Lab2-Solution

August 26, 2020

[1]:
```
# Preamble script block to identify host, user, and kernel
import sys
! hostname
! whoami
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

```
atomickitty.aws
compthink
/opt/conda/envs/python/bin/python
3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0]
sys.version_info(major=3, minor=8, micro=3, releaselevel='final', serial=0)
```

[2]:
```
%%html
<!--Script block to left align Markdown Tables-->
<style>
  table {margin-left: 0 !important;}
</style>
```

```
<IPython.core.display.HTML object>
```

1 ENGR 1330 Sec D52/D54 Laboratory 2

This laboratory is an introduction to variables, operators, expressions, basic I/O, and string manipulation. Notice there is code cell above, from this notebook forward please include and run the script in the cell, it will help in debugging a notebook.

1.1 Variables

Variables are names given to data that we want to store and manipulate in programs. A variable has a name and a value. The value representation depends on what type of object the variable represents. The utility of variables comes in when we have a structure that is universal, but values of variables within the structure will change - otherwise it would be simple enough to just hardwire the arithmetic.

Suppose we want to store the time of concentration for some hydrologic calculation. To do so, we can name a variable `TimeOfConcentration`, and then `assign` a value to the variable, for instance:

`TimeOfConcentration = 0.0`

After this assignment statement the variable is created in the program and has a value of 0.0. The use of a decimal point in the initial assignment establishes the variable as a float (a real variable is called a floating point representation -- or just a float).

1.1.1 Naming Rules

Variable names in Python can only contain letters (a - z, A - Z), numerals (0 - 9), or underscores. The first character cannot be a number, otherwise there is considerable freedom in naming. The names can be reasonably long. `runTime`, `run_Time`, `_run_Time2`, `_2runTime` are all valid names, but `2runTime` is not valid, and will create an error when you try to use it.

[3]:
```
# Script to illustrate variable names
runTime = 1
_2runTime = 2 # change to 2runTime = 2 and rerun script
runTime2 = 2
print(runTime,_2runTime,runTime2)
```

```
1 2 2
```

There are some reserved words that cannot be used as variable names because they have preassigned meaning in Parseltongue. These words include `print`, `input`, `if`, `while`, and `for`. There are several more; the interpreter won't allow you to use these names as variables and will issue an error message when you attempt to run a program with such words used as variables.

1.2 Operators

The `=` sign used in the variable definition is called an assignment operator (or assignment sign). The symbol means that the expression to the right of the symbol is to be evaluated and the result placed into the variable on the left side of the symbol. The "operation" is assignment, the "=" symbol is the operator name.

Consider the script below

[4]:
```
# Assignment Operator
x = 5
y = 10
print (x,y)
x=y # reverse order y=x and re-run, what happens?
print (x,y)
```

```
5 10
10 10
```

So look at what happened. When we assigned values to the variables named `x` and `y`, they started life as 5 and 10. We then wrote those values to the console, and the program returned 5 and 10. Then we assigned `y` to `x` which took the value in y and replaced the value that was in x with this value. We then wrote the contents again, and both variables have the value 10.

1.3 Arithmetic Operators

In addition to assignment we can also perform arithmetic operations on variables. The fundamental arithmetic operators are:

Symbol	Meaning	Example
=	Assignment	x=3 Assigns value of 3 to x.
+	Addition	x+y Adds values in x and y.
-	Subtraction	x-y Subtracts values in y from x.
*	Multiplication	x*y Multiplies values in x and y.
/	Division	x/y Divides value in x by value in y.
//	Floor division	x//y Divide x by y, truncate result to whole number.
%	Modulus	x%y Returns remainder when x is divided by y.
**	Exponentation	x**y Raises value in x by value in y. (e.g. x^y)
+=	Additive assignment	x+=2 Equivalent to x = x+2.
-=	Subtractive assignment	x-=2 Equivalent to x = x-2.
*=	Multiplicative assignment	x*=3 Equivalent to x = x*3.
/=	Divide assignment	x/=3 Equivalent to x = x/3.

Run the script in the next cell for some illustrative results

[5]:
```
# Uniary Arithmetic Operators
x = 10
y = 5
print(x, y)
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print((x+1)//y)
print((x+1)%y)
print(x**y)
```

```
10 5
15
5
50
2.0
2
```

```
1
100000
```

[6]:
```python
# Arithmetic assignment operators
x = 1
x += 2
print(type(x),x)
x = 1
x -= 2
print(type(x),x)
x = 1
x *=3
print(type(x),x)
x = 10
x /= 2
print(type(x),x)  # Interesting what division does to variable type
```

