



Practicing Python 3

Mosky

Python?



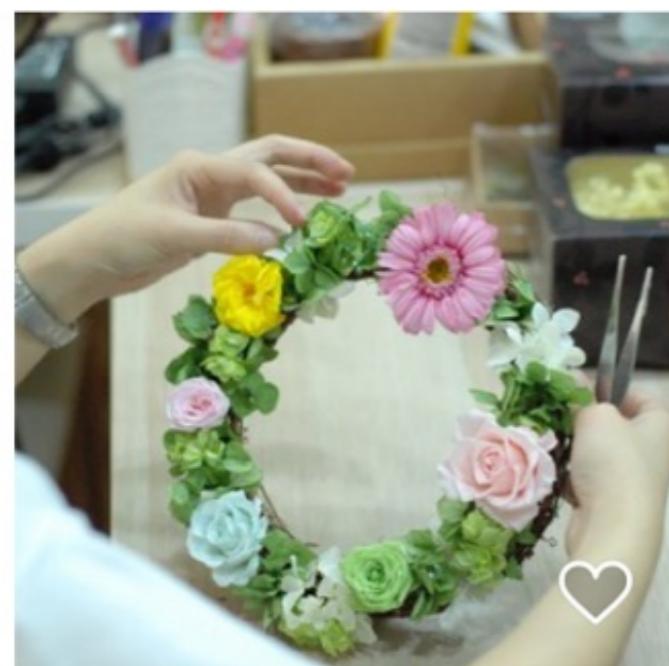
體驗活動

首頁

主題分館



編輯嚴選



15CM自由花圈花藝課程 不
凋花永生花 新北板橋

三星四季田園手作生活器皿
體驗—青花瓷拓繪

Websites

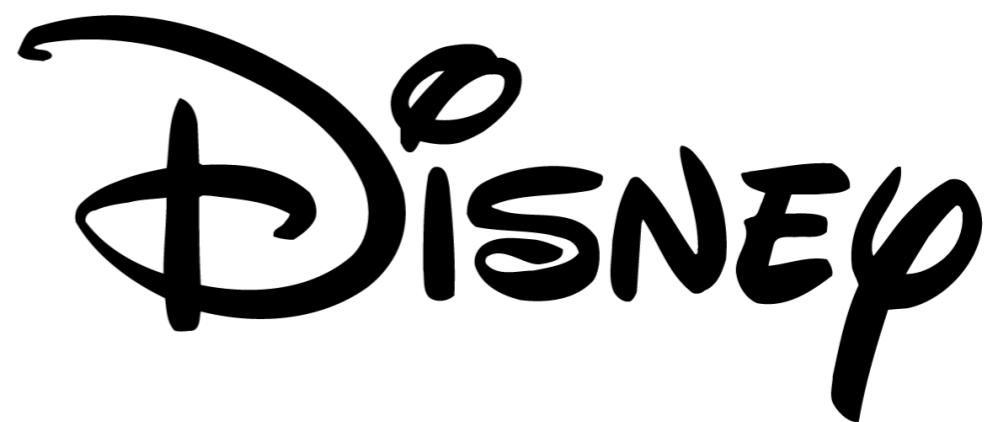
- Pinkoi
- Google search engine
- Pinterest
- Uber
 - 2016 MAU > 16M
- Instagram
 - 2017 MAU > 700M

Desktop Applications



Dropbox

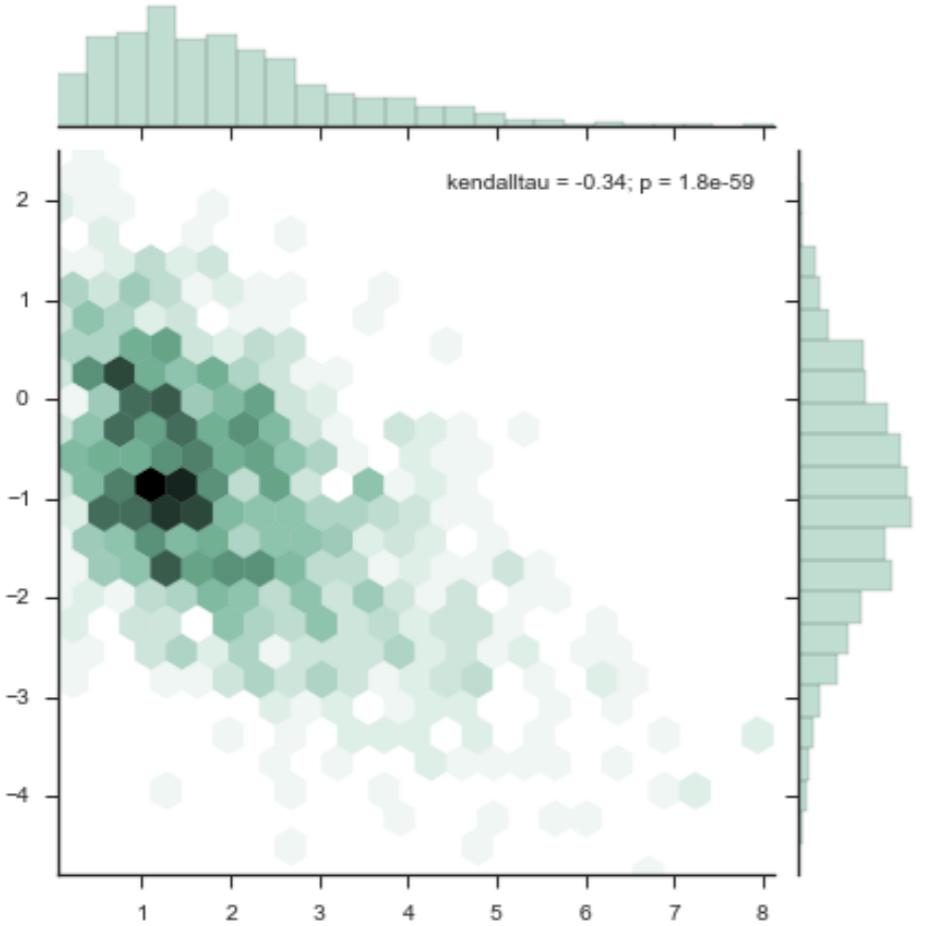
- Dropbox
- Disney
 - For animation studio tools.
- Blender
- A 3D graphics software.



Science

.....

- NASA
- LIGO
- 2016 Gravitational waves
- LHC
- 2013 Higgs boson
- MMTK
- Molecular Modeling Toolkit



Python source code: [\[download source: hexbin_marginals.py\]](#)

```
import numpy as np
from scipy.stats import kendalltau
import seaborn as sns
sns.set(style="ticks")

rs = np.random.RandomState(11)
x = rs.gamma(2, size=1000)
y = -.5 * x + rs.normal(size=1000)

sns.jointplot(x, y, kind="hex", stat_func=kendalltau, color="#4CB391")
```



Embedded System

- iRobot uses Python.
- Raspberry Pi supports.
- Linux has built-in Python.



Why?

