1. Introduction to Python

Computational Methods, Sept. 2017, 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 packages, 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 popular versions: Python 2.7 and 3.6.
Unless you must 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 cal
l 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.

In [ ]:
```

```

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 distributions.

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

- text

`$$` for Latex symbols like α, $\sum_i^n$
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.

In [ ]:

integer and floating-point numbers

In [4]:
```
type(1)
```

Out[4]:
```
int
```

In [5]:
```
type(1.)
```

Out[5]:
```
float
```

In Python 3, division of integers can produce a float.
In Python 2, it was truncated to an integer.

In [6]:
```
3 / 2  # 1.5 by Python 3; 1 by Python 2
```

Out[6]: 1.5

For compatibility and clarity, it is recommended to make explicit which you want.

In [7]:
```
# add a decimal point
3. / 2
```

Out[7]: 1.5

In [8]:
```
# use float() function
a = 3
b = 2
float(a)/b
```

Out[8]: 1.5

In [9]:
```
# integer division
3 // 2
```

Out[9]: 1

In [ ]:
```

```

Variables

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

In [10]:
```
a = 1
```

In [11]:
```
a
```

Out[11]: 1

In [12]:
```
b = a + a
b
```

Out[12]: 2

Multiple variables can be assigned at once.

In [13]:
```
a, b = 1, 2
print(a, b)
```

```
1 2
```

In [ ]:

Lists

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

In [14]:
```
b = [1, 2, 3, 4]
b
```

Out[14]: [1, 2, 3, 4]

In [15]:
```
b[0]
```

Out[15]: 1

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

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

Out[16]: 4

In [17]:
```
b[-1]
```

Out[17]: 4

For lists, + means concatenation

In [18]:
```
b + b
```

Out[18]: [1, 2, 3, 4, 1, 2, 3, 4]

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

In [19]:
```
b[1:3]
```

Out[19]: [2, 3]

In [20]:
```
b[:3]
```

Out[20]: [1, 2, 3]

In [21]:
```
b[1:]
```

Out[21]: [2, 3, 4]

You can create a nested list, like a matrix

In [22]:
```
A = [[1,2,3],[4,5,6]]
A
```

Out[22]:
```
[[1, 2, 3], [4, 5, 6]]
```

An item in a nested list can be picked by [ ][ ], but not [,]

In [23]:
```
A[1]
```

Out[23]:
```
[4, 5, 6]
```

In [24]:
```
A[1][2]
```

Out[24]:
```
6
```

In [25]:
```
A[1,2]
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-25-13df355f18af> in <module>()
----> 1 A[1,2]

TypeError: list indices must be integers or slices, not tuple
```

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

In [26]:
```
a = [1, 2, 3.14, 'apple', "orange", [1, 2]]
a
```

Out[26]:
```
[1, 2, 3.14, 'apple', 'orange', [1, 2]]
```

In [ ]:
```

```

When you assign a list to another list, only the pointer is copied.

In [27]:
```
a = [1, 2, 3]
b = a
b[1] = 4
a
```

Out[27]:
```
[1, 4, 3]
```

When you want to copy the content, use [:]

In [28]:
```
a = [1, 2, 3]
b = a[:]
b[1] = 4
a
```

Out[28]:
```
[1, 2, 3]
```

In [ ]:
```

```

`if` branch

Branching by `if` statement looks like this. In Python, indentation specifes where a block of code starts and ends.

In [29]:
```
x = 1
if x>0:
    y = x
else:
    y = 0
y
```

Out[29]:
```
1
```

There is a shorthand notaion with the condition in the middle:

In [30]:
```
x if x>0 else 0
```

Out[30]:
```
1
```

`for` loop

A common way of `for` loop is by `range()` function.
Don't forget a collon and indentation.

In [31]:
```
j = 0
for i in range(5):
    j = j + i
print(i, j)
```

```
4 10
```

You can specify start, end and interval.

In [32]:
```
for i in range(3,9,2):
    print(i)
```

```
3
5
7
```

`for` loop can also be over a list.

In [33]:
```
a = [1, 2, 3]
for x in a:
    print(x**2)
```

```
1
4
9
```

In [34]:
```
s = "hello"
for c in s:  # characters in a string
    print(c)
```

```
h
e
l
l
o
```

In [35]:
```
s = ["hello", "goodby"]
for c in s:  # strings in a list
    print(c)
```

```
hello
goodby
```

`enumerate()` function gives pairs of index and content of a list.

In [36]:
```
for i, c in enumerate(s):
    print(i, c)
```

```
0 hello
1 goodby
```

In [ ]:
```

```

List 'comprehension'

There is a quick way of constructing a list from a for loop.

