Introduction to Python

2017

Objectives

- Introduction to Python
- Hands-on
  - Syntax
  - Strings
  - Lists
  - Control Flow & Loops
  - Functions
- Import & Future Topics
- Helpful hints & resources

What is Python?

- Is it a Scripting language?
- An Object-oriented language maybe?
- How about an Interpreted language?

Python is

- General purpose, object oriented, high level, interpreted language
- Simple,Portable,Open source & Powerful
- Developed in early 90's by Guido Van Rossum

Python vs Java

- Dynamic vs Static Typing - Handling variables
- Indentation vs Braces - Separate code into blocks
- Terse vs Verbose - Number of lines of code
- Easy to learn vs Steep learning curve
- Use Python + Java

where is Python used

- Used extensively in web - Django, TurboGears, Plone, etc
- Communicating with Databases - MySQL, Oracle, MongoDB, etc
- Desktop GUI - GTK+, QT, TK, etc
- Scientific computing - SciPy, Scientific Python, etc
- Software Development - SCons, Buildbot, Roundup, etc
- Games 3D graphics - Pygame, PyKyra, etc
- For success stories - https://www.python.org/about/success

Download & Installation

- Latest releases - Python 2.7.9 & Python 3.4.3
- Python Interpreter
  - Linux - Installed in all distro's, else https://www.python.org/downloads/source/
  - Mac - Installed, else https://www.python.org/downloads/mac-osx/
  - Windows - Download from https://www.python.org/downloads/
- Text Editor
  - Linux - Vim, Gedit, Kate, Emacs, etc
  - Mac - Vim, TextMate or any unix like editor
  - Windows - Notepad++ or IDE
- Python Documentation - https://www.python.org/doc/
- Set editor to expand tab to 4 spaces

Ways to run python

- Interactive Interpreter (Classic command line interpreter or Python Shell) - Run as a calculator or used to run a python program
- Script from command line - Used to run a python program
- Integrated Development Environment (IDE) - Eclipse, Netbeans, etc

Python programs

- Python programs and modules are written as text files with .py extension
- Program - .py file executed directly as program (often referred to as scripts)
- Module - .py file referenced via the import statement
- Write a simple print program and save as 'myFirst.py' and save on desktop

  print "Hello, Python!"
- Run Python Shell 'C:\path..\myFirst.py'

Variables & Data types

- Variables
  - Stores a piece of data and gives it a specific name
  - Do not need to specify data type
  - Equal sign (=) is used to assign values to variables
- Data Types
  - Numbers - int,long,float,complex
  - Boolean - True, False
  - String
  - List
  - Tuple
  - Dictionary

Indentation,Comments & Math operators

- Indentation
  - Whitespace is used to structure the code instead of braces
  - A tab - 4 spaces
- Comments
  - Single line - #
  - Multi-line - Not supported in Python but people use Triple quote.
- Math operators
  - Addition (+),Substraction(-),Multiplication(*),Division(/)
  - Modulus(%),Exponent(**),Floor Division (//)
- Exercise 1 (Run as calculator and a saved .py program)

Tip Calculator!!

Cost of meal = 54.76

Tax = 7.85%

Tip = 15%

Find cost after tax, tip, cost after tip, total cost

Strings

- Can be declared in single quotes (’ ’) or double quotes (” ”) or triple quotes (””” ””” or ''' ''')
  singleQuote = 'One Line'
  doubleQuote = ”One Line with ' eg. Jame's”
  tripleQuote = """Span
  multiple lines"""
- Each character can be accessed by using their index. Index starts from zero(0)
- Consider ”Python”
  PYTHON
  P will be at 0^{th} position of string and N will be at 5^{th} position from the start
- Strings are ”immutable”
- Immutable means that we cannot change the value. If we have an instance of the String class, any method you call which seems to modify the value, will actually create a new string.

String Methods

Let our string variable be called 'var'

- Length of string - len(var)
- Convert to string - str(var)
- Convert to lowercase - var.lower()
- Convert to uppercase - var.upper()
- Is digit/Is alpha - var.isdigit()/var.isalpha()
- Replace characters - var.replace(old,new)
- Split a sentence - var.split(delimiter)
- Swap case - var.swapcase()
- Range slice - var[start index:end index]

String Concatenation Formatting

- Concatenation
  - Combining of strings is done by using the (+) operator between them
  - In order to combine a string with a non-string variable, use str() method to convert non-strings to strings.
- Formatting
  The string format operator is %
  - %c - character
  - %s - string conversion via str() prior to formatting
  - %d - signed decimal integer
  - %x %X - hexadecimal integer(lowercase/uppercase)
  - %f - floating point real number

