Python 3

*CIS*2750*

Advanced Programming Concepts

Material for this lecture was developed by Dr. D. Calvert.

What is Python?

- Python is an *object-oriented* scripting language developed by Guido van Rossum.
- It supports polymorphism, operator overloading and multiple inheritance.
- It is easy to bind to other languages:
  - **Extend**
    - call components written in C or C++
  - **Embed**
    - call Python from C or C++
  - **Jython** is an implementation written in Java and can be used to link with Java

What is Python?

- It is portable and available on most platforms including Unix/Linux, Windows, Mac OS, others.
- There exist many libraries for Python which support complex programming tasks including networking, GUIs, object serialization, and complex math.
- It is interpreted which allows for a rapid development cycle.
- **Version 3** *not* fully backward compatible with 2

Basic Syntax

- There is no end-of-statement character other than the **newline** (carriage return).
  - Type a backslash to continue a long \ statement.
- Blocks of code are identified using **indentation** and not a begin-end or brace {} pair.
  - *Consistent* indentation is important or the interpreter will not work properly.
  - Care must be taken when *mixing* tabs and spaces.
- Comments begin with the # sign (pound or hash).
  - Everything to the end-of-line is ignored.

Basic Syntax

- The script should contain a very specific first line which tells the shell (e.g. bash) what program will interpret the script.

  #!/usr/bin/python3

  – #! tells the shell to run the program which follows.
  – Other scripting languages use other interpreters:

  #!/usr/bin/perl

  #!/bin/sh

- To execute without typing “python”, the script must be made executable:

  chmod +x <filename>

Variables

- Python has built-in support for numbers and strings as well as several more complex data structures.
- Normal integer (**int**) and floating-point (**float**) numbers are supported.

  3, 3.0, 3.14e-10

Variables

- **int** type has “arbitrary precision”!
  - no max. value, uses as much memory as needed
- Binary, octal and hexadecimal numbers (**int** type)

  0b10110110, 0o270, 0xbeef
- Complex numbers (**complex** type)

  3 + 4j

Variables

Strings using either single or double quotes

- s1 = “fred”
- s2 = ‘fred’
- s3 = “fred’s”
- s4 = s1 + s2 # concatenation
- print (s4) # outputs ‘fredfred’
- print (s4[2]) # outputs ‘e’
- print (s4[2:4]) # “slice” of string outputs ‘ed’
- print (len(s4)) # outputs 8 = string length

Complex Structures

Python supports several other complex data structures:

- Lists
- Dictionaries
- Tuples

Lists

- Lists are **ordered** collections of other objects.
- They can contain numbers, strings, or other lists (i.e., *heterogeneous* data).
- They are accessed using an **index**.
- They have a *variable length* **-** can grow and shrink in place.

List Example

```
#!/usr/bin/python3      ← on cis2750 "python" is V2 not 3!
L1 = ["abc", 123, "fred"]
print (L1[1] )          # outputs 123
print (L1[0:2])         # "slice" of L1 outputs ['abc', 123]
print (len(L1))         # outputs 3
for x in L1:
    print (x)           #outputs abc, 123, fred each on a
                           separate line
L1.append("betty")      # adds betty to the list
print (L1)              # outputs ['abc', 123, 'fred', 'betty']
```

Dictionaries

- Dictionaries are **unordered** collections of objects.
- They are identified using a **key** instead of an index.
- They are similar to arrays **-** often called *associative (or associate) arrays*.

Dictionaries Example

```
#!/usr/bin/python3
table = {"nut" : 1.0,                    # due to {…} no need for \
         "bolt": 2.0,
         "widget": 0.5,
         "gadget": 1.0}
print (table["nut"])                      # references element through
                                          # key, output is 1.0
print (len(table))                        # 4
print ("widget" in table)                 # True
print (table.keys())                      # dict_keys(['gadget', 'bolt', 'nut',
                                          #                 'widget'])
```

Tuples

- Tuples are exactly like lists but they are **immutable** (value cannot be changed in place).

aTuple = (1, “a”, 6)

- To change the value you must create a new tuple.
- They are useful if you want to guarantee that some data will not be changed accidentally.

Some Coding Principles

- If you use the **.py** naming convention, file foo.py is considered to contain a *module* named foo.
- *Importing* a module executes its top-level code (like calling it) and makes its functions known
- A module's *name* is available in the built-in **__name__** variable:
  - The initial file being executed is named **__main__.**
  - But an *imported* module retains its own module name.

Module Example

foo.py

```
#!/usr/bin/python3
print ("Hi, my name is " + __name__)
```

python3 foo.py

Hi, my name is __main__

python3

>>>import foo

Hi, my name is foo

Works because python looks for *module*.py

Control Structures

- The if, else, and for statements end with a colon and the code under them is indented to show which lines constitute the control block.

```
string1 = "barney"
for c in string1:
    print (c, end="")   # Don’t append new line
print()                 #  Just print new line
```

Control Structures

```
if string1 == “barney”:
    print (“rubble”)
elif string1== “dinosaur”:
    print (“I love you, you love me…”)
else:
    print (“flintstone”)
```

The range() function

```
for i in range(5) :
    print (i)
```

- The range() function returns a list of incrementing integers which count to one less than the parameter value.
  - range(3) returns [0,1,2]

Going on with Python 3

- Lab session will take up Python
- By this point, you've been exposed to enough programming languages (procedural and object-oriented) that you can continue learning Python 3 *on your own*
  - Recommended textbook
  - Lots of websites, especially python.org
    - Be sure you're studying "3" not old version "2"!