```
<class 'int'> 3
<class 'int'> -1
<class 'int'> 3
<class 'float'> 5.0
```

1.4 Data Type

In the computer data are all binary digits (actually 0 and +5 volts). At a higher level of abstraction data are typed into integers, real, or alphanumeric representation. The type affects the kind of arithmetic operations that are allowed (as well as the kind of arithmetic - integer versus real arithmetic; lexicographical ordering of alphanumeric , etc.) In scientific programming, a common (and really difficult to detect) source of slight inaccuracies (that tend to snowball as the program runs) is mixed mode arithmetic required because two numeric values are of different types (integer and real).

Learn more from the textbook

https://www.inferentialthinking.com/chapters/04/Data_Types.html

Here we present a quick summary

1.4.1 Integer

Integers are numbers without any fractional portion (nothing after the decimal point { which is not used in integers). Numbers like -3, -2, -1, 0, 1, 2, 200 are integers. A number like 1.1 is not an integer, and 1.0 is also not an integer (the presence of the decimal point makes the number a real).

To declare an integer in Python, just assign the variable name to an integer for example

`MyPhoneNumber = 14158576309`

1.4.2 Real (Float)

A real or float is a number that has (or can have) a fractional portion - the number has decimal parts. The numbers 3.14159, -0.001, 11.11, 1., are all floats. The last one is especially tricky, if you don't notice the decimal point you might think it is an integer but the inclusion of the decimal point in Python tells the program that the value is to be treated as a float. To declare a float in Python, just assign the variable name to a float for example

```
MyMassInKilos = 74.8427
```

1.4.3 String(Alphanumeric)

A string is a data type that is treated as text elements. The usual letters are strings, but numbers can be included. The numbers in a string are simply characters and cannot be directly used in arithmetic. There are some kinds of arithmetic that can be performed on strings but generally we process string variables to capture the text nature of their contents. To declare a string in Python, just assign the variable name to a string value - the trick is the value is enclosed in quotes. The quotes are delimiters that tell the program that the characters between the quotes are characters and are to be treated as literal representation.

For example

```
MyName = 'Theodore'
MyCatName = "Dusty"
DustyMassInKilos = "7.48427"
```

are all string variables. The last assignment is made a string on purpose. String variables can be combined using an operation called concatenation. The symbol for concatenation is the plus symbol `+`.

Strings can also be converted to all upper case using the `upper()` function. The syntax for the `upper()` function is `'string to be upper case'.upper()`. Notice the "dot" in the syntax. The operation passes everything to the left of the dot to the function which then operates on that content and returns the result all upper case (or an error if the input stream is not a string).

[7]:
```
# Variable Types Example
MyPhoneNumber = 14158576309
MyMassInKilos = 74.8427
MyName = 'Theodore'
MyCatName = "Dusty"
DustyMassInKilos = "7.48427"
print("All about me")
print("Name: ",MyName, " Mass :",MyMassInKilos,"Kg" )
print('Phone : ',MyPhoneNumber)
print('My cat\'s name :', MyCatName)  # the \ escape character is used to get␣
 ↪the ' into the literal
print("All about concatenation!")
print("A Silly String : ",MyCatName+MyName+DustyMassInKilos)
print("A SILLY STRING :  ", (MyCatName+MyName+DustyMassInKilos).upper())
```

```
All about me
Name:  Theodore  Mass : 74.8427 Kg
Phone :  14158576309
My cat's name : Dusty
All about concatenation!
A Silly String :  DustyTheodore7.48427
A SILLY STRING :   DUSTYTHEODORE7.48427
```

Strings can be formatted using the % operator or the `format()` function. The concepts will be introduced later on as needed in the workbook, you can Google search for examples of how to do such formatting.

1.4.4 Changing Types

A variable type can be changed. This activity is called type casting. Three functions allow type casting: `int()`, `float()`, and `str()`. The function names indicate the result of using the function, hence `int()` returns an integer, `float()` returns a oat, and `str()` returns a string.

There is also the useful function `type()` which returns the type of variable.

The easiest way to understand is to see an example.

[8]:
```
# Type Casting Examples
MyInteger = 234
MyFloat = 876.543
MyString = 'What is your name?'
print(MyInteger,MyFloat,MyString)
print('Integer as float',float(MyInteger))
print('Float as integer',int(MyFloat))
print('Integer as string',str(MyInteger))
print('Integer as hexadecimal',hex(MyInteger))
print('Integer Type',type((MyInteger)))  # insert the hex conversion and see␣
 ↪what happens!
```