Lists

- It is a datatype you can use to store a collection of different pieces of information as a sequence under a single variable name
- Can be accessed by index
- Creating lists - Putting different comma separated values within square brackets
  list1 = [”physics”,”astronomy”,56.98,”MJ”,-9.36]
- Accessing values - Individual elements or a range of elements
  list1[3];list1[-2];list1[1:4];list1[-3:]
- Updating lists - Update single or multiple entries in a list
  list1[2] = ”Botany 101”
- Negative index represents access from the right starting from -1
  list1[-2] = ”MJ”

A tad bit more

- Deleting elements - Either use del statement or remove() method
  list1 = ["physics","astronomy",56.98,"MJ",-9.36]
  del list1[3] - when you know the index of the element
  list1.remove("physics") - when you don't know the index of the element
- Other List operations
  - Length - len([3,4,5,"supernova","jaguar"])
  - Concatenation - [1,2,3] + [5,6,7] = [1,2,3,5,6,7]
  - Repetition-
    ["hi","bye"]*3 = ["hi","bye","hi","bye","hi","bye"]
  - Membership(returns Boolean) - 56.98 in list1

List Methods

Let our list variable be called 'list1'

- Length of the list - len(list1)
- Maximum/Minimum value - max(list1)/min(list1)
- Append object to list - list1.append(obj)
- Frequency of object - list1.count(obj)
- Return index - list1.index(obj)
- Insert object - list1.insert(index,obj)
- Delete object - list1.pop()/list1.remove(obj)
- Reverse the list - list1.reverse()
- Sort the list(natural order) - list1.sort()

Tuples

Tuples are sequences, just like lists except.

- Tuples are immutable - cannot be changed or updated unlike lists
- Tuples use parentheses (), whereas lists use square brackets []

Dictionary

- Similar to a list by values are accessed by looking up a key instead of an index
- A key can be a string or number
- Creating dictionaries -key-value pairs are separated by (:), items are separated by (,) and everything is enclosed in curly braces

```
dict1 = {"name":"Daniel","age":23,"degree":"MS"}
dict2 = {'name':"Ian Callum",'age':60,
'job':"Car designer",
'brand':"Jaguar",
'worked-for':["Ford","TWR","Aston Martin"]}
```

Some more

- Accessing values - Values can be accessed only through keys
  dict1['age'] - 23
  dict2['brand'] - Jaguar
- Updating dictionary - Add new entry or modify an existing entry
  dict1['subjects'] = ["OS","DBMS","Artificial Intelligence"]
  dict2['worked-for'][1]="Tom Walkinshaw Racing"
- Deleting dictionary elements - 3 variations
  dict1 = {"Sayuri":556-2365,"ken":556-8749, "Tom":556-5800}
  del dict['Sayuri'] - Removes entry with key 'Sayuri'
  dict.clear() - Removes all entries in the dictionary
  del dict - Deletes entire dictionary

Dictionary Methods

Let our dictionary variable be called 'dict1'

- Length of the dictionary - len(dict1)
- Shallow copy - dict1.copy()
- For key,return value - dict1.get(key)
- Check for key (returns boolean) - dict1.has_key(key)
- List of k-v pairs - dict1.items()
- List of keys - dict1.keys()
- List of values - dict1.values()

Control Flow

- Comparison Operators
  - Equal to (==)
  - Not equal to (!=)
  - Less than ($<$)
  - greater than ($>$)
  - greater than or equal to ($\geq$)
  - less than or equal to ($\leq$)
- Logical Operators
  - Logical AND (and)
  - Logical OR (or)
  - Logical NOT (not)
  - Note: Precedence:- not $>$and $>$or

If and else loop

- If statement - Consists of a Boolean expression followed by one or more statements

```
if<condition >:
  statement(s)
```

- If...else statement-If statement followed by an optional else statement which executes when the Boolean expression is false

```
if<condition >:
  statement(s)
else :
  statement(s)
```

If,elif and else loop

- If...elif...else statement Otherwise if the following expression is true, do this!

```
if <condition >:
  statement(s)
elif <condition >:
  statement(s)
else:
  statement(s)
```

Exercise 2

Program to generate 6+2 UC id for a given name.(6+2 ID takes the first 6 letters from the last name & first and last letters from the first name)

- Store the first and last names in two different variables.
- Check if the length of the last name is <6 or not.
  If < 6, pick first letters of first name to make up for the length the last name.
- Accordingly, make use of slicing and concatenate the letters to give the 6+2 ID.

