



Practicing Python 3

Mosky



Python?

[Shop](#)[Wall](#)[Window](#)

423017 unique items

Search for great designs

Trending Now Check out the most recommended



Websites

- Pinkoi
- Google search engine
- Uber
- Pinterest
- use as their back-end language

Shoes & Bags

Accessories

Home & Li

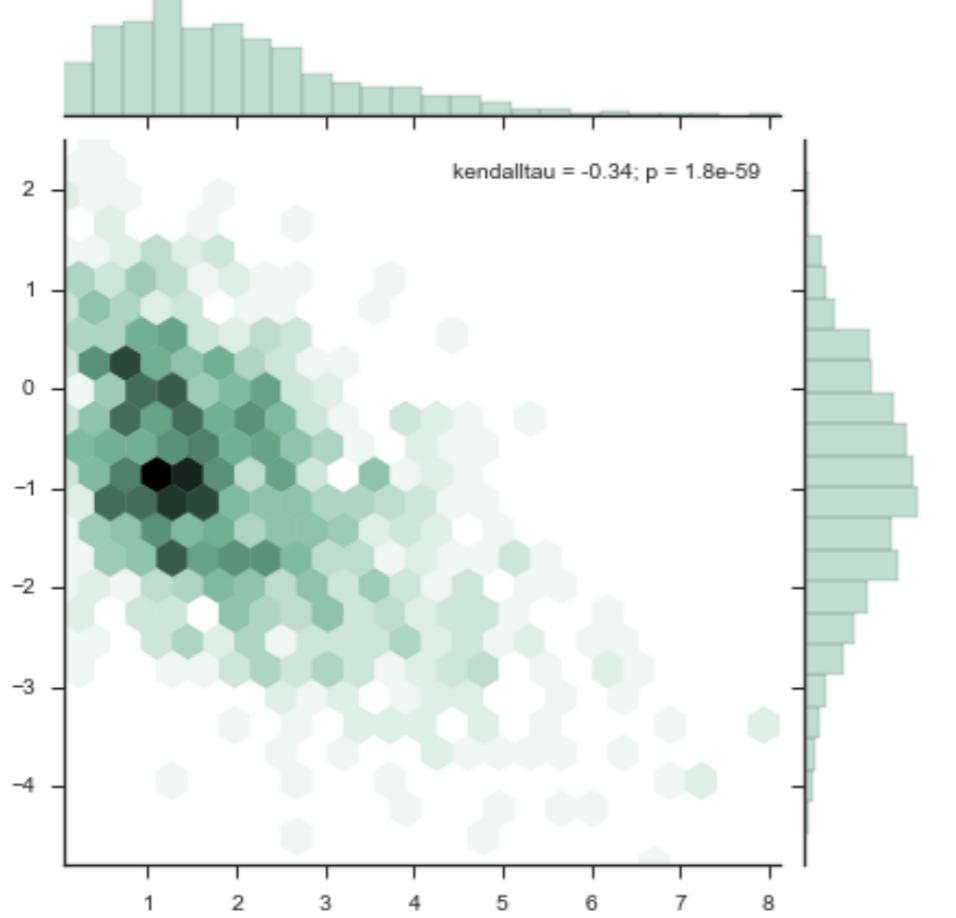


Dropbox

Desktop Applications

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

Science



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")
```

- NASA
 - gravitational waves
 - etc.
- CERN LHC
- MMTK (Biology)
- use Python to do
 - Data mining
 - Data visualization
 - Scientific programming
 - Simulation



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
- easy to speed up
 - Numba
 - Cython
- has 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 init_printing  
init_printing()  
  
from sympy import symbols  
x = symbols('x')  
x**2
```