```
234 876.543 What is your name?
Integer as float 234.0
Float as integer 876
Integer as string 234
Integer as hexadecimal 0xea
Integer Type <class 'int'>
```

1.5 Expressions

Expressions are the ”algebraic” constructions that are evaluated and then placed into a variable. Consider

`x1 = 7 + 3 * 6 / 2 - 1`

The expression is evaluated from the left to right and in words is

```
Into the object named x1 place the result of:

integer 7 + (integer 6 divide by integer 2 = float 3 * integer 3 = float 9 - integer 1 = float
```

The division operation by default produces a float result unless forced otherwise. The result is the variable `x1` is a float with a value of `15.0`

[9]:
```python
# Expressions Example
x1 = 7 + 3 * 6 // 2 - 1  # Change / into // and see what happens!
print(type(x1),x1)
## Simple I/O (Input/Output)
```

```
<class 'int'> 15
```

1.6 Simple string manipulation

Exercise-1: Change the cell below to a code cell and run the script print('Sup World?') print(3 + 2) print('3 + 2') MyNumber = 3+2 MyName = 'Dusty' print(MyName, MyNumber) print('MyName', 'MyNumber')

Answer the following questions based on the output

1. What is the difference between `print( 3 + 2 )` and `print( '3 + 2')`?
2. What is the difference between `print( MyName, MyNumber)` and `print('MyName', 'MyNumber')`
3. Change `MyNumber = 3+2` to `MyNumber = 3+2.0`, and re-run the script, what happens? Why?

Write your answers below: (Change the cells to Markdown)

1. Question 1 The print statements write to the console, in this case an output cell
2. `print(3 + 2 )` prints the result of integer addition of 3 and 2, which is the integer 5.
3. `print('3 + 2')` prints the literal (string) characters "3", "+", and "2" with a single whitespace between each character.
2. Question 2 The print statements write to the console, in this case an output cell
3. `print( MyName, MyNumber)` prints the contents of the variables `MyName` and `MyNumber`, in this case `Dusty` and `5`
4. `print('MyName', 'MyNumber')` prints the literal strings `MyName` and `MyNumber`
3. Question 3 Changing `MyNumber = 3+2` to `MyNumber = 3+2.0` changes the arithemetic to floating point because the 2.0 is a float. The kernel performs mixed-mode arithmetic as float by default.

Exercise-2: Variable Types Create a simple script that illustrates the following types of variables

- integer

- floating point (real)
- string (alphanumeric)
- boolean

by

1. Assigning a value to a string, integer, float, and boolean variable. Use the names below

   ```
   string_theory = ...
   integer_type = ...
   floating_point = ...
   boolean_type = ...
   ```

2. Then print the type and contents of each variable.

   ```
   print(type(string_theory),string_theory)
   print(type(integer_type), integer_type))
   ...
   ...
   ```

[10]:
```
string_theory = "What's up?"
integer_type = 2
floating_point = 2.0
boolean_type = True
print(type(string_theory),string_theory) # prints contents of string without␣
 ↪literal delimiter (the quotes)
print(type(integer_type),integer_type)
print(type(floating_point),floating_point)
print(type(boolean_type),boolean_type)
```

```
<class 'str'> What's up?
<class 'int'> 2
<class 'float'> 2.0
<class 'bool'> True
```

Exercise-3: Arithmetic and Expressions Calculate the expressions below by hand taking care to keep track of result type (integer or float)

`x1 = 7 + 3 * 6 / 2 - 1`

`x2 = 2 % 2 + 2 * 2 - 2 / 2`

`x3 = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) )`

Write your results below

x1 (by hand) = 15.0 type real

x3 (by hand) = 3.0 type real

x3 (by hand) = 324.0 type real

Now write a script to evaluate and print the results, by

1. Assigning a value to a variable. Use the names above

   ```
   x1 = 7 + 3 * 6 / 2 - 1
   x2 = ...
   x3 = ...
   ```

2. Then print the type and contents of each variable.

   ```
   print(type(x1),x1)
   print(type(x2),x2)
   ...
   ```

[11]:
```
x1 = 7 + 3 * 6 / 2 - 1
x2 = 2 % 2 + 2 * 2 - 2 / 2
x3 = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) )

print(type(x1), x1)
print(type(x2), x2)
print(type(x3), x3)
```

```
<class 'float'> 15.0
<class 'float'> 3.0
<class 'float'> 324.0
```

Exercise-4: Simple Input/Output Get two floating point numbers via the `input()` function and store them under the variable names `float1` and `float2`.

```
float1 = input("Please enter float1: ")
float1 = float(float1)
...
```

Print `float1` and `float2` to the output screen.