.....

- Python is slower than C, Java,
- But much faster to write,
- And easy to speed up.
 - Numba | Cython
- Has the rich libraries.
- Emphasizes code readability.
 - Easier to learn.
 - Easier to co-work.
- “Time is money.”

Showcases

A Website in a Minute

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Symbolic Mathematics

```
from sympy import symbols  
from sympy import diff
```

```
from sympy import init_printing  
init_printing()  
  
from sympy import symbols  
x = symbols('x')  
x**2
```

x^2

```
diff(x**2)
```

$2x$

```
x = symbols('x')  
  
x**2  
diff(x**2)
```

Data Visualization

```
%matplotlib inline  
  
import matplotlib  
matplotlib.style.use('ggplot')  
  
import pandas as pd  
import numpy as np  
  
ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))  
ts = ts.cumsum()  
ts.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x109ba8128>
```



```
import pandas as pd  
import numpy as np  
  
ts = pd.Series(  
    np.random.randn(1000),  
    index=pd.date_range(  
        '1/1/2000', periods=1000  
    )  
)  
ts = ts.cumsum()  
  
ts.plot()
```



Mosky

- Python Charmer at Pinkoi.
- Has spoken at: PyCons in TW, MY, KR, JP, SG, HK, COSCUPs, and TEDx, etc.
- Countless hours on teaching Python.
- Own the Python packages like ZIPCodeTW.
- <http://mosky.tw/>

The Outline

- The Foundation Part:
 - Primitives
 - If & While
 - Composites
 - For, Try, Def
 - Common Functions
 - Input & Output
 - Command Line Arguments
- The Fascinating Part:
 - Yield
 - Comprehensions
 - Functional Tricks
 - Import Antigravity
 - With 8 notebooks!
 - Module & Package
 - Class
 - And the checkpoints!

Our Toolbox

We Will Use

(a terminal)

Master the machine.

Python 3

Not Python 2.

Jupyter Notebook

Learn Python with browsers.

Visual Studio Code

A full-featured source code editor.

(other libs)

Will be introduced in this slides.

On Mac

- Open a terminal:
 - Spotlight (Cmd-Space) / “terminal”
- Install Homebrew by executing the command:
 - `$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
- Execute the commands:
 - `$ brew install python3`
 - `$ pip3 install jupyter numpy scipy sympy matplotlib ipython pandas flask beautifulsoup4 requests seaborn statsmodels scikit-learn`
- Note the above commands are just a single line.

On Mac

- Open a terminal:

Hint to talk to macOS. Enter
the command without \$.

e) / “terminal”

- Executing the command:

```
➤ $ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

- Execute the commands:

```
➤ $ brew install python3
```

```
➤ $ pip3 install jupyter numpy scipy sympy matplotlib
ipython pandas flask beautifulsoup4 requests seaborn
statsmodels scikit-learn
```

- Note the above commands are just a single line.

On Windows

- Install Python 3 with Miniconda:
 - <http://conda.pydata.org/miniconda.html>
 - Python 3.6 / Windows / 64-bit (exe installer)
- Open Anaconda's terminal:
 - Start Menu / Search / Type “Anaconda Prompt”
 - Right-click the item and choose “Run as administrator”.
- Execute the commands:
 - > conda install jupyter numpy scipy sympy matplotlib ipython pandas flask beautifulsoup4 requests seaborn statsmodels scikit-learn
- Note the above commands are just a single line.

On Windows

- Install Python 3 with Miniconda:
 - <http://conda.pydata.org/miniconda.html>
 - Python 3.6 / Windows / 64-bit (exe installer)
- Open Anaconda's terminal:
 - Start Menu / Search / Type “Anaconda Prompt”

Hint to talk to Windows. Enter the command without >.
 - > conda install jupyter numpy scipy sympy matplotlib ipython pandas flask beautifulsoup4 requests seaborn statsmodels scikit-learn
- Note the above commands are just a single line.

If You Already Have Python

- Try to install the packages:
 - The Jupyter Notebook
 - jupyter
 - The SciPy Stack
 - numpy scipy sympy matplotlib ipython pandas
 - The web-related libs
 - flask beautifulsoup4 requests
 - The data-related libs
 - seaborn statsmodels scikit-learn
- If fail, *clean uninstall* Python, and follow the previous slides.

Common MS-DOS Commands

C:	Go C: drive.
cd PATH	Change directory to <i>PATH</i> . <i>PATH</i> can be <i>/Users/python/</i> , <i>python/</i> , etc.
dir	Display the contents of a directory.
cls	Clear the terminal screen.
python PATH python3 PATH	Execute a Python program.
exit	Exit the current command processor.

Common Unix-like (Mac) Commands

<code>cd PATH</code>	Change Directory to <i>PATH</i> . <i>PATH</i> can be <code>/Users/python/</code> , <code>python/</code> , etc.
<code>ls</code>	List the contents of a directory.
<code><Ctrl-L></code>	Clear the terminal screen.
<code>python PATH</code> <code>python3 PATH</code>	Execute a Python program.
<code><Ctrl-D></code>	Exit the current terminal.

Common Terminal Shortcuts

<Tab>

Complete the command.

<Up>

Show the last command.

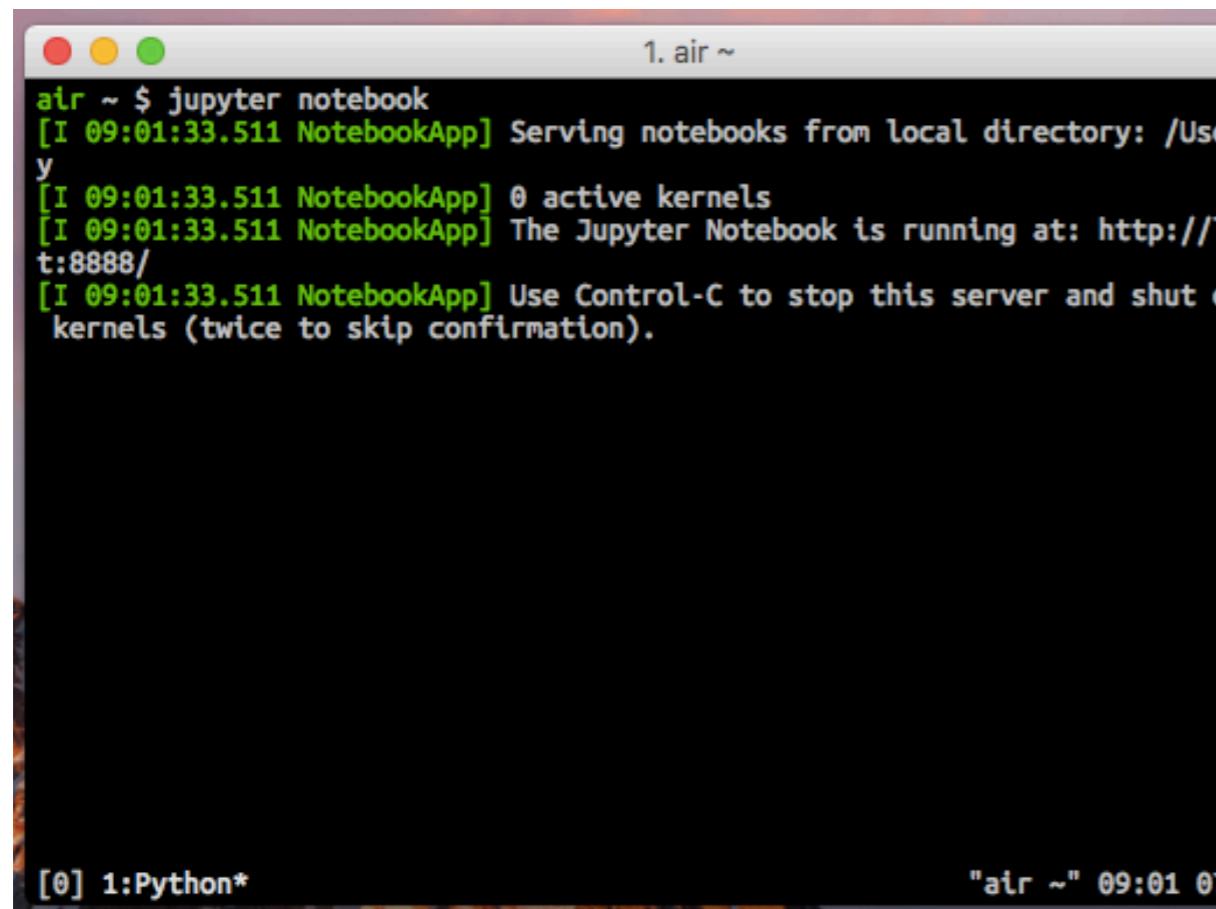
<Down>

Show the next command.

<Ctrl-C>

Enter new command, or
interrupt a running program.

Start Jupyter Notebook



A terminal window titled "1. air ~" displaying the command "air ~ \$ jupyter notebook". The output shows the notebook is serving from the local directory, has 0 active kernels, and is running at http://127.0.0.1:8888/. It also indicates to use Control-C to stop the server and shut down all kernels (twice to skip confirmation). The terminal prompt "[0] 1:Python*" is visible at the bottom.

