## OCNC 2019 Introductory Session for Biologists:

# Introduction to Python

2019.6.24 by Kenji Doya

Python is a programming language developed in 1990s by Guido van Rossum. Its major features are:

- consice -- (relatively) easy to read
- extensible -- (so-called) object oriented
- free! -- unlike Matlab

It was originally used for "scripting" sequences of processing. Now it is widely used for scientific computing as well.

## Installing Python

Most Linux and Mac machines usually have Python pre-installed.
To install and setup a variety of libraries, it is the best to install a curated distribution, such as:

- Anaconda: http://anaconda.com (http://anaconda.com)
- Canopy: https://www.enthought.com/product/canopy/ (https://www.enthought.com/product/canopy/)

Currently there are two popularly versions: Python 2.7 and 3.6.
Unless you need to use a library that has not been updated, use Python 3.6.

## Starting Python

From a terminal, type

```
$ python
```

to start a python interpreter.

## Python as a calculator

At the python prompt >>>, try typing numbers and operators, like

```
>>> 1+1
```

```
In [1]: 1+1
```

```
Out[1]: 2
```

```
In [2]: 2**8
```

```
Out[2]: 256
```

```
In [3]: exp(2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-2f1428f27603> in <module>()
----> 1 exp(2)

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

The plain Python does not include math functions. You need to import numpy.

## Jupyter Notebook

For building a program step-by-step with notes and results attached, it is highly recommended to use a notebook interface, such as Jupyter Notebook (https://jupyter.org (https://jupyter.org)), which is included in Anaconda and other popular packages.

To start Jupyter Notebook type in the terminal

```
$ jupyter notebook
```

which should open a web page showing your working directory.

You can create a new notebook from the New menu on the upper right corner, or open an existing .ipynb file like this.

## Working with the notebook

A notebook is made of "cells."

You can make a new cell by "+" button on the Toolbar, or by typing `ESC+A` (above) or `ESC+B` (below).

You can make a cell as Markdown (documentation) by `ESC+M`, as Code by `ESC+Y`, or simply by the Toolbar menu.
You can delete a cell by `ESC+DD`, or the "Cut" button on the Toolbar.

## Markdown cell

Markdown is a simple text formatting tool, with
`#, ##, ###,...` for headings
`*, +, -,...` for bullets
`$$` for Latex symbols like $\pi$ or $\sum_{i=1}^n x_i^2$
and two spaces at the end of line for a line break.

See https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) for details.

You can format a Markdown cell by `Shift+Return`, and go back to Edit mode by `Return`.

## Code cell

You can type `Control+Return` to run the cell or `Shift+Return` to run and move to the next cell.
You can also use the triangle button or "Cell" menu to run cells.

A line after # is neglected as a comment.

# Getting help

In [4]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=Fals
e)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.
stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

In [5]:
```
# ? opens a help window
print?
```

In [6]:
```
help()  # This starts an interactive help.
```

```

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely che
ck out
the tutorial on the Internet at https://docs.python.org/3.6/tutori
al/. (https://docs.python.org/3.6/tutorial/.)

Enter the name of any module, keyword, or topic to get help on wri
ting
Python programs and using Python modules.  To quit this help utili
ty and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics,
type
"modules", "keywords", "symbols", or "topics".  Each module also c
omes
with a one-line summary of what it does; to list the modules whose
```

You can also check a variety of references from `Help` menu in the top of the page.

In [ ]:
```

```

## Variables

You can assing a number or result of computation to a variable.

```
In [7]: a = 1
```

```
In [8]: a
```

```
Out[8]: 1
```

```
In [9]: b = a + a
        b
```

```
Out[9]: 2
```

Multiple variables can be assigned at once.

```
In [10]: a, b = 1, 2
         print(a, b)
```

```
1 2
```

```
In [ ]:
```

## Lists

You can create a list by surrounding items by [ ].

```
In [11]: b = [1, 2, 3, 4]
         b
```

```
Out[11]: [1, 2, 3, 4]
```

```
In [12]: b[0]
```

```
Out[12]: 1
```

An item can be referenced by [ ], with index starting from 0.

```
In [13]: b[3]
```

```
Out[13]: 4
```

```
In [14]: b[-1]

Out[14]: 4
```

For lists, + means concatenation

```
In [15]: b + b

Out[15]: [1, 2, 3, 4, 1, 2, 3, 4]
```

A colon can be used for indexing a part of list.

```
In [16]: b[1:3]

Out[16]: [2, 3]

In [17]: b[:3]

Out[17]: [1, 2, 3]

In [18]: b[1:]

Out[18]: [2, 3, 4]
```

A list can contain different types of itmes with different lengths.

```
In [19]: a = [1, 2, 3.14, 'apple', "orange", [1, 2]]
         a

Out[19]: [1, 2, 3.14, 'apple', 'orange', [1, 2]]

In [ ]:
```

## Numpy

`numpy` provides `ndarray` data format suitable for numeric arrays, such as vectors and matrices.

This is the convention for importing `numpy`.