x^2

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

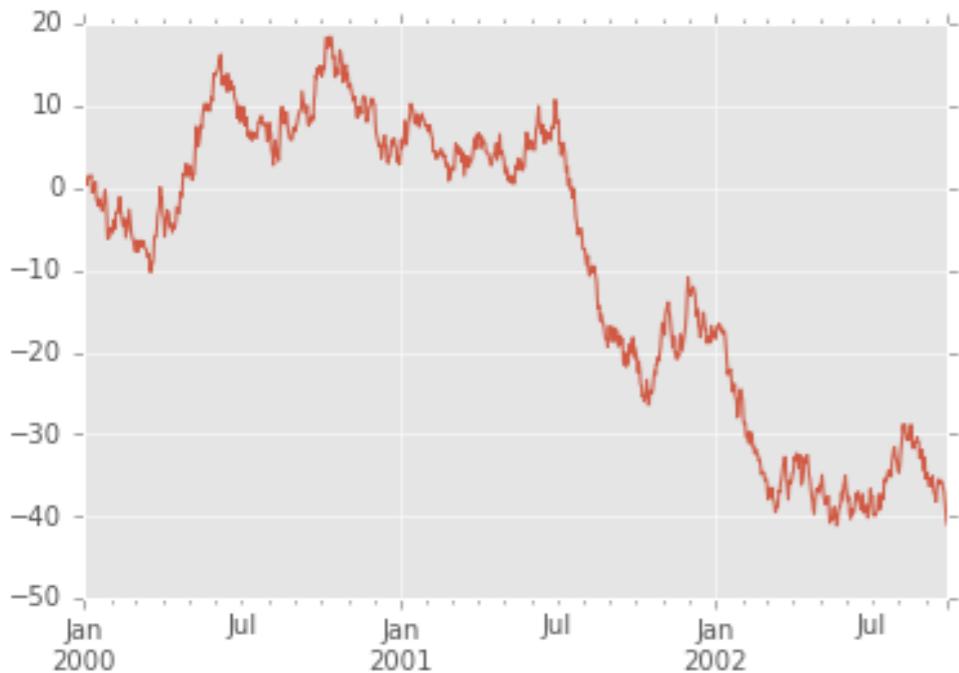
$2x$

```
from sympy import symbols  
from sympy import diff  
x = symbols('x')
```

x^2
 $\text{diff}(x^2)$

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/'  
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/>

Outline

- Environment
 - Introduction & Setup
 - “Hello, Python!”
- General Programming Topics
 - Data Type and Control Flow
 - Function
 - Built-in Functions
 - IO and File
 - Methods of String
- The Challenges: Find the big file!

Outline (cont.)

- Fascinating Programming Topics in Python
 - Comprehensions
 - Functional Tools
 - Module and Package
 - Useful Packages
 - Object-oriented Programming in Python
- A Mini Project: Crawl the Internet!

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

Open a Terminal

- Windows:
 - Start Menu / Search / Type “cmd” or “Command Prompt”
 - It has super limited features. If you wanna get a better one:
 - ConEmu
 - Babun
 - Cygwin
- Mac:
 - Spotlight (Cmd-Space) / “terminal”
 - It also has only limited features. Get a better one:
 - iTerm2

Common MS-DOS Commands

C:	Go to 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	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

