Intro to Python Part 2

P. TenHoopen - WMLUG

What is Python?

Python is a cross-platform object-oriented programming language invented by Guido van Rossum. It is an interpreted language but there is support for compiling the programs.

The most recent version is 3.0 (also known as Python 3000) which was released on December 3, 2008. It is incompatible with the 2.x releases.

- Python Home Page - http://www.python.org/
- Beginners Guide - http://wiki.python.org/moin/BeginnersGuide/NonProgrammers
- Python Language Reference - http://docs.python.org/reference/index.html#reference-index
- Guido van Rossum Personal Page - http://www.python.org/~guido/

Native Data Types

Native Python Data Types:

- Dictionaries
- Lists
- Tuples

Dictionaries

Dictionaries are an unordered collection of key-value data pairs.

It is unordered in the fact that as you add items, they are not necessarily appended to the end.

Keys need to be unique.

Creating Dictionaries

Creating an empty dictionary:

```
name={} # empty dictionary
```

Example:

```
colors={}
```

Creating Dictionaries

Creating a pre-popluated dictionary:

```
name={"key":"value"}
```

Example:

```
hexColor={"white":"ffffff", "black":"000000",
"red":"ff0000"} # prepopulated dictionary

print hexColor
{'white': 'ffffff', 'black': '000000', 'red':
'ff0000'}
```

Adding Items

Adding items to a dictionary:

```
name["key"]="value"

hexColor["blue"]="0000ff"

print hexColor
{'blue': '0000ff', 'white': 'ffffff',
'black': '000000', 'red': 'ff0000'}
```

Notice that the item was inserted at the beginning, and not the end – hence unordered!

Changing Item Values

Changing values in a dictionary:

Same syntax as adding but the value is updated.

```
hexColor["blue"]="blue"

print hexColor
{'blue': 'blue', 'white': 'ffffff',
'black': '000000', 'red': 'ff0000'}
```

Deleting Items

Deleting items from a dictionary:

```
del name["key"]
```

Example:

```
del hexColor["blue"]

print hexColor
{'white': 'ffffff', 'black':
'000000', 'red': 'ff0000'}
```

Clearing All Items

Clearing all items from a dictionary:

```
name.clear()
```

Example:

```
hexColor.clear()
print hexColor
{}
```

Lists

Lists are an ordered collection of indexed values.

The values can be strings or numbers and can be duplicated.

List indexes are zero-based.

Creating New Lists

Creating new lists:

```
name[] # empty list
```

Example:

```
list1[]
```

Creating New Lists

Creating a prepopulated list:

```
list2["white", "black", "42"]

print list2
['white', 'black', '42']
```

Insert Item Into List

Inserting items into a list:

```
list2.insert(2, "purple")

print list2
['white', 'black', 'purple', '42']
```

Appending Items

Appending items to a list:

```
list2.append("red")

print list2
['white', 'black', 'purple', '42',
'red']
```

Deleting Items

Deleting items from a list:

```
list2.remove("white")
```

This will remove the first occurance of the value "white" from the list.

```
print list2
['black', 'purple', '42', 'red']
```

Using List Values

Using values from a list:

`name[index]`

Example:

List2[1]
'purple'

Searching Lists

Searching for values in a list:

```
name.index("key")
```

Example:

```
list2.index("42")
2
```

Check A List For Values

If you try to search a list for a value that isn't present, python will give an exception.

To avoid this, you can check a list for a certain value:

```
"black" in list2
True
```

You get a True or False response back.

List

You can use the len function to get the number of items in a list:

```
len(list2)
4
```

Tuples

A tuple is an indexed immutable list.

It can't be changed once it is created.

Good for constant variables.

Why Use Tuples

Tuples are faster than lists.

Lists can be turned into tuples using the `tuple` function.

Tuples can be turned into lists using the `list` function.

Creating Tuples

Creating a new tuple:

```
name=(“value1”, “value2”)
```

Examples:

```
t=("a", "b", "c")
screensize=(800, 600)
```

Checking A Tuple for Value

You can check a tuple for a certain value:

```
"b" in t
True
```

You get a True or False response back.

Functions

Functions are blocks of code that perform a certain function.

The statements are indented inside the function.

Defining A Function

To define a function use the `def` command:

```
def functionName(parameters):
```

The parameters are optional variable names.

Documenting

You can document functions use a doc string.

A doc string is a triple quotation mark enclosed string. It can be multi-lined.

