REVIEW OF PYTHON-PART 1

Jahan Ghofraniha, Ph.D.

Adapted from Python for Everybody by Charles R. Severance

Introduction

- A computer program consists of data and operation on data.
- A computer program stores and manipulates data by storing it in a computer memory.
- In order to refer to that memory location and the value stored in it we need to come up with the concept of a variable.
- A variable is a mapping of a label to a memory location.
- You can think of the computer memory as a spreadsheet (a grid) that you can refer to a specific location in memory by its row and column location/address.
- When we define a variable such as A, name, ID, …, the computer assigns a memory location to it but there is no value in that location.
- The assignment operation will give that location a value, for example in Python we can use the following assignment operation:
- Name = 4

Variables, Statements & Expression

- In the example, name is the **variable** name and the **value** of the variable is 4.
- Depending on what **type** of data we assign to a variable or how we define a variable the length of storage and the interpretation of it changes.
- The concept of data type is important since the same storage value can be interpreted by the computer differently depending on the type.
- The standard data types in Python and most other languages are: **integers**, **floats**, **doubles**, **strings**, **Boolean**.
- The following examples demonstrate different types of data in Python:

Data Types

- intVar = 4 # intVar is an integer with value 4 assigned
- ftVar = 3.14 # ftVar is a floating point variable with value of 3.14
- str = ‘Hello’ # str is string variable
- cond = True # cond is a Boolean variable that is set to True

- If we want to check the type of a variable we can use type() by:
- type(str)

- Variable names should be chosen in a way that is meaningful and descriptive while being concise.
- Variable names can be arbitrarily long. They can contain both letters and numbers, but they cannot start with a number. It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter.

Variable Names

- Illegal variable names:
  - Start with a number: 76name
  - Contain illegal characters such as @: more@
  - Use a Python keyword such as class= 2

```
76trombones = 'big parade'
SyntaxError: invalid syntax
more@ = 1000000
SyntaxError: invalid syntax
class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Python Reserved Keywords

- Python reserves 33 keywords:

And, del, from, None, True

As, elif, global, nonlocal, try

Assert, else, if, not, while

Break, except, import, or, with

Class, False, in, pass, yield

Continue, finally, is, raise

Def, for, lambda, return

- A statement is a unit of code that the Python interpreter can execute. We have seen two kinds of statements: print being an expression statement and assignment.

Statements, Expressions, Operators

- When you type a statement in interactive mode, the interpreter executes it and displays the result, if there is one.
- An **expression** is an instruction that combines values and operators and always evaluates down to a single value. For example 2+2 results in the output of 4. Here there is no assignment but an expression. Both expressions and assignments are statements.
- **Operators** are special symbols that represent computations like addition and multiplication.
- **Operators**: *, /, +, -, **, // and % represent: multiplication, division, addition, subtraction, exponentiation, integer divide and remainder operators in Python 3.x.

String Operations & User Input

- The + operator works with strings, but it is not addition in the mathematical sense.
- Instead it performs *concatenation*, which means joining the strings by linking them end to end.
- In many occasions you need to ask the user to enter or input something. The input() provides this functionality in Python.
- name = input(“What is your name?\n”)
- There are two ways to comment in Python, single line using # (number sign or hash tag) or multiple lines by using three single quotes to indicate the start of the comments and three single quotes to end the multi-line comments.

Exercise

- Ex1: Write a program that uses input to prompt a user for their name and then welcomes them.
- Ex2: Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature.

Conditional Statements

- A Boolean *expression* is an expression that is either true or false. The following examples use the operator ==, which compares two operands and produces True if they are equal and False otherwise:
- 5 == 5
- 5 == 6
- Other comparison operators besides ==, are:

```
x != y          # x is not equal to y
x > y           # x is greater than y
x < y           # x is less than y
x >= y          # x is greater than or equal to y
x <= y          # x is less than or equal to y
x is y          # x is the same as y
x is not y      # x is not the same as y
```

Logical Operators

- There are three *logical operators*: **and**, **or**, and **not**. The semantics (meaning) of these operators is similar to their meaning in English. For example,
- x > 0 and x < 10
- is true only if x is greater than 0 *and* less than 10.
- n%2 == 0 or n%3 == 0 is true if *either* of the conditions is true, that is, if the number is divisible by 2 *or* 3.
- Finally, the not operator negates a Boolean expression, so not (x > y) is true if x > y is false; that is, if x is less than or equal to y.

One-way & Two-way if statement

```
# one-way if statement
x = 3
If x < 10:
    print(‘Small’)
# two-way if statement
if x%2==0:
    print (‘Even’)
Else:
    print(‘Odd’)
# chained if-statement (multiple conditions)
if x < y:
    print('x is less than y')
elif x > y:
     print('x is greater than y')
else:
    print('x and y are equal')
```

Nested if statement

- One conditional can also be nested within another. We could have written the three-branch example like this:

```
if x == y:
        print('x and y are equal')
else:
        if x < y:
                print('x is less than y')
        else:
                print('x is greater than y')
```

Exercise

- Ex1: Write a program to prompt the user for hours and rate per hour to compute gross pay. If the number of hours is more than 40, the number of hours in excess of 40 should be calculated as 1.5 the normal rate.

Iterations

- Computers are often used to automate repetitive tasks. Repeating identical or similar tasks without making errors is something that computers do well and people do poorly. Because iteration is so common, Python provides several language features to make it easier.
- One form of iteration in Python is the while statement.
- The while statement is easy to interpret. It is the same meaning as the wording implies:
- While the condition holds (is true), do the following stuff and stop when the condition is no longer true.

```
n = 10
While n>0:
    print (n)
    n = n - 1
```

Iterations

- There are occasions that we need to have an infinite loop. Infinite loops are quite common in embedded systems.
- A loop is constantly running on a microprocessor and when there is a hardware notice/interrupt, it will branch to the proper location in the program to do the task that is tied to that hardware interrupt and then returns to the loop.
- As indicated above, infinite loops run forever so we need a mechanism to break out of the loop. In Python, we use the **break** keyword.
- The break keyword normally tied to a condition breaks out of the loop when the condition is met. Look at the following example:

Break & Continue Keywords in Loops

```
while True:
    line = input('> ')
    if line == 'done':
        break       # crucial in breaking out of an infite loop
    print(line)
print('Done!')
```

Continue

- Sometimes we want to finish or skip certain iterations in a loop. In this case, we use the **continue** keyword.

```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```

For Loops

- Sometimes we want to loop through a *set* of things such as a list of words, the lines in a file, or a list of numbers.
- When we have a list of things to loop through, we can construct a *definite* loop using a for statement.
- While loops are called indefinite loops since it runs until the condition become false. The for loop is called a definite loop since we know exactly how many iterations the loop should run through before it is done.

```
# For-Loop
for i in range(10):
        print(i)
```

Exercise

- Ex1: Write a program which repeatedly reads numbers until the user enters “done”. Once “done” is entered, print out the total, count, and average of the numbers.