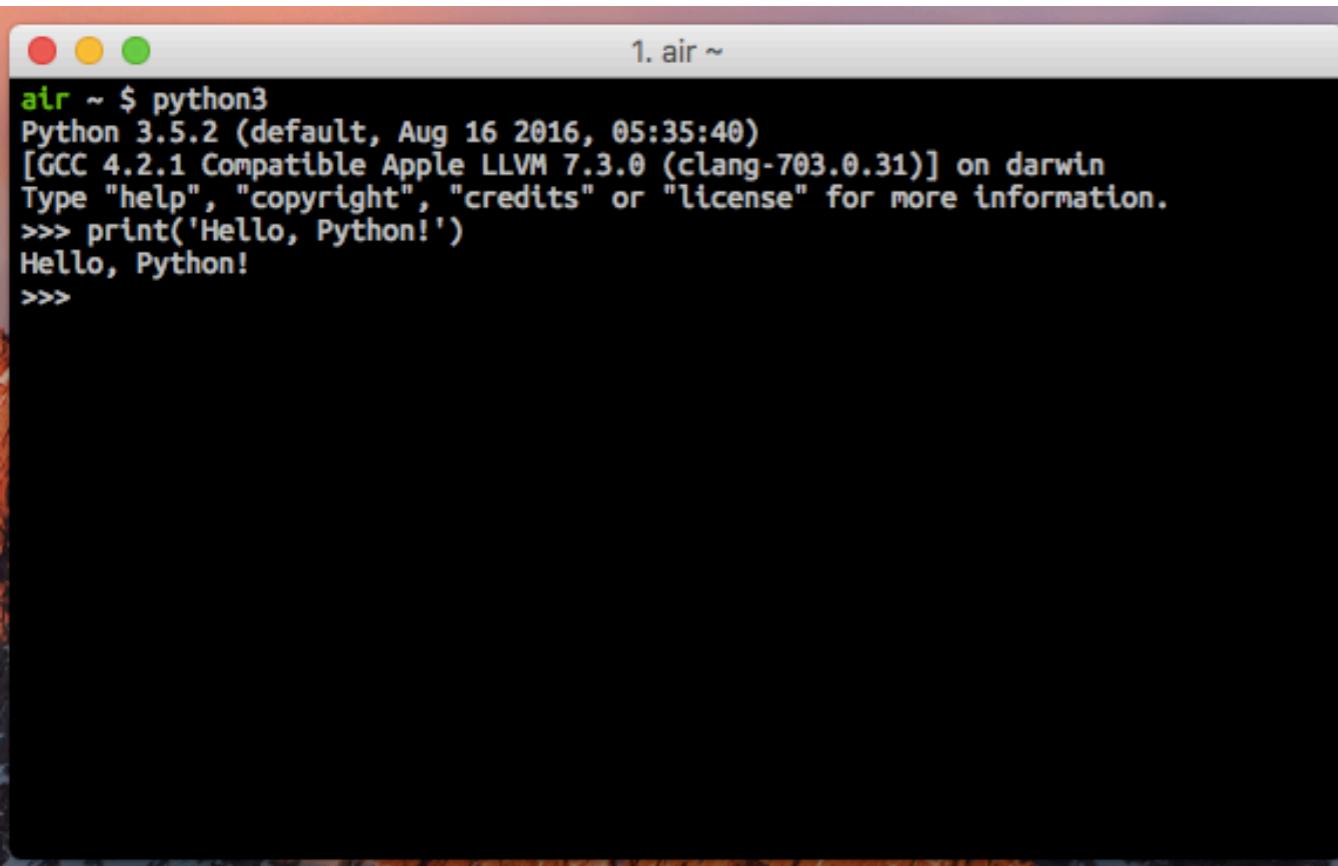
Exit the current terminal.

Install Python 3 with Miniconda

- Install Python 3 with Miniconda
 - <http://conda.pydata.org/miniconda.html>
- On Mac and Linux,
 - “bash installer”
 - `chmod +x Miniconda3-latest-MacOSX-x86_64.sh`
 - `./Miniconda3-latest-MacOSX-x86_64.sh`

- If you are a pro,
 - Mac:
 - `$ brew install python3`
 - Ubuntu / Debian (Linux):
 - `$ sudo apt-get install python3`
 - and get pip:
 - <https://pip.pypa.io/en/stable/installing/#installing-with-get-pip-py>

OS Shell, and Python Shell



```
air ~ $ python3
Python 3.5.2 (default, Aug 16 2016, 05:35:40)
[GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, Python!')
Hello, Python!
>>>
```