Example:

```
def function():
  """This is a doc string."""
```

Return Values

You can return a value back from the function using the `return` command. If that is not done, the function automatically returns `None` which is the Python null value.

```
def functionName(parameters):
  print “Hello from the function”
  return 1
```

Variable Scope

The scope of a variable is where the variable is accessible. There is local and global scope.

A variable defined within a function only is accessible within that function, hence it has local scope.

A variable can have global scope so that it is accessible to all functions.

Using Global Variables

Sometimes you need to use variables that were declared outside of the function which are normally inaccessible.

Use the global command to access the values.

```
global variableA
```

Using Functions

To call a function:

```
def sayHello():
  """Function to print a message."""
  print "Hello, how are you?"
  return 1

sayHello()

Hello, how are you?
```

Function Parameters

To call a function that takes a parameter:

```
def sayHello(passedName):
    """Function to print a message.
    The passedName variable accepts the value passed to it from
    the calling statement.  It has local scope in the function.
    """
    print "Hello, %s how are you?" % passedName
```

Example 1 – Static string as a parameter:

```
sayHello("Homer")
Hello, Homer how are you?
```

Example 2 – Variable as a parameter:

```
myName="Bart"
sayHello(myName)
Hello, Bart how are you?
```

Putting It All Together

Demo

Putting It All Together

```
#!/usr/bin/python
"""
FileStats.py
P. TenHoopen, 11/21/08

Read the statistics of files from a specified directory and subdirectories.
Display number of files, types and number of files, average size, smallest size,
largest size
"""

# Import Modules
import os

# Global Variables
extensions = {}          # dictionary to hold extensions and counts
largestFileSz = 0        # size of largest file
largestFile = ""         # name of largest file
numFiles = 0             # number of files found
sumFileSizes = 0.0       # total size of files used to calc average file size
```

Putting It All Together

```
# Functions

def recordExtension(thisExtension):
    """This function keeps track of the extensions that are found."""

    # add to dictionary if new, or update count if existing
    if thisExtension not in extensions:
        extensions[thisExtension] = 1
    else:
        extensions[thisExtension] += 1

    return 1
```

Putting It All Together

```
def collectFileInfo(fileName):
    """This function examines file statistics to keep track of largest file and
total file sizes."""

    # use global variables defined above
    global largestFileSz
    global largestFile
    global sumFileSizes

    # collect file statistics
    fileStats = os.stat(fileName)
    # get file size
    size = fileStats.st_size
    # keep size of all files
    sumFileSizes += size
    # is this the largest file found so far?
    if size > largestFileSz:
        largestFileSz = size
        largestFile = fileName

    return 1
```

Putting It All Together

```
def displayStats():
    """This function displays the collected statistics."""

    print "\n\nResults:"
    print "\nThere were %d files found." % numFiles
    print "\nCount \t File Ext"
    print "----- \t --------"
    for extension in extensions:
        print "%d \t %s" % (extensions[extension], extension)
    print "\n"
    print "Largest file:  %s at %d bytes" % (largestFile, largestFileSz)
    print "Total size of files is %d bytes" % (sumFileSizes)
    print "Average size of files is %.2f bytes" % (sumFileSizes / numFiles)
    print "\n"

    return 1
```

Putting It All Together

```
def main():
    """This function is called when the program starts.  It analyzes the
files specified in the topDir variable."""

    # variables
    topDir = "."   # directory to search, default to current directory

    # use global variables defined above
    global numFiles

    # get top directory to analyze from user
    userDir = raw_input("Input the directory to analyze: ")
    if (len(userDir) > 0):  # weak error checking!
        topDir = userDir    # as long as the user enters something, use
it otherwise use default value of "."

    print "\nAnalyzing files in '%s'..." % topDir

    # walk through the files
    walklist = os.walk(topDir)

    # os.walk returns {dirpath, dirnames, filenames}
```

Putting It All Together

```
    for root, dirs, files in walklist:
        # examine each file found
        for eachFile in files:
            numFiles += 1
            # look for a extension delimiter (.) in the file name
            if "." in eachFile:
                # get the file extension using a string slice
                # extract all text after the period
                extension = eachFile[eachFile.index(".")+1:]
            else:
                # no period delimiter so no extension can be determined
                extension = "none"

            # keep track of extensions that are found
            recordExtension(extension)

            # get file name and path
            fileName = os.path.join(root, eachFile)

            # examine files stats
            collectFileInfo(fileName)

    # display the results of the analysis
    displayStats()

    Return 1

# This calls the 'main' function when this script is executed.
if __name__ == '__main__': main()
```

That's All Folks

Questions?