In [37]:
```
y = [x**2 for x in range(5)]
y
```

Out[37]:
```
[0, 1, 4, 9, 16]
```

In [ ]:
```

```

Numpy arrays

In [38]:
```
import numpy as np
```

Numpy array is specialized for numbers

In [39]:
```
b = np.array([1, 2, 3])
b
```

Out[39]:
```
array([1, 2, 3])
```

Index starts from zero

In [40]:
```
b[1]
```

Out[40]:
```
2
```

Operators work component-wise.

In [41]:
```
b + b
```

Out[41]:
```
array([2, 4, 6])
```

In [42]:
```
b * b
```

Out[42]:
```
array([1, 4, 9])
```

In [43]:
```
b + 1  # broadcast
```

Out[43]:
```
array([2, 3, 4])
```

`arange()` gives an evenly spaced array.

In [44]:
```
np.arange(10)
```

Out[44]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [45]:
```
np.arange(0, 10, 0.5)
```

Out[45]:
```
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 [46]:
```
np.linspace(0, 10, num=10)
```

Out[46]:
```
array([  0.        ,   1.11111111,   2.22222222,   3.33333333,
         4.44444444,   5.55555556,   6.66666667,   7.77777778,
         8.88888889,  10.        ])
```

In [ ]:
```

```

Nested array

You can make a matrix as a nested array.

In [47]:
```
A = np.array([[1,2],[3,4]])
A
```

Out[47]:
```
array([[1, 2],
       [3, 4]])
```

Components can be accessed by [,]

In [48]:
```
A[1][1]
```

Out[48]:
```
4
```

In [49]:
```
A[1,1]
```

Out[49]:
```
4
```

Take the first row

In [50]:
```
A[0]
```

Out[50]:
```
array([1, 2])
```

In [51]:
```
A[0,:]
```

Out[51]:
```
array([1, 2])
```

Take the second column

In [52]:
```
A[:,1]
```

Out[52]:
```
array([2, 4])
```

Component-wise arithmetics

In [53]:
```
A + A
```

Out[53]:
```
array([[2, 4],
       [6, 8]])
```

In [54]:
```
A * A
```

Out[54]:
```
array([[ 1,  4],
       [ 9, 16]])
```

In [ ]:
```

```

Common matrices

In [55]:
```
np.zeros([2,3])
```

Out[55]:
```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

In [56]:
```
np.eye(3)
```

Out[56]:
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

In [57]:
```
np.empty([3,2])
```

Out[57]:
```
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

In [ ]:
```

```

Tuples and sequences

A 'tuple' is a sequence like a list, but 'immutable', cannot change its content.

In [58]:
```
t = 1, 2, 'cat'
t
```

Out[58]:
```
(1, 2, 'cat')
```

Items in a tuple can be 'unpacked.'

In [59]:
```
a, b, c = t
print(a, b, c)
```

```
1 2 cat
```

In [ ]:
```

```

Sets and dictionaries

A *set* is an unordered collection of unique elements.

In [60]:
```
s = {1, 2, 'cat', 'dog', 2, 'cat'}
s
```

Out[60]:
```
{1, 2, 'dog', 'cat'}
```

In [61]:
```
'dog' in s
```

Out[61]:
```
True
```

A *dictionary* is a set of key:value pairs.

In [62]:
```
phone = {'kenji':8594, 'emiko':8824, 'kikuko':8976}
```

In [63]:
```
phone['kenji']
```

Out[63]:
```
8594
```

In [64]:
```
list(phone.keys())
```

Out[64]:
```
['kenji', 'emiko', 'kikuko']
```

In [ ]:
```

```

Text outputs

You can use `.format` for elaborate output.

In [65]:
```
print(1/7)
```

```
0.14285714285714285
```

In [66]:
```
x = 7
print('1/{0} is {1:0.5f}'.format(x, 1/x))
```

```
1/7 is 0.14286
```

In [ ]:
```

```

Magic functions

In Jupyter notebook (ipython), some *magic* functions preceded by `%` are available for working with the file system, etc.

In [67]:
```
# present working directory
%pwd
```

Out[67]:
```
'/Users/doya/Dropbox (OIST)/Python/ComputationalMethods'
```

In [68]:
```
# list the files
%ls
```