```
print("float1:", float1)
...
```

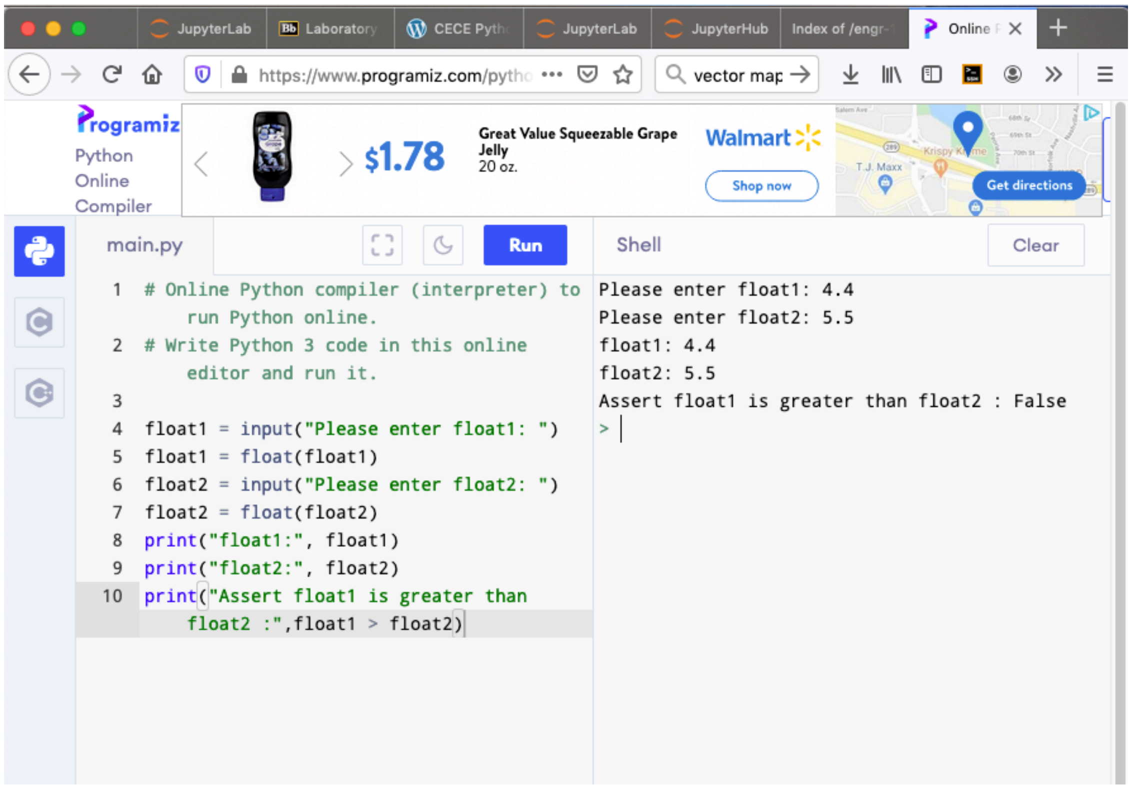
Then check whether `float1` is greater than or equal to `float2`.

[12]:
```
float1 = input("Please enter float1: ")
float1 = float(float1)
float2 = input("Please enter float2: ")
float2 = float(float2)
print("float1:", float1)
print("float2:", float2)
print("Assert float1 is greater than float2 :",float1 > float2)
```

```
Please enter float1:  4.4
Please enter float2:  5.5
```

```
float1: 4.4
float2: 5.5
Assert float1 is greater than float2 : False
```

Optional Copy your script to an on-line Python compiler and run are the results the same? Yes



```
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.

float1 = input("Please enter float1: ")
float1 = float(float1)
float2 = input("Please enter float2: ")
float2 = float(float2)
print("float1:", float1)
print("float2:", float2)
print("Assert float1 is greater than float2 :",float1 > float2)
```

```
Please enter float1: 4.4
Please enter float2: 5.5
float1: 4.4
float2: 5.5
Assert float1 is greater than float2 : False
>
```

Exercise-5: String Element Manipulation Define the string given below in quotes to a meaningful variable name.

`some_string ='Computational Thinking'`

Then

1. Index and print all the elements from index positions 2 to 10.

```
begin = ???
end = ???
print(some_string[begin:end])
```

2. Index and print the string 'Think'.

[13]:
```
mystring = 'Computational Thinking'
begin = 2 ; end = 10 # note syntax ; allows multiple assignments on same line
print(mystring[begin:end])
begin = 14 ; end = 19
print(mystring[begin:end])
```

```
mputatio
Think
```

[ ]: