CS 102 Unit 15

Python

Mark Redekopp
(Optional – Instructor may skip due to time constraints)

PROGRAMMING LANGUAGES
Recall that all computer programs must be converted to 1's and 0's (aka machine code).

Similar to translating from one spoken language to another.

Imagine you need to give a speech in front of a crowd that does not speak your native language. How could you do it?
Compiled vs. Interpreted Languages

Compiled (Natively)
- Requires code to be converted to the native machine language of the processor in the target system before it can be run
- Analogy: Taking a speech and translating it to a different language ahead of time so the speaker can just read it
- Faster
- Often allows programmer closer access to the hardware

Interpreted
- Requires an interpreter program on the target system that will interpret the program source code command by command to the native system at run-time
- Analogy: Speaking through an interpreter where the speaker waits while the translator interprets
- Better portability to different systems
- Often abstracts HW functionality with built-in libraries (networking, file I/O, math routines, etc.)

https://www.youtube.com/watch?v=qaj7nO1HUqA
Best of Both Worlds?

- Many languages used for web and desktop apps (e.g. Java and Python) will compile their code to an intermediate form (aka bytecode)
  - Then an interpreter can be used to execute the byte code faster than interpreting the high-level language directly
  - New interpreters can be provided for new devices (platforms)

- Other languages like C/C++ compile their code directly to a form that can be executed and run on the device
A Live Demo

• Sort an array of integers from N-1 to 0
  – [9,999 9,998 9,997 ... 3 2 1] =>
  – [1 2 3 ... 9,997 9,998 9,999]

• With a Python script (interpreted)

• With C++ (compiled natively)

• With a "built-in" Python library function that does the same task we just wrote manually (different algorithm)
  – a = range(N)
  – a.reverse()
  – a.sort() // built-in sort implementation (non-interpreted)

• Note: Algorithms can make all the difference!
Credits

• Many of the examples below are taken from the online Python tutorial at:
  – http://docs.python.org/tutorial/introduction.html
Python in Context

• Two major versions with some language differences
  – Python 2.x
  – Python 3.x (we will focus on this version)

• Interpreted, not compiled like C++
  – Can type in single commands at a time and have them execute in "real time"
  – Somewhat slower
  – Better protection (no memory faults)
Interactive vs. Scripts

• Can invoke python and work interactively
  – % python    #python 2.x
  – % python3   #python 3.x
    >>> print("Hello World")
    Ctrl-D (Linux/Mac) [Ctrl-Z Windows] at the prompt will exit.

• Can write code into a text file and execute that file as a script
  – % python3 myscript.py

```python
# python2.x
>>> print "Hello world"

# python3.x
>>> print("Hello world")
```

myscript.py
Types

• Types
  – Bool: True/False (not true/false)
  – Integers
    • Integer division => see examples
  – Floats
  – Complex
  – Strings

• Dynamically typed
  – No need to "type" a variable
  – Python figures it out based on what it is assigned
  – Can change when re-assigned

```python
# python 2.x
>>> 3 / 2
1    # python2.x
1.5  # python3.x

# python 3.x
>>> 3 // 2
1

>>> 1.25 / 0.5
2.5

>>> 2+4j + 3-2j
(5+2j)

>>> "Hello world"
'Hello world'

>>> 5 == 6
False

>>> x = 3
>>> x = "Hi"
>>> x = 5.0 + 2.5
```
Strings

• Enclosed in either double or single quotes
  – The unused quote type can be used within the string
• Can concatenate using the ‘+’ operator
• Can convert other types to string via the `str(x)` method
• Compare with ==, !=, etc.

```python
>>> 'spam eggs'
'spam eggs'

>>> "doesn't"
"doesn't"

>>> "Yes," he said.'
"Yes," he said.'

>>> "Con" + "cat" + "enate"
'Concateate'

>>> i = 5
>>> j = 2.75
>>> "i is " + str(i) + " & j is" + str(j)
'i is 5 & j is 2.75'
```
Simple Console I/O

- **Python3.x**
  - **Output using print()**
    - Must use parentheses
    - Use end='' argument for ending options
  - **Input using input(prompt)**
    - Returns a string of all text typed until the newline

- **Conversion to numeric types:**
  - `int(string_var)` convert to an integer
  - `float(string_var)` convert to a float

```python
>>> print("A new line will")
>>> print('be printed')
A new line will be printed

>>> print('A new line will', end='')
>>> print(' not be printed')
A new line will be printed

# Getting input
>>> response = input("Enter text: ")
Enter text: I am here

>>> print(response)
I am here

>>> response = input("Enter a num: ")
Enter a num: 6

>>> x = int(response)
>>> x = float(response)
```
Selection Structures

- **if...elif...else**
- Ends with a `:` on that line
- Blocks of code delineated by indentation (via tabs/spaces)

```python
myin = input("Enter a number: ")
x = int(myin)
if x > 10:
    print("Number is greater than 10")
elif x < 10:
    print("Number is less than 10")
else:
    print("Number is equal to 10")
```
Iterative Structures

• while <cond>:

• Again code is delineated by indentation

```python
secret = 18
attempts = 0
while attempts < 10:
    myin = input("Enter a number: ")
    if int(myin) == secret:
        print("Correct!")
        break
    attempts += 1
```
Lists

- Lists are like arrays from C++ but can have different (heterogenous) types in a single list object
- Comma separated values between square brackets
- Basic operations/functions:
  - append(value)
  - pop(loc)
  - len(list)

```python
>>> x = ['Hi', 5, 6.5]
>>> print(x[1])
5

>>> y = x[2] + 1.25
>>> y
7.75

>>> x[2] = 9.5
>>> x
['Hi', 9.5]

>>> x.append(11)
>>> x
['Hi', 9.5, 11]

>>> y = x.pop(1)
>>> y
5

>>> print(y)
5

>>> len(x)
3
```
Iterative Structures

- **for <item> in <collection>:**
  - collection can be list or some other collection
  - For a specific range of integers just use range() function to generate a list
    - Start is inclusive, stop is exclusive
    - range(stop)
      - 0 through stop-1
    - range(start, stop)
      - start through stop-1
    - range(start, stop, step)
      - start through stop in increments of stepsize

```python
# Prints 0 through 5 on separate lines
x = [0,1,2,3,4,5]  # equiv to x = range(6)
for i in x:
    print(i)

# Prints 0 through 4 on separate lines
x = 5
for i in range(x):
    print(i)

# Prints 2 through 5 on separate lines
for i in range(2,6):
    print(i)

x = ["hi", "world", "bye"]
mystring = ""
for word in x:
    mystring += word + " "
```
Exercise 1

• Get integers from the user until they type **quit**
• Output only the sum of the 1\textsuperscript{st} and last integers entered