```
air ~ $ jupyter notebook
[I 09:01:33.511 NotebookApp] Serving notebooks from local directory: /Users/airy
[I 09:01:33.511 NotebookApp] 0 active kernels
[I 09:01:33.511 NotebookApp] The Jupyter Notebook is running at: http://127.0.0.1:8888/
[I 09:01:33.511 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[0] 1:Python* "air ~" 09:01 07
```

- Mac:
 - \$ jupyter notebook
- Windows:
 - Search / "Jupyter Notebook"

Install Visual Studio Code

- If you already have a source code editor, it's okay to skip.
- Install:
 - <https://code.visualstudio.com/download>
- Open an integrated terminal:
 - Visual Studio Code / View / Integrated Terminal
- Execute a Python program:
 - \$ cd PROJECT_DIR
 - \$ python hello.py
 - or
 - \$ python3 hello.py

Hello, Python!

Checkpoint: Say Hi to Python

```
print('Hello, Python!')
```

Checkpoint: Say Hi to Python – on a Notebook

- Type the code into a notebook's cell.
- Press *<Ctrl-Enter>*.
- The output should be “Hello, Python!”

Checkpoint: Say Hi to Python – on an Editor

- Save a “hello.py” file into your project folder.
- Write the code into the file.
- Open up a new terminal, or use the integrated terminal.
- Change directory (`$ cd ...`) to your project folder.
- Execute (`$ python ...` or `$ python3 ...`) the file.
- The output should be “Hello, Python!”

Common Jupyter Notebook Shortcuts

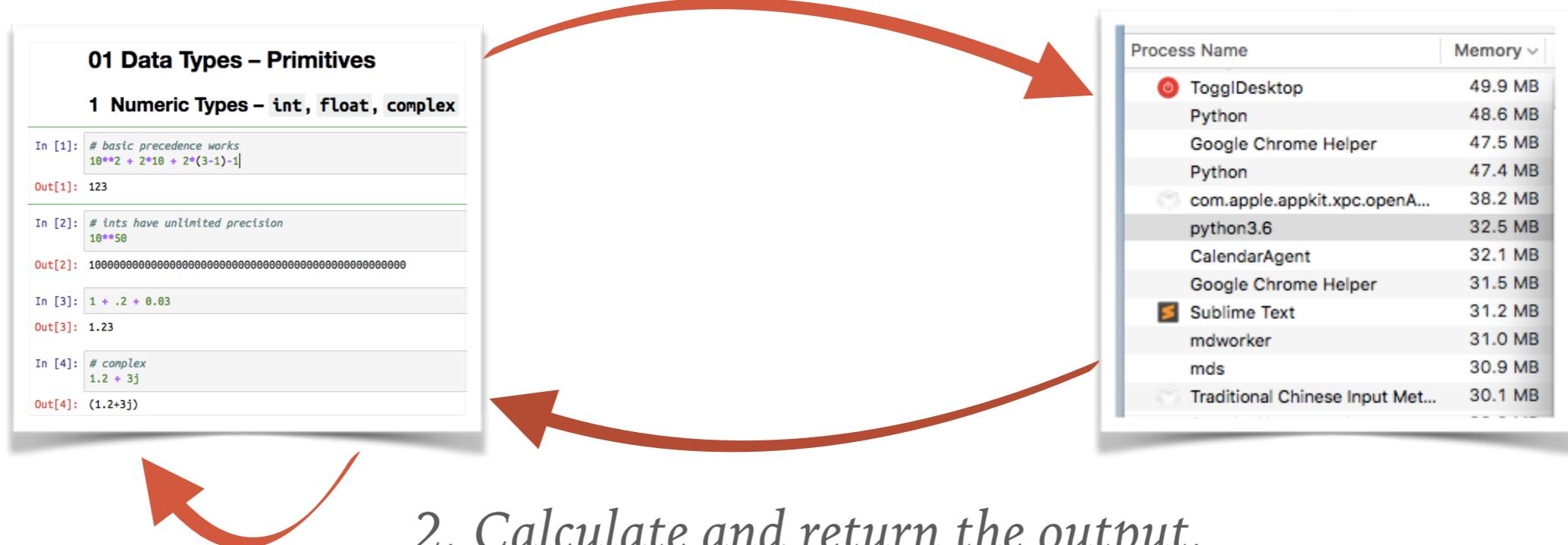
Esc	Edit mode → command mode.
Ctrl-Enter	Run the cell.
B	Insert cell below.
D, D	Delete the current cell.
M	To Markdown cell.
Cmd-/	Comment the code.
H	Show keyboard shortcuts.
P	Open the command palette.

Common Markdown Syntax

# Header 1	Header 1
## Header 2	Header 2.
> Block quote	Block quote.
* Item 1 * Item 2	Unordered list.
1. Item 1 2. Item 2	Ordered list.
emphasis	<i>Emphasis.</i>
strong emp	Strong emphasis.
	Markdown Cheatsheet markdown.tw

How Jupyter Notebook Works?

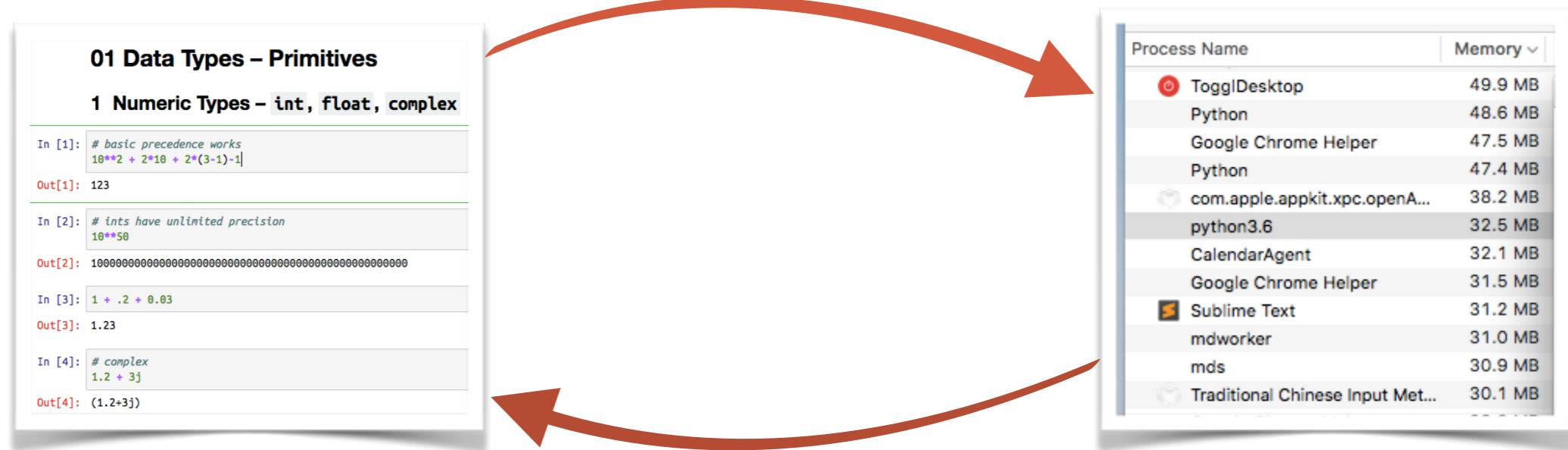
1. Connect to a Python kernel.



2. Calculate and return the output.

3. Store the output.

*A. When kernel halts or restart,
notebook keeps the outputs.*



*B. “Run All”
to execute all the code again*

The Numbers

01 Data Types – Primitives

1 Numeric Types – `int`, `float`, `complex`