Hint for taking input

```
Python 2 -> var =input('enter value') # will convert and return number
            var =raw_input('enter value') # will return string
Python 3 -> var =input('enter value') # will return string
            var =raw_input('enter value') # not available
```

While loop

- While loop will execute as long as the looping condition is satisfied or True.

```
while <loop_condition>:
  statement(s)
```

- Infinite loop-Occurs when
  - Loop condition cannot possibly be wrong (while 1!=2:)
  - Logic of the loop prevents the loop condition from becoming false

```
count = 10
while count>0 :
  count += 1
```

While loop

- Break statement - One liner which means, ”exit the current loop”

```
count = 0
while True :
  Print count
  count +=1
  if count >= 10:
    break
```

While/else loop

- A feature very unique to Python
- The 'else' block executes when either
  - Loop is never entered
  - Loop exits normally
- The 'else' block does not execute when, the loop exits as a result of a break

```
import random #will explain import in latter slide
count = 0
while count<3:
    num = random.randint(1,6)
    print num
    if num == 5:
        print "sorry, you lose!"
        break
    count +=1
else:
    print "You win!"
```

For loop

- Each item in the sequence is assigned to the iterating variable and the statements are executed until the entire sequence is exhausted
  for <iterating_var>in <sequence>:
  statement(s)
- Iterating over a range of values

  -
    ```
    for letter in 'string':
      print letter
    ```

  -
    ```
    for num in range(10):
      print num
    ```

  -
    ```
    for num in range(-6,6):
      print num
    ```

  -
    ```
    for num in range(-10,-100,-30):
      print num
    ```

  -
    ```
    for fruit in ['bannana', 'orange']:
      print fruit
    ```

More Iterations

- Iterating a string
```
for letter in "Monty Python!" :
  print "Current letter:", letter
```
- Iterating a list
```
list1 = ["F-type","C-X75","XJ-13"]
for index in range(len(list1)):
  print "current model: ",list1[index]
```
- Iterating a dictionary
```
dict1 = {'name':"Bob",'age':34,'dob':"6-25-1990" }
for keys in dict1.keys():
  print "Value of "+keys+" :%s" %dict1[keys]
```

For/else loop

- A feature very unique to Python just like while /else
- The 'else' block executes when the loop exits normally
- Does not execute when, the loop exits as a result of a break

```
fruits = ['banana','apple','orange','tomato','pear']
print 'You have...'
for f in fruits:
  if f == 'tomato':
    print 'A tomato is not a fruit!'
    break
  print 'A',f
else:
   print 'A fine selection of fruits!'
```

Exercise 3
DNA Transcription - The process of converting DNA to RNA
Example :
G->C;C->G;A->T;T->A
Input: GCTAGCCTACG
Output: CGATCGGATGC
Hint: Use "for" loop to traverse the DNA string. Use "if...else" construct to check and replace each letter in the string.

Function Definition

- It is a block of organized, reusable code that is used to perform an action
- Defining a Function
  - Begins with keyword 'def' followed by the function name and parentheses
  - Any input parameter/arguments should be placed within these parentheses
  - Code block starts with colon (:) and is indented
  - return [expression] statement exits a function, optionally passing back an expression to the caller

```
def function_name(parameters):
  "function docstring"
  function suite
  return [expression]
```

Calling a Function

- After the basic structure is finalized, we can execute by either
  - Calling it from another function
  - Calling it directly from command prompt
- Called function

```
def printme(str):
   "This prints a string passed into the function"
   print str
   return :

#Function call

printme('First call to user defined function')
printme('Second call to user defined function')
```

Importing Modules

- Module is a file that contains definitions - including variables and functions - that you can use once it is imported
- Generic import - Import only a module
  Syntax: import module_1[,module_2[,...module_N]
  Usage: import math,support
- Function import-Import specific attributes from a module
  Syntax: from module_name import name_1[,name_2[,....name_N]
  Usage: from math import sqrt,pow

More Importing

- Universal import - Import all variables and functions in a module
  Syntax: from module_name import *
  Usage: import math import *
- dir() Function - Return a sorted list of strings containing names defined in a module
  import math
  content = dir(math)
  print content

Advanced Topics

- File input&output
- Exceptions
- Classes&Objects
- Regular Expressions
- Database Access
- Multithreading

Helpful hints&resources

- CEAS Library Python resources
  - http://guides.libraries.uc.edu/python
- Online links & tutorials
  - Python documentation - https://www.python.org/doc/
  - Python Programming wiki book - http://en.wikibooks.org/wiki/Python_Programming
  - Python tutorials - Udemy, Code academy, etc

Questions ??

Survey

https://www.surveymonkey.com/r/python-Feb22

Pick one of the solution manual

Thank you for attending the workshop !!
Your kind suggestions/feedbacks are more than welcome