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

2017-01-19

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

University of British Columbia
Department of Computer Science

2016 W2
Admin

• **TA office hours:** Schedule pinned on Piazza.

• **Lab Section attendance**
  – Labs are very full!
  – Therefore, you’re expected to attend your registered lab
  – There are legitimate reasons to attend another section (e.g. missed lab for illness), but you should notify the instructor/TAs ahead of time

• **Lab 2:** Due Sunday @ 9am (usual due date / time)
  – See hint on Piazza regarding making the food disappear

• **Lab 3:** Posted

• **There will be pre-class materials for Tuesday**
Rounding operation (relevant for lab 3)

- Python comes with many handy built-in functions that perform common operations.

- The function `round()` will explicitly round floating point values
  - `round(1.66666, 0)` will generate the result `2.0`

- The function takes two arguments:
  - First argument is the number to round. You can pass it a number directly, or a variable that stores a number.
  - Second argument is the number of decimal places to round to.

- For example, the following input to the console would produce
  
  ```python
  >>> number = 12.566
  >>> round(number, 1)
  12.6
  ```
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
• In our simplified memory model, we represent strings with quotes
  – myString → “Hello there”
• 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  – '\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
Some Common Types of Errors

- **SyntaxError**: occurs when you type something that isn’t valid in Python code. I.e. you’ve written it in a way python can’t understand.
  - E.g. 5 = x

- **NameError**: occurs when you use a variable or a function that does not exist. Can occur even because of differences in case. E.g.
  ```
  double = 1
  double = Double * 2
  ```

- **TypeError**: occurs when you apply an operation or function to a variable or value of an inappropriate type
  - “5” + 5
    will produce a TypeError because Python cannot add a string and an int together.
  - You need an extra piece of code to tell Python to treat a number as a string.
  - E.g. “5” + str(5) is “55”
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 (comma separated) 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 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)
Opening/running a program (a .py file)

- Open program
- Run program
- Refresh variables (after run)

Program will typically run in selected console (default setting can be changed)

Variables shown in explorer are for selected console.
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 = input()
10
>>> speed = float(input())
10
```

OR

```python
>>> speed = float(speed)
>>> print( speed + 2 )
>>> 12.0
```
# 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): "))

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

print()  # print an empty line

# 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 =
Summary

- 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.
- Floats can be rounded using round()
- 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().
Change log from pre-class slides

• Added a slide on `round()` function
• Added an example on the String slide to show how we represent strings in a simplified memory model
• Added a slide summarizing types of errors
• Changed the ‘Print’ calls on the slide about Printing to lowercase (e.g. `print()`).
  – Note that upper case (e.g. `Print()` throws a name error).
• Added a slide illustrating how to run a program.
• Updated summary slide