```
In [1]: 16 * 10 + 2 * (3 - 1) - 1  
Out[1]: 123
```

```
In [2]: # ints have unlimited precision  
10**50
```

```
In [3]: 1 + .2 + 0.03
```

```
In [4]: # complex  
      1.2 + 3j
```

Out[4]: (1.2+3j)

- In [n]
 - n is the execution order, like the line number.
 - It may be:
 - 1, 2, 3, 4
 - 3, 5, 2, 4
 - Depends on how you run it.
 - “Restart & Run All”
to reorder the numbers.

Primitives

Data Types

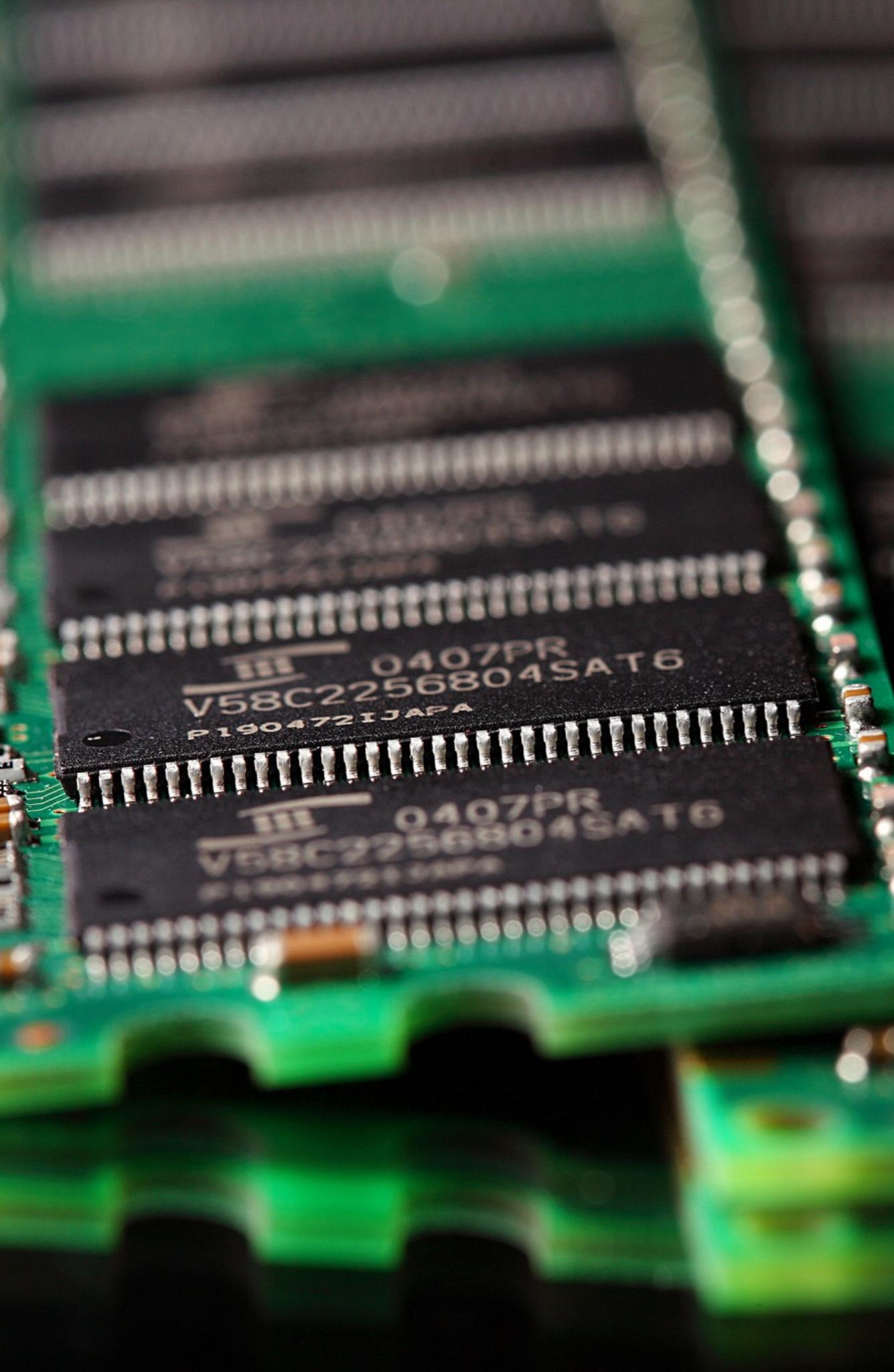


Programming?

- Programming is abstracting.
- Abstract the world with:
 - Data types: 1.
 - Operations: $1 + 1$.
 - Control flow: if ... else.
- They are from:
 - Libraries.
 - Your own code.

The Common Primitives

<code>int</code>	Integer: 3
<code>float</code>	Floating-point numbers: 3.14
<code>str</code>	Text Sequence: '3.14'
<code>bytes</code>	Binary Sequence: b'\xe4\xb8\xad\xe6\x96\x87'
<code>bool</code>	True or False
<code>None</code>	Represents the absence of a value.



Variables

- Points to an “object”.
- Everything in Python is an object, including class.
- The object:
 - Has a (data) type.
 - Supports an operation set.
 - Lives in the memory.

“

01_data_types_primitives.ipynb

-The Notebook Time

Delete the “Pointing”

- `del x`
- Now the *x* points to nothing.
- The object will be collected if no variable points to it.

Checkpoint: Calculate BMR

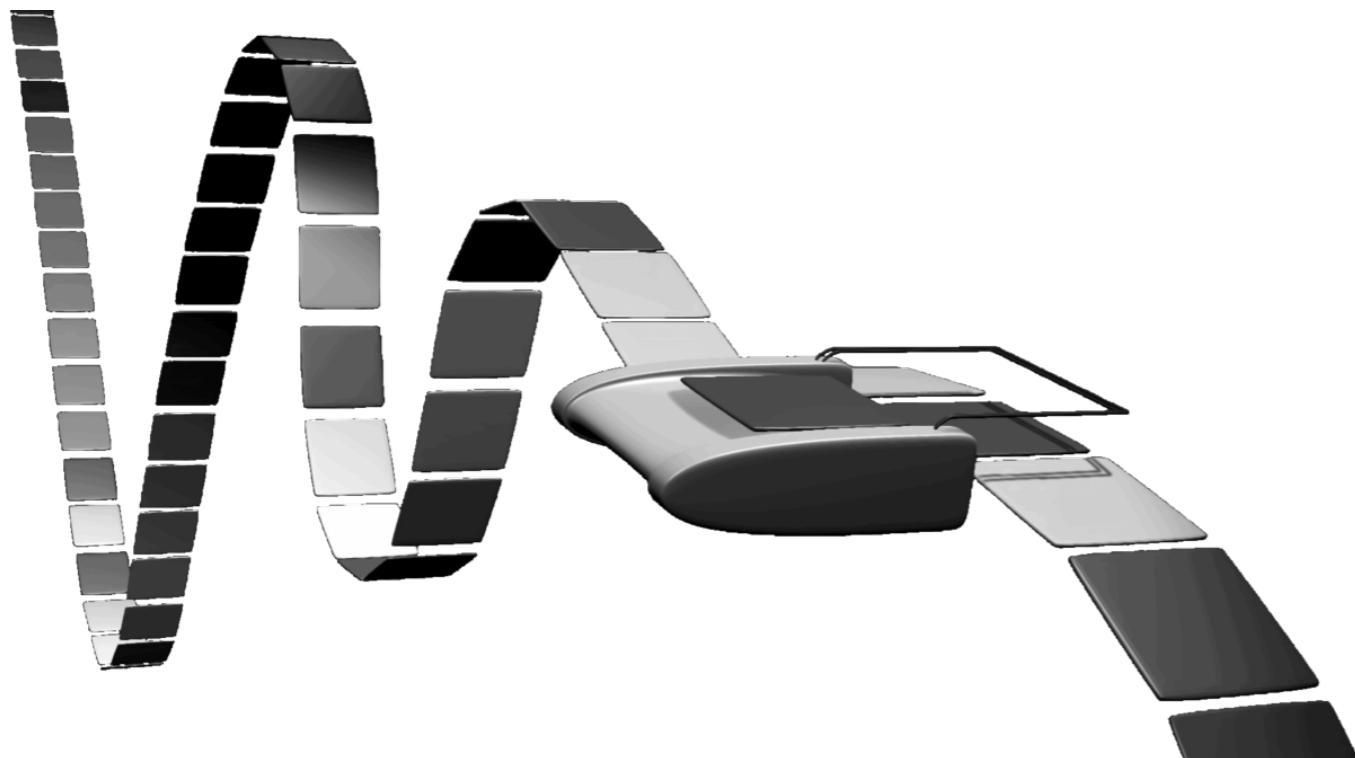
- The Mifflin St Jeor Equation:
 - $P = 10m + 6.25h - 5a + s$
 - Where:
 - P: kcal / day
 - m: weight in kg
 - h: height in cm
 - a: age in year
 - s: +5 for males, -161 for females
 - Just simply calculate it in notebook.

If & While

Control Flow

If & While

- if <condition>: ...
 - Run inner block *if* true.
- while <condition>: ...
 - The while-loop.
 - Run inner block *while* true.
 - I.e. run until false.



“

02_control_flow_if_while.ipynb

-The Notebook Time

Keep Learning

The Domains

- Web
 - [Django Girls Tutorial](#)
 - [The Taipei Version](#)
- Data / Data Visualization
 - [Seaborn Tutorial](#)
 - [The Python Graph Gallery](#)
 - [Matplotlib Gallery](#)
- Data / Machine Learning
 - [Scikit-learn Tutorials](#)
 - [Standford CS229 \(ML\)](#)
 - [Hsuan-Tien Lin](#)
- Data / Deep Learning
 - [TensorFlow Getting Started](#)
 - [Standford CS231n \(CNN\)](#)
 - [Standford CS224n \(RNN\)](#)
- Data / Statistics
 - [Seeing Theory](#)
 - [Statistics – SciPy Tutorial](#)
 - [StatsModels](#)
 - [Biological Statistics](#)
 - [Research Design](#)
- Ask or google
for your own domain!

The Language Itself

- All the “Dig More”.
- Understand the built-in batteries and their details:
 - [The Python Standard Library – Python Documentation](#)
- Understand *the* language:
 - [The Python Language Reference – Python Documentation](#)
 - [Data model](#)

Introducing Python

MODERN COMPUTING IN
SIMPLE PACKAGES



Bill Lubanovic

If You Need a Book

.....

- “Introducing Python – Modern Computing in Simple Packages”
- “精通Python：
運用簡單的套件進行現代運算”

The Learning Tips