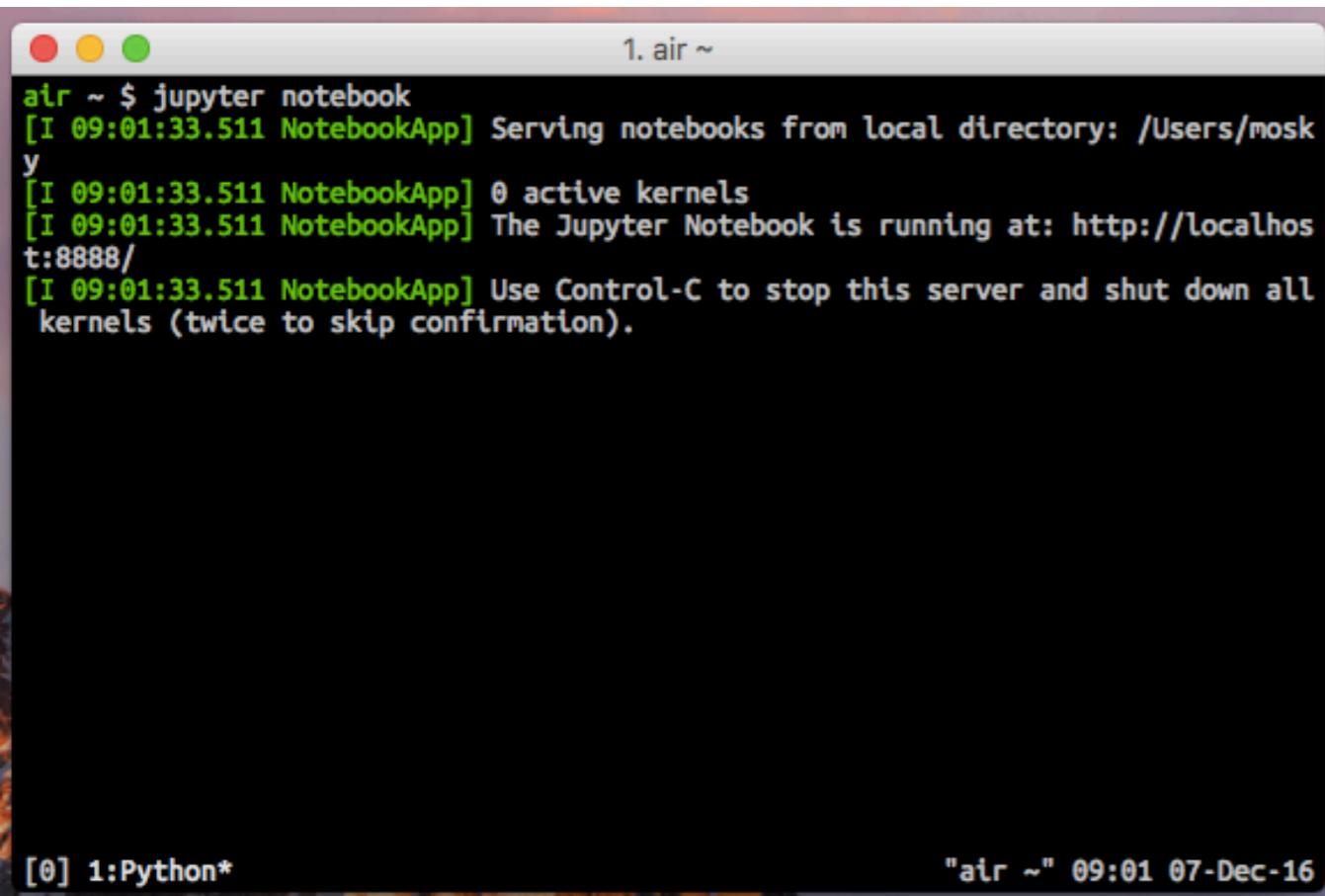
- Give command to ____ in a shell.
- On Mac and Linux,
default OS shell is Bash
 - \$
- On Windows, default OS shell is
Command Prompt (cmd)
 - >
- Usually we use \$ to indicate
command on an OS shell.
- Python's
 - python or python3
 - >>>
 - exit()

Install Jupyter Notebook and Other Packages

- \$ conda install numpy scipy matplotlib ipython pandas sympy jupyter requests beautifulsoup4 flask
- Includes
 - SciPy stack
 - Jupyter Notebook
 - Libs for a minimal web crawler: requests & beautifulsoup4
 - A lightweight web framework: flask
- If you are a pro,
 - Use pip instead.
 - On Mac, *do not* add sudo .

