LAB # 1: Introduction to Anaconda/Python.

Anaconda is a FREE enterprise-ready Python distribution for data analytics, processing, and scientific computing. Anaconda comes with Python 2.7 or Python 3.4 and 100+ cross-platform tested and optimized Python packages. All of the usual Python ecosystem tools work with Anaconda.

Additionally, Anaconda can create custom environments that mix and match different Python versions (2.6, 2.7, 3.3 or 3.4) and other packages into isolated environments and easily switch between them using conda, our innovative multi-platform package manager for Python and other languages.

INSTALLATION

Anaconda installs cleanly into a single directory, does not require Administrator or root privileges, does not affect other Python installs on your system, or interfere with OSX Frameworks.

System Requirements

System Requirements		
Linux	Windows	Mac OSX
32/64 bit Intel processor	32/64 bit Intel processor	64-bit Intel processor

Download Anaconda

Linux / Mac OS X command-line

After downloading the installer, in the shell execute, bash <downloaded file> Mac OS X (graphical installer)

After downloading the installer, double click the .pkg file and follow the instructions on the screen.

Windows

After downloading the installer, double click the .exe file and follow the instructions on the screen. Detailed Anaconda Installation Instructions are available at

http://docs.continuum.io/anaconda/install.html

Why Python?

1. Very Simple Language to learn.

   Get data (simulation, experiment control), Manipulate and process data, visualize results, quickly to understand, but also with high quality figures, for reports or publications.

2. Include best packages for Artificial Intelligence.

   Matplotlib, scikitlearn, scipy, numpy and tenserflow etc.

3. Extensively used in industry.

   General purpose Language,

How does Python compare to other solutions?

Compiled languages: C, C++, Fortran…

Pros: Very fast. For heavy computations, it's difficult to outperform these languages.

Cons: Painful usage: no interactivity during development, mandatory compilation steps, verbose syntax, manual memory management. These are difficult languages for non-programmers.

Matlab scripting language

Pros: Very rich collection of libraries with numerous algorithms, for many different domains. Fast execution because these libraries are often written in a compiled language. Pleasant development environment: comprehensive and help, integrated editor, etc. Commercial support is available.

Cons: Base language is quite poor and can become restrictive for advanced users. Not free.

Julia

Pros: Fast code, yet interactive and simple. Easily connects to Python or C.

Cons: Ecosystem limited to numerical computing. Still young.

Other scripting languages: Scilab, Octave, R, IDL, etc.

Pros: Open-source, free, or at least cheaper than Matlab. Some features can be very advanced (statistics in R, etc.)

Cons: Fewer available algorithms than in Matlab, and the language is not more advanced. Some software are dedicated to one domain. Ex: Gnuplot to draw curves. These programs are very powerful, but they are restricted to a single type of usage, such as plotting.

Python

Pros: Very rich scientific computing libraries. Well thought out language, allowing to write very readable and well structured code: we "code what we think". Many libraries beyond scientific computing (web server, serial port access, etc.). Free and open-source software, widely spread, with a vibrant community. A variety of powerful environments to work in, such as IPython, Spyder, Jupyter notebooks, Pycharm.

Cons: Not all the algorithms that can be found in more specialized software or toolboxes.

Starting Python

Start the Ipython shell (an enhanced interactive Python shell): by typing "ipython" from a Linux/Mac terminal, or from the Windows cmd shell, or by starting the program from a menu, e.g. in the Python(x,y) or EPD menu if you have installed one of these scientific-Python suites.

If you don't have Ipython installed on your computer, other Python shells are available, such as the plain Python shell started by typing "python" in a terminal, or the Idle interpreter. However, we advise to use the Ipython shell because of its enhanced features, especially for interactive scientific computing.

SPYDER (The Scientific Python Development Environment.)

Spyder is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

Python Keyword.