```
00_ComputationalMethods.html   Ex01/
00_ComputationalMethods.ipynb  Ex01_Introduction.ipynb
00_ComputationalMethods.pdf    Ex01_Introduction_draft.ipynb
01_Introduction.ipynb          Ex02_Visualization.ipynb
02_Visulaization.ipynb         Ex03_Matrix.ipynb
03_Matrix.ipynb                Ex03b_PCA.ipynb
04_Function.ipynb              Ex04_Function.ipynb
05_Iteration.ipynb             Ex04_Function_answers.ipynb
06_ODE.ipynb                   Ex05_Iteration.ipynb
07_PDE.ipynb                   Ex06_ODE.ipynb
08_Optimization.ipynb          Ex08_Optimization.ipynb
09_Sampling.ipynb              Ex09_Sampling.ipynb
10_Distributed.ipynb           Ex10_Distributed.ipynb
10_Distributed_code.zip        Ex11_Management.ipynb
11_Management.ipynb            Project.ipynb
2017/                          Projects/
A01_Introduction.ipynb         Stability.pptx
A02_Visualization.ipynb        SymPy.ipynb
A03_Matrix.ipynb               __pycache__/
A03b_PCA.ipynb                 datafile
A04_Function.ipynb             figures/
A05_Iteration.ipynb            first.py
A06_ODE.ipynb                  first2.py
A07_PDE.ipynb                  haisai.py
A09_Sampling.ipynb             iris.txt
A10_Distributed.ipynb          lp.py
A10_Distributed_code.zip       parallel_python/
ComplexEV.ipynb                random.c
DV01/                          random.ipynb
DifferentialEquations.ipynb    second.py
DynaView.ipynb                 second2.py
DynaView.py                    textfile
Evolutionary.ipynb*            vector.py
```

Check http://ipython.readthedocs.io/en/stable/interactive/magics.html (http://ipython.readthedocs.io/en/stable/interactive/magics.html) for the full list.

In [ ]:
```

```

Saving and loading data

You can use `open()`, `write()` and `read()` function to access data files.

In [69]:
```
with open('textfile', 'w') as f:
    f.write('Haisai!\n')
    f.write('Mensore!\n')
# f is closed when the `with` block is finished
```

In [70]:
```
with open('textfile', 'r') as f:
    s = f.read()
print(s)
```

```
Haisai!
Mensore!
```

JSON format

Python has `pickle` library to store variables in a binary format, which can be read only by Python.
JSON is a common data format that can be accesses by many languages.

In [71]:
```
import json
```

You can see data in json format

In [72]:
```
x = list(range(10))
json.dumps(x)
```

Out[72]:
```
'[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]'
```

And save in a file.

In [73]:
```
with open('datafile', 'w') as f:
    json.dump(x, f)
```

In [74]:
```
with open('datafile', 'r') as f:
    y = json.load(f)
y
```

Out[74]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A caveat is that a numpy array has to be converted to a list.

In [75]:
```
x = np.arange(0,10)
json.dumps(x.tolist())
```

Out[75]:
```
'[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]'
```

In [ ]:

Getting help

In [82]:
```
range?
```

In [*]:
```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start
(inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2,
..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0,
1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
```

```
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  __reduce__(...)
 |      helper for pickle
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(...)
 |      Return a reverse iterator.
 |
 |  count(...)
 |      rangeobject.count(value) -> integer -- return number of occu
rrences of value
 |
 |  index(...)
 |      rangeobject.index(value, [start, [stop]]) -> integer -- retu
rn index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

In [*]:
```
help()
```

```
Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check
out
the tutorial on the Internet at http://docs.python.org/3.6/tutorial/
. (http://docs.python.org/3.6/tutorial/.)

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

To get a list of available modules, keywords, symbols, or topics, ty
pe
"modules", "keywords", "symbols", or "topics".  Each module also com
es
with a one-line summary of what it does; to list the modules whose n
ame
or summary contain a given string such as "spam", type "modules spam
".

help>
```

Or use the 'Help' menu of the Jupyter Notebook.

In [ ]:
```

```

Exercise

Lists and loops

Create the following lists, possibly in multiple ways.

1) odd positive integers below 20.

In [ ]:
```

```

2) sums from 1 to n, for up to n=10.

In [ ]:
```

```

3) primer numbers below n=20.

In [ ]:

4) n=10 random numbers between 0 and k=5.
(use np.random.randint())

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

In [ ]:

5) from two lists of the same length, make a list with the larger of the items at the same position.

In [ ]:

6) from a random list with 10 items find the medium

In [ ]:

In [ ]:

Arrays and matrices

1) an m-by-n matrix with random integers from 0 to k.

In [ ]:

2) from a matrix, make a sub matrix of items in odd rows and odd columns.

In [ ]:

3) make a transpose of a matrix.

In [ ]:

4) make a m-by-n zero matrix with random integers from 0 to k in the diagonal

In [ ]:

In [ ]:

Dictionary and file

For saving multiple variables in a JSON file, it is customary to use a dictionary to store the name:value pairs.
1) make two arrays or matrices, register them in a dictionary, save to a JSON file, and check the file content.

In [ ]:

2) Read the file and restore the data.

In [ ]:

In [ ]:

In [ ]: