CPSC 301: Computing in the Life Sciences
Lecture Notes 4:
Getting Started with Python

pre-class version

Jessica Dawson
jqdawson@cs.ubc.ca
http://www.cs.ubc.ca/~jqdawson

University of British Columbia
Department of Computer Science

2016 W2
Objectives

At the end of this lecture you should be able to:
• trace Python code that involves numerical and string operations
• trace Python code that uses numerical and string variables
• identify errors in code related to numerical and string operations
• write simple code involving these operations
• write code that reads input from the user, perform simple calculations and displays results to the user
Scratch vs Python

• Python is a general programming language, different than Scratch in many ways
• Scratch:
  – Multiple sprites, each with multiple scripts
  – Build scripts by plugging together blocks from a visual library
  – Blocks cause sprites to do things: move, change costume, play sounds, draw lines, etc.
  – Inputs from mouse, keyboard, microphone
  – Outputs to screen, speaker
• Python
  – One program doing one thing at a time
  – Build program by writing commands
  – Commands read input, change variables, write output
  – Inputs from keyboard, files
  – Outputs to screen, files
Getting Started in Python

• Technically, “Python” refers to the basic language
  – However, many of the most interesting features require the use of separate packages
  – A “Python distribution” includes not just the basic language but also a collection of common packages and often a simplified installer for the whole thing

• To practice Python at home, download and install the Anaconda distribution:

  http://continuum.io/downloads
  – Choose the Graphical Installer for Python 3.5 (NOT 2.7).

• We will not be using exactly the same Python as the book
  – We will be using the Spyder Integrated Development Environment (IDE) instead of IDLE
Spyder Integrated Development Environment (IDE)

- One of many ways of working with & running Python code

Area for editing your Python programs: the “text editor” or “editor”

Area for running Python code or watching the output of a running program: the “console” (we suggest you make it bigger)

Area for looking at variable values or documentation
What should you do if you are typing commands into Python at the >>> prompt and the responses are not what you expect:

1) Check what you typed. A single mistyped character can cause an entirely different response.

2) Restart Python. You can do this by selecting “open a Python Console” from either the Consoles menu (at the top of the Spyder window) or from the small white dropdown button on the top right edge of your console window (if you have not closed the console window). See the next slide.

3) Check what you typed again.

4) Restart Spyder.
Starting a new Python Console

Choose “open a Python console” from either of these menus

The new window will normally appear here as a tab called “Python #” (where “#” is an integer)
We will not use IPython

• Spyder offers the options of opening an “IPython console”
  – IPython is another way of interacting with Python which offers some powerful features
  – We will stick with regular Python consoles because that is much closer to the textbook (eg: the IDLE platform)
  – If you use Python interactively (eg: not just running other people’s scripts) in the future, you should strongly consider using IPython
How to Get Rid of IPython Consoles

• An IPython console may create up to two windows:
  – “Kernel ####” in the Console tab, where the regular Python consoles are open. (only appears in some versions of Spyder, including on lab computers)
  – “Console ####/A” in the IPython tab
• To get rid of these windows (may vary slightly depending on Spyder version):
  – close the IPython console window window (or type “exit” in it) and then when asked “Do you want to kill the associated kernel and all of its clients” click “yes”
  – From the spyder menu bar select ‘View’ > ‘Panes’ and then deselect ‘IPython Console’
Online Python Tutor

- [http://pythontutor.com/visualize.html](http://pythontutor.com/visualize.html)
- Useful for visualizing Python execution
  - You can step back and forth through the code, watching how data changes in memory
Numeric Values and Operations

• Two types of numbers:
  – integers (called \textit{int}): 0, 5, -20
  – decimal numbers (\textit{float}): 2.5, 0.25555, 5.0, 0.0

• The following arithmetic operations are defined for both types:
  – $+, -, \ast$: addition, subtraction and multiplication
  – $/$: division; always returns a float
    • i.e. $6 / 3$ is 2.0
  – $\mathbf{//}$: integer division; always returns an int
    • i.e. $14 \mathbf{//} 5$ is 2
  – $\%$: remainder (or modulo) operator
    • i.e. $14 \% 5$ is 4
  – $\ast\ast$: exponentiation
    • i.e. $2 \ast\ast 6$ is 64

• The result of an operation is of the same type as the operands, if they are of same type. Otherwise the int operand is converted to float. I.e.
  – $5 + 2$ is 7, but $5.0 + 2$ is 7.0
  – $5 \mathbf{//} 2$ is 2, but $5.0 \mathbf{//} 2$ is 2.0
Numeric Values and Operations (cont)

• Floats have finite precision:
  – Equality on floats is tricky
  – $2 / 3 = 0.6666666666666666$
  – $5 / 3 = 1.6666666666666667$
  – $1 + 2 / 3$ is not the same as $5 / 3$

• Operators are performed from left to right with this precedence:
  – **
  – -(negation)
  – *, /, //, %
  – +, -
  – For instance: $10 - 2 * 5 + 6 / 2$ is ____