Start Jupyter Notebook

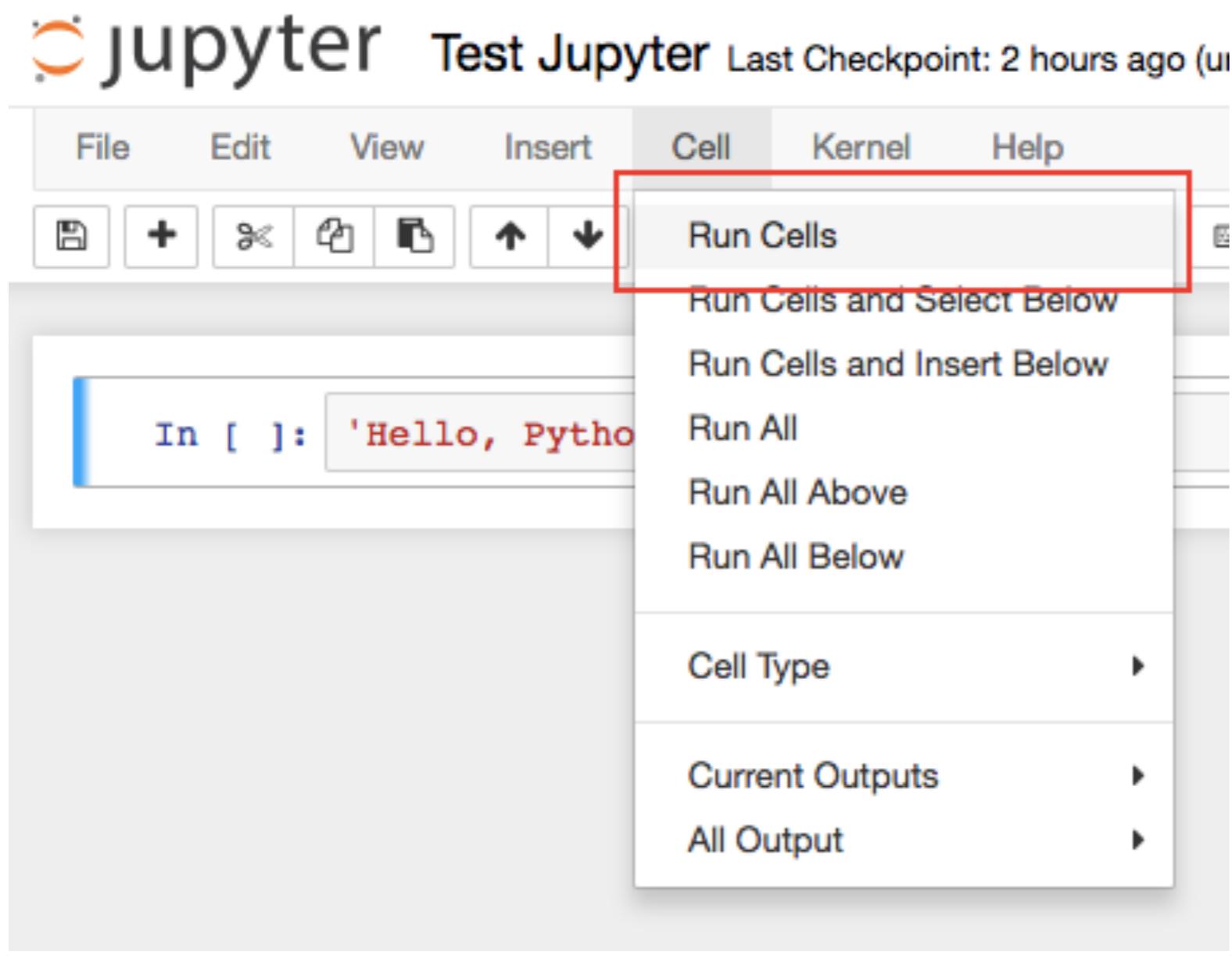
- \$ jupyter notebook
- It will open up a browser.
- Keep the terminal open.



A screenshot of a Mac OS X terminal window titled "1. air ~". The window contains the following text output from running the command \$ jupyter notebook:

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

The terminal window has a red, yellow, and green close button in the top-left corner. The status bar at the bottom shows "[0] 1:Python*" on the left and "air ~ 09:01 07-Dec-16" on the right.