- “Nothing is particularly hard if you divide it into small jobs.” – Henry Ford
- “The master has failed more times than the beginner has even tried.” – Stephen McCranie
- Also learn from the great people in the communities:
 - Taipei.py, PyHUG
 - Taichung.py, Tainan.py, Kaohsiung.py
 - PyCon TW
 - Python Taiwan

Composites

Data Types

The Common Composites

list	Contains objects.
tuple	A compound objects has an unique meaning, e.g., a point: (3, 1).
dict	Maps an object to another object.
set	Contains non-repeated objects.

“

03_data_types_composites.ipynb

-The Notebook Time

Recap

- list []
- tuple ()
- dict {::}
- set {}

For

Control Flow

For & Loop Control Statements

- for
 - The for-loop.
 - In each iteration, get the next item of a collection.
 - Supports str, list, tuple, set, and dict, etc.
 - I.e. iterate an iterable.
- break
 - Leave the loop.
- continue
 - Go the next iteration.
- loop ... else
 - If no break happens, execute the *else*.

Pass

- pass
 - Do nothing.
 - The compound statements must have one statement.
 - The if, while, for, etc.

“

04_control_flow_for.ipynb

-The Notebook Time

Checkpoint: Calculate Average BMR

height_cm	weight_kg	age	male
152	48	63	1
157	53	41	1
140	37	63	0
137	32	65	0

Try

Control Flow

“

05_control_flow_try.ipynb

-The Notebook Time

Raise Your Exception

- `raise RuntimeError('should not be here')`
 - Raise an customized exception.
 - Use *class* to customize exception class.
- `raise`
 - Re-raise the last exception.

The Guidelines of Using Try

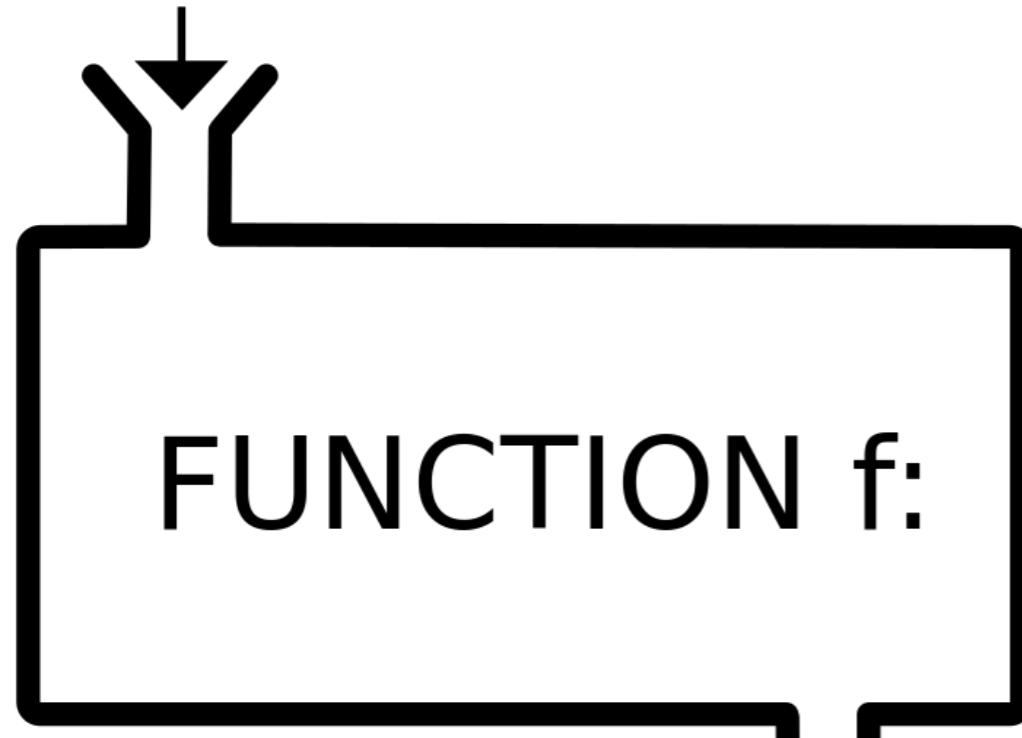
- An exception stops the whole program.
- However sometimes stop is better than a bad output.
- Only catch the exceptions you expect.
- But catch everything before raise to user.
- And transform the exception into log, email, etc.

Def

Control Flow

Functions

INPUT x



OUTPUT $f(x)$

- Reuse statements.
- def
- Take the inputs.
- return
- Give an output.
- If no return, returns *None*.
- Method
- Is just a function belonging to an object.

“

06_control_flow_def.ipynb

-The Notebook Time

Recap

- When calling function:
 - Keyword arguments: $f(a=3.14)$
 - Unpacking argument list: $f(*list_like, **dict_like)$
- When defining function:
 - Default values: $def f(a=None): \dots$
 - Arbitrary argument list: $def f(*args, **kwargs): \dots$
- Using docstrings to cooperate with others.

Docs – the Web Way

- The Python Standard Library
 - Tip: Search with a leading and a trailing space.
 - “ sys ”
- DevDocs
 - Tip: Set it as one of Chrome's *search engine*.
- seaborn API reference
 - For an example of 3rd libs.

The Guidelines of Designing Functions

- Use the simplest form first.
- Unless the calling code looks superfluous.

Checkpoint: Calculate Average BMR With Functions

height_cm	weight_kg	age	male
152	48	63	1
157	53	41	1
140	37	63	0
137	32	65	0

Common Functions

Libraries

“

07_libraries_
common_functions.ipynb

-The Notebook Time

Input & Output

Libraries