• Can use () to force different order.
  – I.e. $10 - 2 * (5 + 6) / 2$ is ____
Variables

- Variables are names that have values we like to keep and use later (like Scratch variables)
- Variable names may only have letters, digits and underscore(_) and the first character must be a letter or _
- A variable is created *(declared)* using an *assignment statement* which assigns a value to a variable using the *assignment operator* ":="
  
  ```
  speed = 4
  time = 2.5
  ```

- A variable’s value is stored in a memory location, and the variable refers to the appropriate memory location by its address
- When we trace Python code, to keep track of the variables and their values we use the following notation *(memory model)*:
  
  ```
  speed → 4
  time → 2.5
  ```

- After created, when a variable is used Python uses its value

  ```
  distance = speed * time
  time = time + 2
  distance = speed * time
  ```

**Assignment statements are executed as following:**

1. evaluate the expression on the right of =
2. assign the value to the variable on the left of =
Combined Operators (Augmented Assignment)

• Notice the difference between the following operations:
  
  \[
  \begin{align*}
  \text{time} &= 20 \\
  \text{time} &= \text{time} + 20
  \end{align*}
  \]

• Operations like the last one can be combined as following
  
  – \( \text{time} += 20 \) means \( \text{time} = \text{time} + 20 \)
  – \( \text{time} -= 20 \) means \( \text{time} = \text{time} - 20 \)
  – \( \text{time} *= 20 \) means \( \text{time} = \text{time} \times 20 \)
  – \( \text{time} /= 20 \) means \( \text{time} = \text{time} / 20 \)

• Evaluation of these expressions:
  
  – first evaluate the expression on the right of =
  – apply the operation (attached to =) to the value of the variable on the left of =
  – assign the result to variable on the left of =

• Suppose distance, time and seed are as shown on the box to the right. Show their new values after executing
  
  \[
  \text{distance} += \text{speed} \times \text{time}
  \]
Strings

• In Scratch we could type either a number or a string into some rectangular boxes but it was interpreted according to the operation
• Python is more specific: A string is any sequence of characters within " " (double quotes) or ' ' (single quotes)
  – i.e. "Hello there", 'bus 123', "bigshot@gmail.com"
  – 35 and 2.5 are numbers, but "35" and '2.5' are strings
  – "" and " are the empty string (a string with 0 characters)
• The type of a string is str
• If a string spans multiple lines, use three quote characters of either type to start and finish the string (a block string)
• Backslash character generates "escape" characters
  – "\\" becomes '\', '"" becomes "", '"\" becomes \\
  – '\n' is translated into the control character newline (when it is printed, it will end the current line and start the next)
  – "that's right" is right, and so is 'that\'s right'
String Operators

• The function `len` returns the length of a string
  – `len("Big Bang")` is 8
  – `len(""""")` is 0

• Can join two strings by placing them one after the other
  – "Hi" "There" will be "HiThere"

• Can also use `+` (**concatenation operator**) to join two strings
  – "Hi" + "There" will be "HiThere"
  – "Hi" + "5" is "Hi5"
  – "Hi" + 5 produces an error, but "Hi" + str(5) is "Hi5"

• We can repeat a string using the `*` operator
  – 'GC' * 3 is 'GCGCGC' (and so is 3* 'GC')
  – 'GC' * 0 is " (empty string)
  – 'GC' * - 5 is "

• More operations on strings later
Printing

• When a statement is written at the shell, its value is displayed on the next line. This is not true with programs that are not written line by line at the shell.

• To display one or more values, a program has to use the `print` function:
  – `Print( 1 + 4 / 2)` will print: `3.0`
  – `Print( "this is a string" )` will print: `this is a string`
  – `Print( "The result is", 5 + 20 // 4, "units." )` will print: `The result is 10 units.`

• `print` accepts any number of arguments of any type:
  – Arguments are printed in order.
  – By default, a space is placed between each argument and a newline after the final argument (the default can be changed).
Example 1: Room Painting Cost

# A program that computes the cost of painting a room with four walls.
#
# Input:
#    The dimensions of the room.
#    The price of paint per gallon.
#    The cost of labour per hour.
# Output: the cost to paint the walls of the room broken down
#         in paint and labour costs.
# The painting company has determined that for every 50 square feet
# of wall they need 1 gallon of paint and 5 hours of labour.