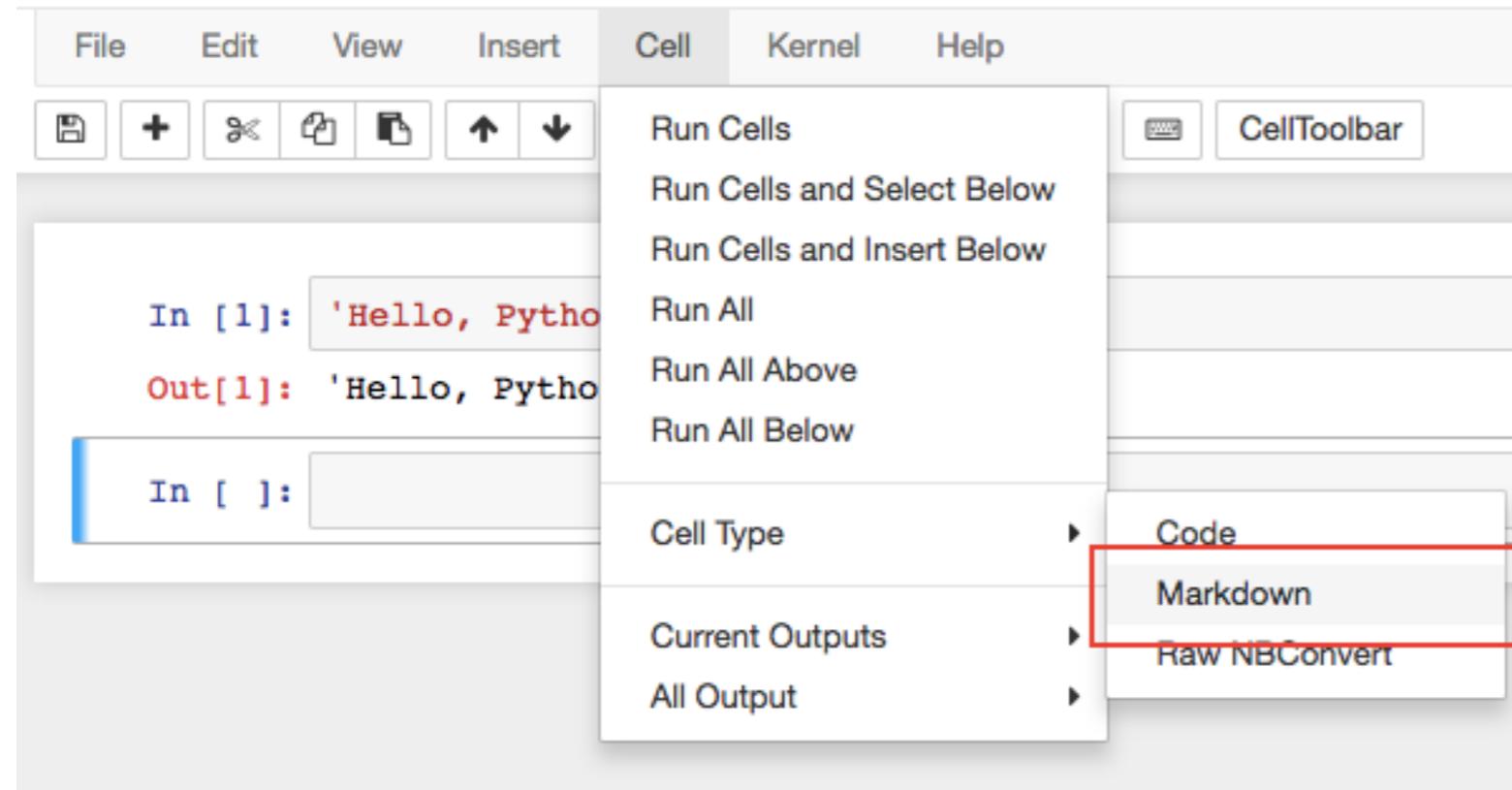


```
In [1]: 'Hello, Python!'
Out[1]: 'Hello, Python!'

In [1]: 'Hello, Python!'
Out[1]: 'Hello, Python!'

In [ ]:
```

Jupyter Notebook – Run Cell, “Ctrl-Enter”, and “B”



'Hello, Python!'

'Hello, Python!'

Common Shortcuts

1. `Esc` to escape from cell (from edit mode to command mode).
2. `Ctrl-Enter` to run.
3. `B` to insert cell below.
4. `D, D` to delete the current cell.
5. `Shift-Cmd-P` to open command palette.
6. `H` to open keyboard shortcuts.

'Hello, Python!'

'Hello, Python!'

Common Shortcuts

1. Esc to escape from cell (from edit mode to command mode).
2. Ctrl-Enter to run.
3. B to insert cell below.
4. D, D to delete the current cell.
5. Shift-Cmd-P to open command palette.
6. H to open keyboard shortcuts.