Keywords are the reserved words in python. We can't use a keyword as variable name, function name or any other identifier. Keywords are case sensitive.

```
import keyword
print(keyword.kwlist)
print("\nTotal number of keywords ", len(keyword.kwlist))
```

```
In [3]: runfile('C:/Users/ALI/.spyder-py3/temp.py', wdir='C:/Users/ALI/.spyder-py3')
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global',
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']

Total number of keywords  35
```

Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

Rules for Writing Identifiers:

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.

```
abc_12=12
```

```
In [7]: 1a=1
  File "<ipython-input-7-fbf1bec92ff8>", line 1
    1a=1
     ^
SyntaxError: invalid syntax
```

```
In [5]: global=1
  File "<ipython-input-5-89339e2d91ea>", line 1
    global=1
           ^
SyntaxError: invalid syntax
```

We cannot use special symbols like !, @, #, $, % etc. in our identifier.

```
In [6]: abc@=12
Traceback (most recent call last):

  File "<ipython-input-6-418b82dab90c>", line 1, in <module>
    abc@=12

NameError: name 'abc' is not defined
```

Python Comments

Comments are lines that exist in computer programs that are ignored by compilers and interpreters. Including comments in programs makes code more readable for humans as it provides some information or explanation about what each part of a program is doing.

In general, it is a good idea to write comments while you are writing or updating a program as it is easy to forget your thought process later on, and comments written later may be less useful in the long term. In Python, we use the hash (#) symbol to start writing a comment.

```
#Print Hello, world to console
print("Hello, world")
```

Multi Line Comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line.

```
#This is a long comment
#and it extends
#Multiple lines
```

Another way of writing multiline is to use triple quotes, either ''' or """.

```
"""This is also a
perfect example of
multi-line comments"""
```

```
In [7]: """This is also a
   ...: perfect example of
   ...: multi-line comments"""
Out[7]: 'This is also a\nperfect example of\nmulti-line comments'
```

Python Indentation

1. Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.
2. A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.
3. Generally, four whitespaces are used for indentation and is preferred over tabs.

```
for i in range(10):
    print (i)
```

```
In [3]: for i in range(10):
   ...:     print(i)
   ...:
0
1
2
3
4
5
6
7
8
9

In [4]:
```

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable.

```
if True:
    print "Machine Learning"
    c = "AAIC"

if True: print "Machine Learning"; c = "AAIC"
```

Python Statement

Instructions that a Python interpreter can execute are called statements.

Examples:

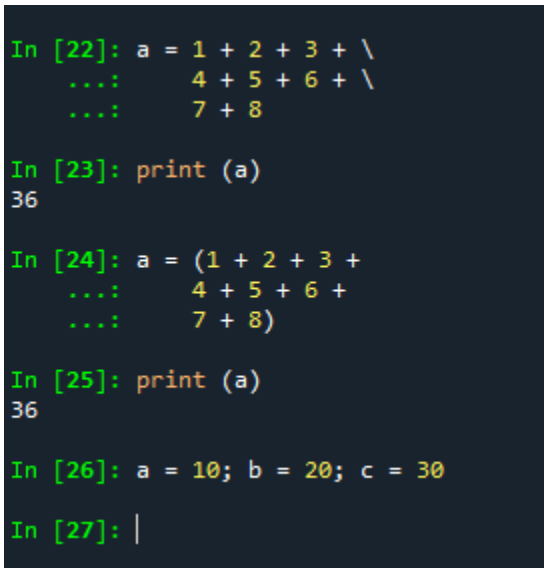
```
a = 1  #single statement
```

Multi-Line Statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character ().

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8
print (a)

#another way is
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8)
print a
a = 10; b = 20; c = 30
#put multiple statements in a single line using ;
```



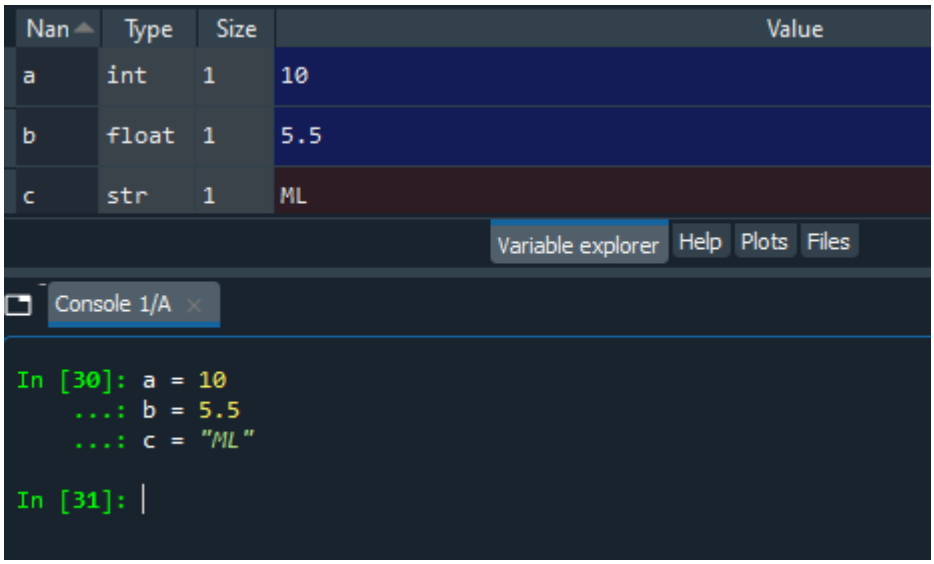
Variables

A variable is a location in memory used to store some data (value). They are given unique names to differentiate between different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable Assignments