# Room dimensions (in feet).
room_length = 10
room_width = 15
room_height = 12

# Paint price per gallon
paint_price = 100

# Labour cost per hour
labour_price = 20

# Calculate the area of the two types of walls the room
# has
wall_1 = room_length * room_height
wall_2 = room_width * room_height

total_area = 2 * wall_1 + 2 * wall_2

print("Total area is:", total_area)

# Calculate the cost of paint needed
gallons = total_area / 50
paint_cost = gallons * paint_price

print("Paint cost is:", paint_cost)

# Calculate labour cost
hours = total_area / 50 * 5
labour_cost = hours * labour_price

print("Labour cost is:", labour_cost)

# Calculate total cost
total_cost = paint_cost + labour_cost

print("Total cost is:", total_cost)
Getting Input from User

- Often a program needs to get input values from the user.
- This can be done using `input()` which will read a single line of text from the keyboard and return it as a string.
- Examples:
  ```python
  >>> name = input()
  John Smith
  >>> email = input()
  smith@gmail.com
  >>> print( "Hi ", name, " I'll email you at ", email )
  Hi John Smith I'll email you at smith@gmail.com
  ```
- `Input()` can display a message and then wait for the input:
  ```python
  >>> name = input("Enter Your name: ")
  Enter Your name: John Smith
  >>> email = input("Enter your email address: ")
  Enter your email address: smith@gmail.com
  >>> print( "Hi ", name, " I'll email you at ", email )
  Hi John Smith I'll email you at smith@gmail.com
  ```
If the input is a number, we need to convert the string of digits read in by `input()` to the actual number using `int()` or `float()`.

Examples:

```python
>>> speed = input()
10
>>> print( speed + 2 )
>>> Error......
```

produces an error as `speed` is a string, but the following works:

```python
>>> speed = float( input() )
10
>>> speed = float(speed)
>>> print( speed + 2 )
12.0
```

OR

```python
>>> print( speed + 2 )
12.0
```
Version 2 of Last Example

# A program that computes the cost of painting a room with
# four walls.
#
# Input:
#   - The dimensions of the room.
#   - The price of paint per gallon.
#   - The cost of labour per hour.
#
# Output: the cost to paint the walls of the room broken down
#   in paint and labour costs.
#
# The painting company has determined that for every 50
#   square feet of wall they need 1 gallon of paint and 5
#   hours of labour.

# Get room dimensions (in feet).
room_length = float(input("Enter room's length: "))
room_width = float(input("Enter room's width: "))
room_height = float(input("Enter room's height: "))

# Get paint price per gallon
paint_price = float(input("Enter paint's price (per gallon): "))

# Get labour price per hour
labour_price = float(input("Enter labour price (per hour): "))

# Calculate the area of the two types of walls the room has
wall_1 = room_length * room_height
wall_2 = room_width * room_height

# Total area for painting
total_area = 2 * wall_1 + 2 * wall_2
print("Total area is:", total_area)

# Calculate the cost of paint needed
gallons = total_area / 50
paint_cost = gallons * paint_price
print("Paint cost is:", paint_cost)

# Calculate labour cost
hours = total_area / 50 * 5
labour_cost = hours * labour_price
print("Labour cost is:", labour_cost)

# Calculate total cost
total_cost = paint_cost + labour_cost
print("Total cost is:", total_cost)
Getting Help for Python

- Figure out which version you are using, and make sure you get the corresponding version of the documentation
  - The current Anaconda distribution’s version of Spyder uses version 3.5, which appears to match the book well
- Python has many components, all with documentation
  - For this course, you will typically look at what is called the “Library Reference” or the “Python Standard Library”
- In Spyder: Help -> Python documentation pops up a window
- From the Python web page www.python.org
  - General documentation page (including a search box): http://docs.python.org/
- Google for many more tutorials or references
Some Terminology: Python Keywords

- The **Python keywords** are words that you cannot use as variable names
  - This list of reserved words has nothing to do with the “keyword arguments” that are used for calling functions
  - An attempt to use a reserved keyword as a variable name (or function name, etc.) generates a SyntaxError

- The list includes words used for various purposes
  - Special values: True, False, None
  - Operators: and, or, not, in, is,...
  - Statements: return, def, import, from, if, elif, else, while, for,...

- Case matters
  - You can still use variables called true, And, Return,... (although it is a really bad idea)
Some Terminology: Expressions vs Statements

• An “expression” is a small piece of code which can be evaluated (its steps can be performed) to determine a value (a piece of data)
  – Expressions can sometimes cause things to happen as a “side-effect”
• A “statement” is a small piece of code which does something
  – Statements often contain one or more expressions
• In Scratch both expressions and statements were blocks
• Why do you care?
  – When you type statements at the command prompt (eg: “>>>”): If the statement is an expression (eg: generates a value) and nothing else (eg: no assignment) then Python will display that value
  – When you run statements that are inside a script (eg: saved in a *.py file): Python will display nothing unless you explicitly tell it to (eg: using `print()`; see PP 3)
  – In both cases, the value generated by expressions will be discarded unless saved into a variable using the assignment operator =
Every value in Python has a specific type.
Variables are declared by assignment ("=") statements that assign a value to them.
Variables have the same type as their value. They must be declared before they are used.
Two numeric types: integers (int) and decimal numbers (float)
All arithmetic operations apply, the result is of same type as operands (if operands are not the same they are converted).
Arithmetic operations can be combined with assignment op.
Strings are sequences of characters in " " or ' ' or ''' '''.
Special characters like newlines are represented by escape sequences using \.
Values can be displayed using print().
The program can get input from the user using input().