```
In [20]: import numpy as np
```

You can create a numpy array from a list by `array()` function.

```
In [21]: b = np.array([1,2,3])
         b

Out[21]: array([1, 2, 3])
```

Index starts from zero

```
In [22]: b[1]

Out[22]: 2
```

Operators work component-wise.

```
In [23]: b + b

Out[23]: array([2, 4, 6])

In [24]: b * b

Out[24]: array([1, 4, 9])

In [25]: b + 1  # broadcast

Out[25]: array([2, 3, 4])
```

`arange()` gives an evenly spaced array.

```
In [26]: np.arange(10)

Out[26]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [27]: np.arange(0, 10, 0.5)

Out[27]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6
         . ,
                6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5])
```

`linspace()` gives an array *including* the last point.

```
In [28]: np.linspace(0, 10, num=10)

Out[28]: array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.444444
         44,
                 5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.
         ])
```

```
In [ ]:
```

## Matrix by nested array

You can make a matrix as a nested array.

```
In [29]: A = np.array([[1,2,3], [4,5,6]])
         A

Out[29]: array([[1, 2, 3],
                [4, 5, 6]])
```

Components can be accessed by [ ][ ] or [ , ]

```
In [30]: A[1][1]

Out[30]: 5
```

```
In [31]: A[1,1]

Out[31]: 5
```

Take the first row

```
In [32]: A[0]

Out[32]: array([1, 2, 3])
```

```
In [33]: A[0,:]

Out[33]: array([1, 2, 3])
```

Take the second column

```
In [34]: A[:,1]

Out[34]: array([2, 5])
```

`.T` gives the transpose matrix

```
In [35]: A.T
```

```
Out[35]: array([[1, 4],
                [2, 5],
                [3, 6]])
```

Component-wise arithmetics

```
In [36]: A + A
```

```
Out[36]: array([[ 2,  4,  6],
                [ 8, 10, 12]])
```

```
In [37]: A * A
```

```
Out[37]: array([[ 1,  4,  9],
                [16, 25, 36]])
```

For matrix product, you can use `@` operator in Python 3, or use `np.dot()` function