Input & Output

- IO: input & output.
- Standard IOs:
 - *stdout*: standard output.
 - *stderr*: standard error.
 - *stdin*: standard input.
- File IOs:
 - Including networking.
 - Command line arguments
 - Always validate user's inputs.

“

08_libraries_input_output.ipynb

-The Notebook Time

Checkpoint: Calculate Average BMR From the Dataset

- Read the *dataset_howell1.csv* in.
- Skip the first line which doesn't contain data.
- Transform the datatypes.
- Calculate the average BMR from the dataset.

Command Line Arguments

Libraries

“

09_libraries_
command_line_arguments.py

-The Notebook Time

Recap

- *open(...)* → file object for read or write.
- The *with* will close file after the suite.
- The Inputs:
 - *stdin* → *input()*
 - *for line in <file object>: ...*
 - *<file object>.read()*
- The outputs:
 - *print(...)* → *stdout*
 - *print(..., file=sys.stderr)* → *stderr*
 - *<file object>.write(...)*

Checkpoint: Calculate BMR From Command Line

- The requirement:
 - \$ python3 calc_bmr.py 152 48 63 M
 - 1120
- The hints:
 - Read the inputs from *sys.argv*.
 - Transform, calculate, and print it out!
- Get the extra bonus:
 - Organize code into functions by functionality.
 - Let user see nice error message when exception raises.
 - Refer to *09_libraries_command_line_arguments.py*.

Yield

Control Flow

“

10_control_flow_yield.ipynb

-The Notebook Time

“Yield” Creates a Generator

- If a function uses `yield`, it returns a generator.
- Save memory.
- Simplify code.

Comprehensions

Control Flow

“

11_control_flow_
comprehensions.ipynb

-The Notebook Time

Comprehensions & Generator Expression

- List Comprehension []
- Set Comprehension {}
- Dict Comprehension { : }
- Generator Expression ()

Functional Tricks

Libraries

The Functional Tricks

- The functional programming:
 - Is a programming paradigm.
 - Avoids changing-state and mutable data.
 - Sometimes makes code clean, sometimes doesn't.
 - Use it wisely.
- Python is *not* a functional language.
- But provides some useful tools.

“

12_libraries_functional_tricks.ipynb

-The Notebook Time

The Popular Functional Libs

- Recommend:
 - <https://github.com/Suor/fancy>
- Not so recommend, but still useful:
 - <https://github.com/pytoolz/toolz>
 - <https://github.com/EntilZha/PyFunctional>

Checkpoint: Calculate Average BMR With Comprehensions

- Either the table or the CSV is okay.
- Use at least one comprehension.

Import Antigravity

Libraries

The Useful Packages in Standard Library

- sys
- os and os.path
- glob
- math
- random
- decimal
- datetime
- collections
- re
- urllib
- smtplib
- json
- csv
- pickle
- gzip and many others
- threading
- multiprocessing
- asyncio
- pprint
- logging
- doctest
- sqlite3

The Useful Third-Party Packages

- Requests
- BeautifulSoup
- csvkit
- seaborn
- Pillow
- Flask
- Django
- pytest
- Sphinx
- pipenv
- The SciPy Stack
 - NumPy
 - SciPy
 - SymPy
 - Matplotlib
 - IPython
 - pandas
- python3wos.appspot.com

“

