Practicing Python 3

Mosky

Python?

Websites

- Pinkoi
- Google search engine
- Instagram
- Uber
- Pinterest
- Use as their back-end language.

Desktop Applications

- Dropbox
- BitTorrent
- Blender (3D graphics)
- Disney

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")
```

Science

- NASA
  - Gravitational waves, etc.
- CERN LHC
- MMTK (Biology)

Embedded System

- iRobot uses Python.
- Raspberry Pi supports Python.
- 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

```python
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

x = symbols('x')

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

```
from sympy import init_printing
init_printing()

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

x^2

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

$2x$

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_ran
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 Python packages:
- ZIPCodeTW,
  MoSQL, Clime, etc.
- http://mosky.tw/

Our Toolbox

We Will Use

(a terminal)	Master the machine.
Python 3	Not Python 2.
Jupyter Notebook	Learn Python with browser.
Visual Studio Code	A full-featured source code editor.
(other libs)	Will be introduced in this slides.

On Mac

- Install Homebrew:
  - https://brew.sh/
- Open a terminal:

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

ace) / “terminal”

- l:
  - $ brew install python3
  - $ pip3 install jupyter numpy scipy sympy matplotlib ipython pandas flask seaborn statsmodels scikit-learn
  - $ jupyter contrib nbextension install --user

On Windows

- Install Python 3 with Miniconda
  - http://conda.pydata.org/miniconda.html
  - Python 3.6 / Windows / 64-bit (exe installer)
- Open a terminal:
- / Type “cmd” or “Command Prompt”

Hint to talk to Windows. Enter the command without >.

  - `> conda install jupyter numpy scipy sympy matplotlib ipython pandas flask seaborn statsmodels scikit-learn`

Common MS-DOS Commands

`C:`	Go to *C:* drive.
`cd PATH`	Change Directory to *PATH*. *PATH* can be */Users/python/*, *python/*, 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

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

- `$ cd Downloads/Prac<Tab>`
- Or your project folder.
- `$ jupyter notebook`
- It will open up a browser.
- Keep the terminal open.

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 – 1

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

Checkpoint: Say Hi to Python – 2

- 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

`Ctrl-Enter`	Run the cell.
`B`	Insert cell below.
`D, D`	Delete the current cell.
`M`	To Markdown cell.
`Esc`	Edit mode → command mode.
`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*`	*Emphasis.*
`**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 remembers the outputs.

But when connect to a new kernel,

kernel remember ***nothing****.*

01 Data Types – Primitives

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

```
In [1]: # basic precedence works
        10**2 + 2*10 + 2*(3-1)-1
Out[1]: 123

In [2]: # ints have unlimited precision
        10**50
Out[2]: 100000000000000000000000000000000000000000000000000

In [3]: 1 + .2 + 0.03
Out[3]: 1.23

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

Process Name	Memory
TogglDesktop	49.9 MB
Python	48.6 MB
Google Chrome Helper	47.5 MB
Python	47.4 MB
com.apple.appkit.xpc.openA...	38.2 MB
python3.6	32.5 MB
CalendarAgent	32.1 MB
Google Chrome Helper	31.5 MB
Sublime Text	31.2 MB
mdworker	31.0 MB
mds	30.9 MB
Traditional Chinese Input Met...	30.1 MB

B. "Run All"

to refresh the new kernel.