```
In [38]: A @ A.T
```

```
Out[38]: array([[14, 32],
                [32, 77]])
```

```
In [39]: np.dot(A, A.T)
```

```
Out[39]: array([[14, 32],
                [32, 77]])
```

```
In [ ]: 
```

## Creating common matrices

```
In [40]: np.zeros([2,3])
```

```
Out[40]: array([[0., 0., 0.],
                [0., 0., 0.]])
```

```
In [41]: np.eye(2)  # Identity matrix
```

```
Out[41]: array([[1., 0.],
                [0., 1.]])
```

In [42]:
```
np.empty([2,3])  # contents are not initialized
```

Out[42]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [ ]:
```

```

## Matplotlib

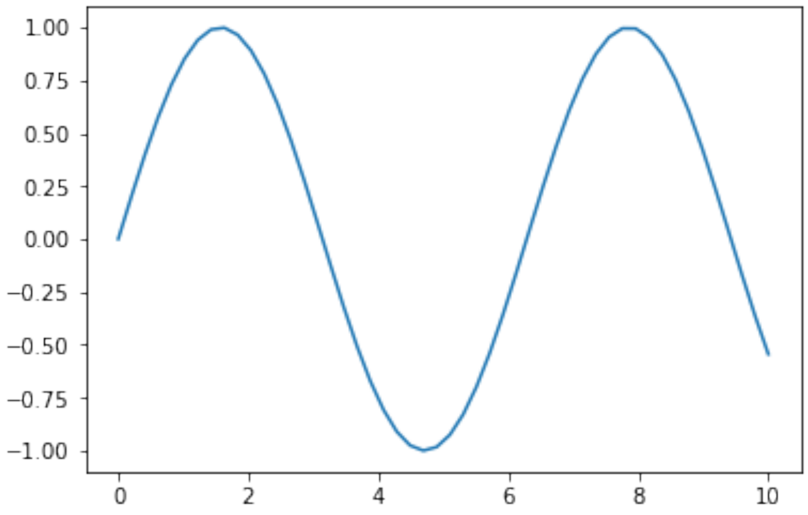
Matplotlib is the standard graphics package for Python. It mimics many graphics functions of MATLAB.

`%matplotlib inline` places plots inside the notebook.

In [43]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [44]:
```
x = np.linspace(0, 10)
y = np.sin(x)
plt.plot(x, y)
```

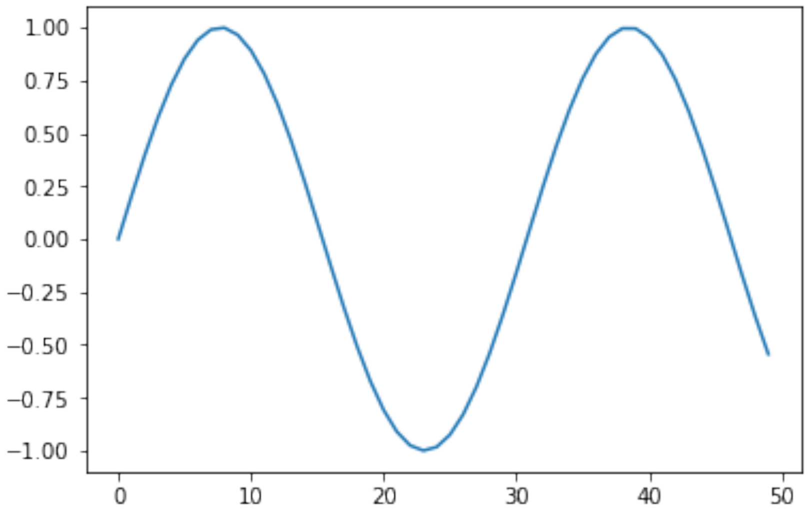
Out[44]:
```
[<matplotlib.lines.Line2D at 0x117dbd780>]
```



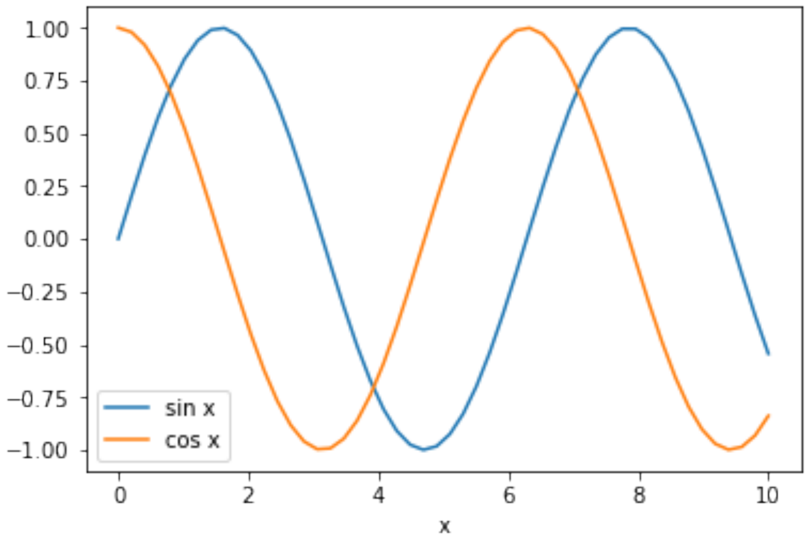
The Matplotlib gallery (http://matplotlib.org/gallery.html (http://matplotlib.org/gallery.html)) illustrates variety of plots to choose from

In [45]:
```
# If x is omitted, it is assumed to be 0, 1, 2,...
plt.plot(y)
```

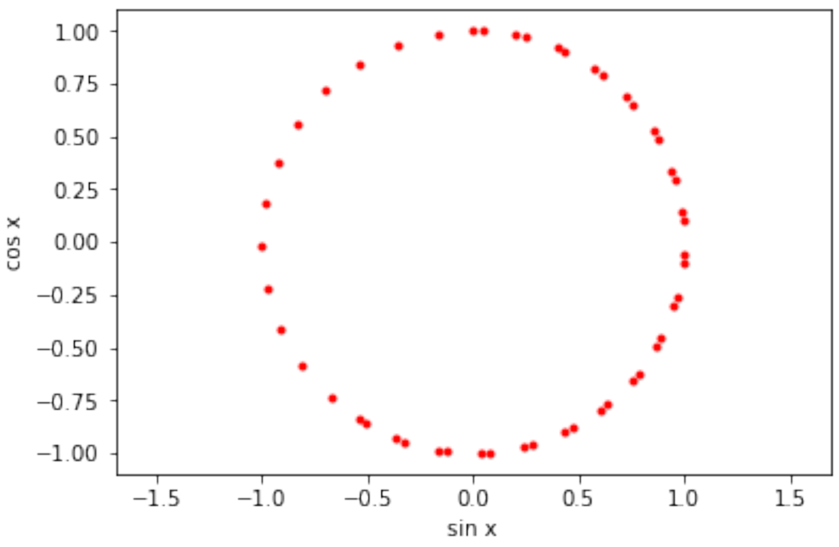
Out[45]:
```
[<matplotlib.lines.Line2D at 0x117e636d8>]
```



In [46]:
```
# If Y is a matrix, each column is plotted
z = np.cos(x)
plt.plot(x, np.c_[y, z])  # combine vectors as columns
# And it is always good to add axis labels and legends
plt.xlabel("x")
plt.legend(("sin x", "cos x")); # omit <matplotlib....> output by
```



In [47]:
```
# Phase space plot often gives us good intuition
plt.plot(y, z, "r.")  # by red dots
plt.axis("equal")  # equal scaling for x and y
# By ";" you can omit output and put multiple statements in a line
plt.xlabel("sin x"); plt.ylabel("cos x");
```



In [ ]:
```

```