```
13_*libraries_
import_antigravity_*.ipynb
```

-The Notebook Time

Install Third-Party Package

- When using pip:
 - `pip install <package name>`
- When using Conda:
 - `conda install <package name>`

Checkpoint: Visualization

- Explore from *13_7_libraries_import_antigravity_seaborn.ipynb* .
- Refer to [Seaborn API Reference](#) to plot different graphs.
- Refer to [StatsModels Datasets](#) for different datasets.
 - Tip: “fair” in <http://www.statsmodels.org/stable/datasets/generated/fair.html> is the name of the dataset.
- Share your interesting insights with us!

Module & Package

Libraries

ma.py

mb.py

Import Module

- A Python file is just a Python module:

```
import ma  
import mb
```

- A module has a namespace:

```
ma.var  
mb.var
```

- The vars are different variables.

p/

__init__.py

ma.py

mb.py

Import Package

- A directory containing a `__init__.py` is just a package:

```
import p
```

- It executes the `__init__.py`.
 - Usually import the common function or module of the whole package.
- Import modules in a package:

```
import p.ma  
import p.mb
```

Make Them Shorter

- Import variables from a module:

```
from ma import x, y  
# or  
from ma import x  
from ma import y
```

- Import modules from a package:

```
from p import ma, mb  
# or  
from p import ma  
from p import mb
```

- Give an alias:

```
from ma import var as _var
```

- Mix them up:

