Download (right-click, save target as ...) this page as a jupyterlab notebook from:

Laboratory 2

Laboratory 2: Elementary Computation

Medrano, Giovanni

R11521018

ENGR 1330 Laboratory 2 - In-Lab

Welcome to your first (or second) Jupyter Notebook. This is the medium that we will be using throughout the semester.

Why is this called a notebook? Because you can write stuff in it!

Is that it? Nope! you can **write** and **run** CODE in this notebook! Plus a bunch of other cool stuff such as making graphs, running tests and simulations, adding images, and building documents (such as this one!).

Notice the code cell below! From this notebook forward please include and run the script in the cell, it will help in debugging a notebook. Its ok if the code makes no sense right now - mostly the cell executes system commands If you change machines, and rerun the cell the output will change (its supposed to!)

In [50]:
```
# Preamble script block to identify host, user, and kernel
import sys
! hostname
! whoami
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

```
DESKTOP-6HAS1BN
desktop-6has1bn\medra
C:\Users\medra\anaconda3\python.exe
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
```

Now, let's get to work!

First lets re-hash (huge pun here!) the preceeding lesson, but this time with feeling! (Kentucky Fried Movie?)

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:

In [51]:
```
TimeOfConcentration = 0.0
print(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).

In [52]:
```
TimeOfConcentration += 5
print(TimeOfConcentration)
```

```
5.0
```

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.

In [53]:
```
# 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.

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:

In [54]:
```python
# 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.

What's with the hash symbol # ?

> Comments in a code cell are added by writing a hashtag symbol (#) followed by any text of your choice. Any text that follows the hashtag symbol on the same line is ignored by the Python interpreter. In a Markdown cell the symbol is used to delimit headings. In a raw cell its just a character.

Its a pretty common comment symbol, some languages use any letter in the first column (FORTRAN) which allows something called conditional compilation, other symbols are:

- % Tex/LaTex
- ** Javascript
- <!-- --!> in HTML and Javascript
- = = PERL
- / / in C

About Commenting

When you are writing a program/notebook, your intent is always clear and self-evident to you. Odd thing, a month later, when you return to the program, it is quite confusing and unclear. No one is certain how the confusion creeps into a program, but it nearly always does.

To fight the bafflement, and to help yourself others understand your code, you need to use comments. Comments are text that is ignored by the interpreter, but that can inform the reader of what you are doing at any particular point in your program.

Some recommend writing comments at the top of each function, explaining what the function does and what values it returns. Functions should be named so that little ambiguity exists about what they do, and confusing and obscure bits of code should be redesigned and rewritten so as to be self-evident. Comments should not be used to clarify obscure code; instead, fix the code. In short, you should write your code well, and use comments to supplement understanding.

Comments that state the obvious are counterproductive because the code might change and the programmer might neglect to update the comment. What is obvious to one person might be obscure to another, however, so judgment is required when adding comments. The bottom line is that **comments** should not say what is happening, they **should say why it is happening**.

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. xy)
+=	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

In [55]:
```
# 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
```

In [56]:
```
# 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
```

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

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
```

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
```

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).

In [57]:
```
# 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 ' int
print("All about concatenation!")
print("A Silly String : ",MyCatName+MyName+DustyMassInKilos)
print("A SILLY STRING :  ", (MyCatName+MyName+DustyMassInKilos).upper())
print(MyName[0:4])    # Notice how the string is sliced- This is Python: ALWAYS start c
```

```
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
Theo
```

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.

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.

In [58]:
```
# Type Casting Examples
MyInteger = 234
MyFloat = 876.543
MyString = 'What is your name?'
MyOtherString = "5"
print(MyInteger,MyFloat,MyString)
print('Integer as float',float(MyInteger))
print('Float as integer',int(MyFloat))
print('Float as integer',int(MyOtherString))
print('Integer as string',str(MyInteger))
print('Integer as hexadecimal',hex(MyInteger))
print('Integer Type',type((MyFloat)))  # insert the hex conversion and see what happens
```

```
234 876.543 What is your name?
Integer as float 234.0
Float as integer 876
Float as integer 5
Integer as string 234
Integer as hexadecimal 0xea
Integer Type <class 'float'>
```

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 8) = float 15
```

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`

In [59]:
```
# Expressions Example
x1 = 7 + 3 * 6 / 2 - 1  # Change / into // and see what happens!
print(type(x1),x1)
## Simple I/O (Input/Output)
```

```
<class 'float'> 15.0
```

Example: Simple Input/Output

Here we introduce some I/O we will spend more time in later lessons on more elaborate constructs, but this gets us away from explicit assignment and lest us have some script interactions

Get two floating point numbers via the `input()` function and store them under the variable names `float1` and `float2`. Then, compare them, and try a few operations on them!

```
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`.

In [60]:
```
float1 = input("Please enter float1: ")
float2 = input("Please enter float2: ")
```

In [61]:
```
print("float1:", float1)
print(type(float1))
print("float2:", float2)
print(type(float2))
```

```
float1: 5.0
<class 'str'>
float2: 3.14159
<class 'str'>
```

In [62]:
```
float1 = float(float1)
float2 = float(float2)
```

```
In [63]: print("float1:", float1)
         print(type(float1))
         print("float2:", float2)
         print(type(float2))

float1: 5.0
<class 'float'>
float2: 3.14159
<class 'float'>
```

```
In [64]: float1>float2

Out[64]: True
```

```
In [65]: float1+float2

Out[65]: 8.14159
```

```
In [66]: float1/float2

Out[66]: 1.5915507752443827
```

Readings

Here are some great reads on this topic:

- "Variables in Python" by John Sturtz available at https://realpython.com/python-variables/
- **"A Beginner's Guide To Python Variables"** by **Avijeet Biswal** available at https://www.simplilearn.com/tutorials/python-tutorial/python-variables
- **"A Very Basic Introduction to Variables in Python"** by **Dr. Python** available at https://medium.com/@doctorsmonsters/a-very-basic-introduction-to-variables-in-python-4231e36dac52

Here are some great videos on these topics:

- **"Python Tutorial for Absolute Beginners #1 - What Are Variables?"** by **CS Dojo** available at https://www.youtube.com/watch?v=Z1Yd7upQsXY
- **"#4 Python Tutorial for Beginners | Variables in Python"** by **Telusko** available at *https://www.youtube.com/watch?v=TqPzwenhMj0
- **"Variables and Types in Python"** by **DataCamp** available at *https://www.youtube.com/watch?v=OH86oLzVzzw