01 Data Types – Primitives

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

```
In [1]:  # ...edence works
         10...2*10 + 2*(3-1)-1
Out[1]:  123

In [2]:  # ints have unlimited precision
         10**50
Out[2]:  100000000000000000000000000000000000000000000000

In [3]:  1 + .2 + 0.03
Out[3]:  1.23

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

The Numbers

- 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 Outline

- Data Types – Primitives
- Control Flow – If & While
- Data Types – Composites
- Control Flow – For
- Control Flow – Try
- Control Flow – Def
- Libraries – Common Functions
- Libraries – Input & Output
- Libraries – Command Line Arguments
- Control Flow – Yield
- Control Flow – Comprehensions
- Libraries – Functional Tricks
- Libraries – Import Antigravity*
- Libraries – Module & Package
- Data Types – Class
- And checkpoints!

Primitives

int	Integer: 3
float	Floating-point numbers: `3.14`
str	Text Sequence: `'3.14'`
bytes	Binary Sequence: `b'\xe4\xb8\xad\xe6\x96\x87'`
bool	`True` or `False`
None	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
  - – Basal metabolic rate - Wikipedia
- 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

Another From of If

- `s = 5 if is_male else -161`
  - Like *c ? t : f* in other languages.

Composites

Data Types

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

For

Control Flow

For & Related 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.
  - For every compound statements.

“

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

- `average(n_1, ...) = sum(n_1, ...)  / count(n_1, ...)`
  - Do you have a single *n* (BMR)?
  - Can you sum all of the *n*?
  - Do you have the count of *n*?
- My favorite problem solving methods:
  - Divide and conquer. Let's break it down!
  - Try and learn from the error!
  - Make it work, make it right, and finally make it fast.

Try

Control Flow

“

05_control_flow_try.ipynb

-The Notebook Time

Raise Your Exception

- `raise`
  - `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.

Function

Control Flow

Function

- Reuse statements.
- Extend built-in operations.
- `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_function.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): ...*
  - Arbitrary argument list: *def f(*args, **kwargs): ...*
- Using duck typing to max the flexibility.
- Using docstrings to cooperate.
- Small functions – *lambda*.

Common Functions

Libraries

“

07_libraries_
common_functions.ipynb

-The Notebook Time

Input & Output

Libraries

Input & Output

- IO: input & output.
- Standard IO:
  - Output (stdout).
  - Error (stderr).
  - Input (stdin).
- File IO:
  - 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.
- Are the data correct and in the type (*float*) we need?
- Calculate the average BMR from the data.
- Remember the problem solving methods!

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.
- Input
  - stdin → *input()*
  - *for line in <file object>: …*
  - *<file object>.read()*
- Output
  - *print(…)* → stdout
  - *print(…, file=sys.stderr)* → stderr
  - *<file object>.write(…)*

Checkpoint: Calculate BMR From Command Line

- `$ python3 calc_bmr.py 152 48 63 M`
- `1120`
- Refer to *09_libraries_command_line_arguments.py*.
- Remember the problem solving methods!

Yield

Control Flow

“

10_control_flow_yield.ipynb

-The Notebook Time

Yield Create 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

Checkpoint: Calculate Average BMR With Comprehensions

- Either the table or the CSV is okay.
- Use at least one comprehension.
- Still, remember the problem solving methods!

Import Antigravity

Libraries

The Useful Packages in Standard Library

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

The Useful Third-Party Packages

- Requests
- Beautiful Soup
- csvkit
- Pillow
- Flask
- Django
- pytest
- Sphinx
- seaborn
- pipenv
- The SciPy Stack
  - Numpy
  - SciPy
  - Sympy
  - Matplotlib
  - IPython
  - pandas
- python3wos.appspot.com
- …

“

13_*_libraries_
import_antigravity_*.ipynb

-The Notebook Time

The Popular Functional Libs

- https://github.com/EntilZha/PyFunctional
- https://github.com/pytoolz/toolz
- https://github.com/Suor/funcy

Install Third-Party Package

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

Checkpoint: Visualization

- TBC

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 `var`s 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 It 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
```

- Import relatively, if in `ma.py`:

```
from . import mb
```

“

14_libraries_moudle_and_package

-The Notebook Time

Class

Data Types

Object-Oriented Programming

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

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
`str.split`	object | attribute | method | function
`'str'`	object | instance
`'str'.split`	object | attribute | method | function

“

15_data_types_class.ipynb

-The Notebook Time

object
Google
mosky
andy

object
Site
Google
Yahoo
mosky
andy

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.
- Iterator Protocol
  - `__iter__()` returns itself.
  - `__next__()` returns the next element.
  - for-loop uses the iterator protocol to iterate an object.

The Guidelines of Using Class

- 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 classmethod, etc.
  - Unless you're sure the method is only associate with class.
- Let's keep code simple!

The Fun Facts

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

Keep Learning

The Domains

- Web
  - Django Girls Tutorial [The Taipei Version]
- Data / Machine Learning
  - scikit-learn Tutorials
- Data / Data Visualization
  - seaborn Tutorial

The Language Itself

- All the “Dig More”.
- Understand built-in power and library details:
  - The Python Standard Library – Python Documentation
- Understand *the* language:
  - The Python Language Reference – Python Documentation
    - Data model

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 General Advice

- Problem → project → learnt things!
- Make it work, make it right, and finally make it fast!
- Learn with the communities:
  - Taipei.py
  - PyHUG
  - PyCon TW
  - Python Taiwan

Checkpoint: Classification

- TBC