:1 Col:1 Sel:0 Dos\Windows ANSI INS



Pydev - Python Project/src/openFileExample2.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

forLoopExample3 forLoopExample4 forLoopExample5 openFileExample openFileExample2 x 15

```
'''
Created on Apr 5, 2011

@author: Mark Llewellyn
'''
#fileName = input('Please enter the file name:')
#second version of file open script that reads all lines in the file at once
fileObject = open('rawtext.txt', 'r')
allLines = fileObject.readlines()
fileObject.close()
for eachLine in allLines:
    print(eachLine)
```

Problems Console x

```
<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\openFileExample2.py
This is the text file that will be opened
by the Python openFileExample text program.
This file was created on 4/4/2011.
```

Writable Insert 1:4

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.





```

'''
Created on Apr 11, 2011

@author: Mark Llewellyn
'''
import os
#fileName = input('Enter a file name: ')
fileObj = open("output.txt", 'w')
while True:
    aLine = input("enter a line ('#' to quit): ")
    if aLine != "#\r":
        #fileObj.write('%s%s' % (aLine, os.linesep)) #old style
        fileObj.write('%s' % aLine)
    else:
        break
fileObj.close()

```



```

1 This file was created 4/11/2011 by MJL
2
3 Text line 1.
4 Text line 2.
5 Text line 3.
6

```

```

<terminated> C:\Users\Lei Wei\workspace\CNT 4714 - Fall 2009\Python Project\src\writingAFileExample.py
enter a line ('#' to quit): This file was created 4/11/2011 by MJL
enter a line ('#' to quit):
enter a line ('#' to quit): Text line 1.
enter a line ('#' to quit): Text line 2.
enter a line ('#' to quit): Text line 3.
enter a line ('#' to quit): #

```