```
from p.ma import var as _var
```

“

14_libraries_moudle_and_package

-The Notebook Time

Class

Data Types

The Object-Oriented Programming

- Class makes objects.
- Customize your:
 - Data type.
 - And its operations.
- A powerful tool to abstract the world.

The Object-Oriented Terms in Python

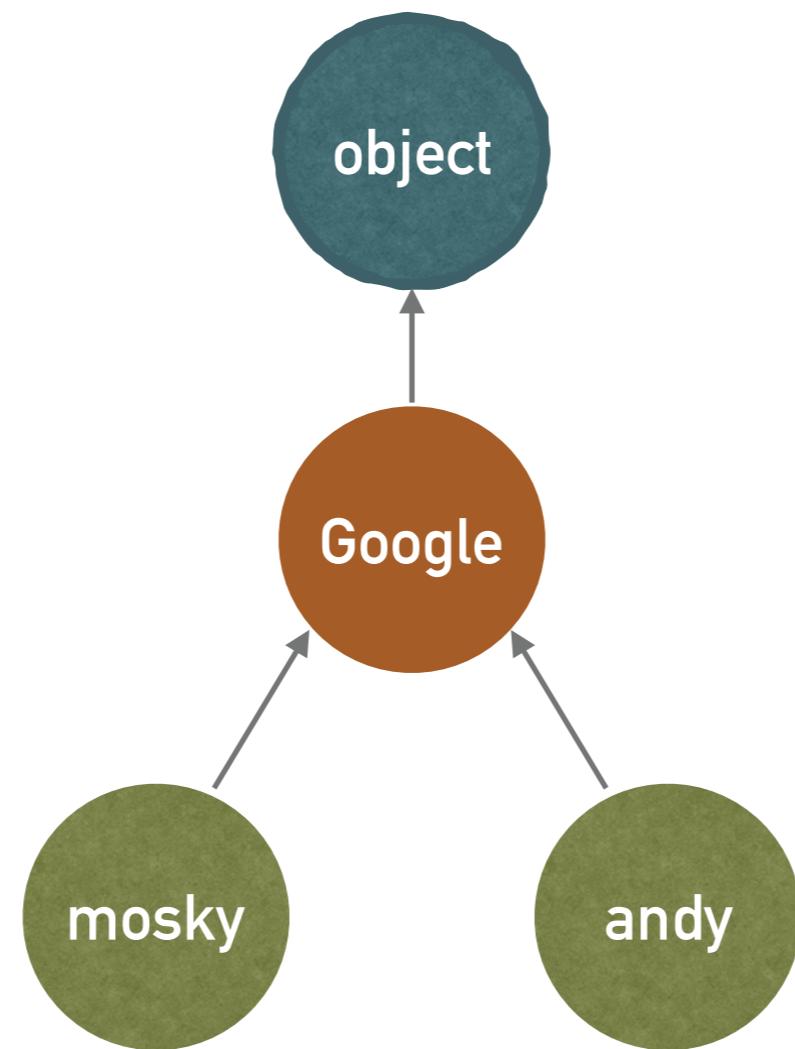
object	Everything in Python is an object, e.g., str, 'str'.
class	Makes instances, e.g., str.
instance	Made from class, e.g., 'str'.
attribute	Anything an object owns.
method	A function an object owns.

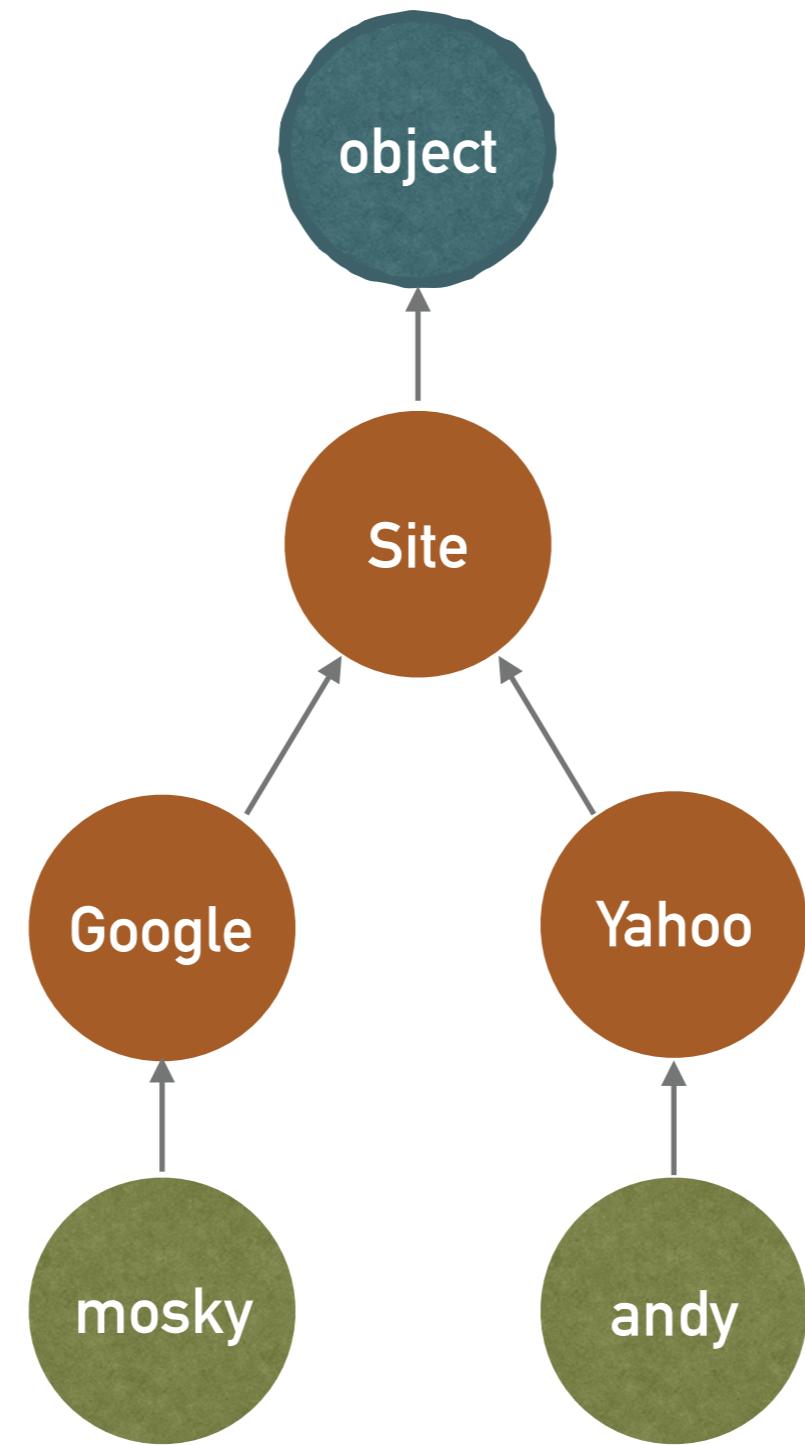
str	object class
str.__class__	object attribute class
str.split	object attribute function method
'str'	object instance
'str'.split	object attribute function method

“

15_data_types_class.ipynb

-The Notebook Time





Duck-Typing & Protocol

- Duck-Typing
 - “If it looks like a duck and quacks like a duck, it must be a duck.”
 - Avoids tests using `type()` or `isinstance()`.
 - Employs `hasattr()` tests or EAFP programming.
 - EAFP: easier to ask for forgiveness than permission.
- Iterator protocol, for example:
 - `__iter__()` returns itself.
 - `__next__()` returns the next element.
 - The for-loops use the iterator protocol to iterate an object.

The Guidelines of Designing Classes

- Don't use class, unless:
 - Many functions have the same arguments, e.g.,:
 - `def create_user(uid, ...): ...`
 - `def retrieve_user(uid, ...): ...`
 - `def update_user(uid, ...): ...`
 - When design or implementation.
- Don't use class method, etc., unless:
 - You're sure the method is only associate with class.

The Fun Facts

- `import this`
- Python's easter eggs and hidden jokes – Hacker Noon

Checkpoint: Classification

- Extend `13_8_libraries_import_antigravity_scikitlearn.ipynb` .
- Refer to [Scikit-Learn Random Forest](#) for the arguments.
- Share your awesome metrics with us!

At the End

- Python is useful in lot of domains. 
- Programming is just abstracting. 
 - The data types and control flows.
 - The ints, lists, if, while, functions, classes, etc.
 - The libraries.
- Practice is the key of learning programming.
 - Revisit the handouts and the checkpoints. 
 - Drive yourself to practice.
 - Resolve your own fun problems! 

Photo Credits

- “iRobot”
 - <http://fotonin.com/959257.html>
- “The Money”
 - https://commons.wikimedia.org/wiki/File:Forex_Money_for_Exchange_in_Currency_Bank.jpg
- “The Lamp”
 - <http://www.wikiart.org/en/pablo-picasso/still-life-with-lamp-1944>
- “The Memory”
 - <https://www.flickr.com/photos/jonathancohen/14326234016/in/photolist-nPXGfL-6A5mka-8g8V8p-6sh3q9-7TQBPi-6XEKAgiC4WX-7TTQBj-7Rmq5j-7TQjkk-7TQkLk-7TQm1R-dj4aCh-nNUEda-tkHM-tkHT-NEozn-77y2Yj-a58kY-5zg7q-4zCggd-foNHkM-no917U-avhrG1-tkHN-8MFpt1-n9ZpnN-gkhTDG-8yQtot-7TTzB7-7TTKZG-5UEAVY-4w9tXh-5b495D-bxZYcGiJYKMV-Vv7m-Uq9B-8JoDpy-dtmAgu-qLrDH-77u5Wi-r9c9w3-6BdtZs-6E1U8-rNBx7U-aWJ5SF-5WaWZp-5cnXjL-cZBJ3w>
- “The Turing Machine”
 - <https://zh.wikipedia.org/wiki/%E5%9B%BE%E7%81%B5%E6%9C%BA#/media/File:Maquina.png>
- “The Function”
 - https://commons.wikimedia.org/wiki/File:Function_machine2.svg
- “The Disk”
 - <https://www.flickr.com/photos/scaar/8472199817/in/photolist-dUEeQD-5HdAN4-5xWVKb-dUKPZS-dUEe2r-5snGPf-59skU8-dUEf6z-4oNs1w-6p5jdX-atp6TY-aC3AqV-a4BfGL-aAAAfg-9NWnEM-aG17Z2-9oAykX-61benn-8Cmy4D-7HN3aV-49jHVw-LrZVp-asPyUp-sezv1-a2iAsW-4GauZa-zTjVX-ejs1aH-abdSy1-93wj7g-4wJnUd-6vjZ7N-9csbbv-gKA3J-6tuRY1-58CCzq-ze5cZ-48L5Kt-3V4d2-7BKdfS-34RnyZ-4tHg3C-xDZHYA-pZL3UJ-QCNmm-rj3zys-Q5FL-8k69gA-bWDz84-dT1K7g>