```
#We use the assignment operator (=) to assign values to a variable
a = 10
b = 5.5
c = "ML"
```



Multiple Assignments

```
a, b, c = 10, 5.5, "ML"
a = b = c = "AI" #assign the same value to multiple variables at once
```

Storage Locations

```
x = 3
print(id(x))                #print address of variable x

y = 3
print(id(y))                #print address of variable y
```

Observation:
x and y points to same memory location

```
y = 2
print(id(y))                    #print address of variable y
```

```
In [32]: x = 3
    ...: print(id(x))
140710114441056

In [33]: y = 3
    ...: print(id(y))
140710114441056

In [34]: y = 2
    ...: print(id(y))
140710114441024
```

Data Types

Every value in Python has a datatype. Since everything is an object in Python programming, data types are classes and variables are instance (object) of these classes.

Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

```
a = 5                   #data type is implicitly set to integer
print(a, " is of type", type(a))
a = 2.5                 #data type is changed to float
print(a, " is of type", type(a))
```

```
In [1]: a = 5               #data type is implicitly set to integer
   ...: print(a, " is of type", type(a))
5  is of type <class 'int'>

In [2]: a = 2.5             #data type is changed to float
   ...: print(a, " is of type", type(a))
2.5  is of type <class 'float'>
```

```
a = 1 + 2j              #data type is changed to complex number
print(a, " is complex number?")
print(isinstance(1+2j, complex))
```

```
In [3]: a = 1 + 2j            #data type is changed to complex number
   ...: print(a, " is complex number?")
   ...: print(isinstance(1+2j, complex))
(1+2j)  is complex number?
True
```

Boolean

Boolean represents the truth values False and True

```
a = True                                #a is a boolean type

print(type(a))
```

```
In [4]: a = True                        #a is a boolean type
   ...: print(type(a))
<class 'bool'>
```

Conversion between Datatypes

```
>>> float(1)
1.0
```

Integer division

In Python2:

```
>>> 3 / 2
1
```

In Python 3:

```
>>> 3 / 2
1.5
```

To be safe: use floats:

```
>>> 3 / 2.
1.5
>>> a = 3
>>> b = 2
>>> a / b # In Python 2
1
>>> a / float(b)
1.5
```

Future behavior: to always get the behavior of Python3

```
>>> 3 / 2
1.5
```

If you explicitly want integer division use //:

```
>>> 3.0 // 2
1.0
```

The behavior of the division operator has changed in Python 3.

Lab Tasks:

1. Print your name and Reg No.
2. Take two variables and perform different mathematical operations on them and also use type
3. Write a Python program to print the following string in a specific format (see the output).
   Sample String : "Twinkle, twinkle, little star, How I wonder what you are! Up above the world so high, Like a diamond in the sky. Twinkle, twinkle, little star, How I wonder what you are"

   Output :

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.
Twinkle, twinkle, little star,
How I wonder what you are

4. Write a Python program to get the Python version you are using
5. Write a Python program to display the current date and time.
   Sample Output :
   Current date and time :
   2014-07-05 14:34:14
6. Write a Python program to display the examination schedule. (extract the date from exam_st_date).
   exam_st_date = (11, 12, 2014)
   Sample Output : The examination will start from : 11 / 12 / 2014
7. Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn.
   Sample value of n is 5
   Expected Result : 615
8. Write a Python program which accepts the radius of a circle from the user and compute the area.
   Sample Output :
   r = 1.1
   Area = 3.8013271108436504
9. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.
10. Write a Python program to print the calendar of a given month and year.
    Note : Use 'calendar' module.