Jupyter Notebook – Markdown and Shortcuts

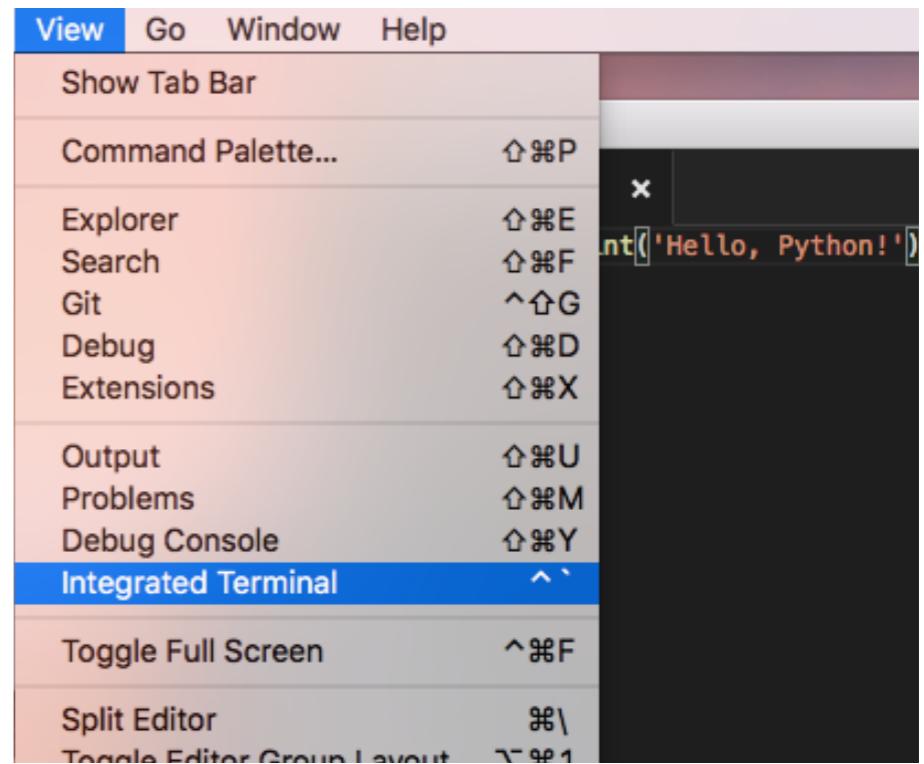
Install Visual Studio Code

- <https://code.visualstudio.com/download>
- Just download and install it!
- Execute a program:
 - \$ cd DIR_PATH_OF_PYTHON_FIELS
 - \$ python hello.py
 - or
 - \$ python3 hello.py

A screenshot of the Visual Studio Code interface. On the left is a dark sidebar with icons for file operations, search, and other tools. A file named 'hello.py' is open in the main editor area. The code inside the file is:

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

The status bar at the bottom shows the word 'PYTHON'.



A screenshot of the Integrated Terminal window in Visual Studio Code. The terminal displays the following text:

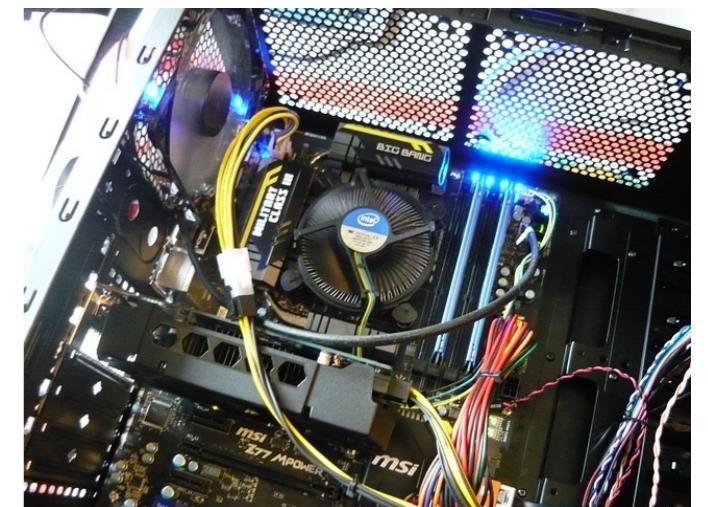
```
TERMINAL
air ~ $ cd Desktop/
air ~/Desktop $ python3 hello.py
Hello, Python!
air ~/Desktop $
```

Visual Studio Code – Edit and Run in Integrated Terminal

When to Use What

- Jupyter Notebook
 - Interactive programming
 - Shorter (\leq 10 lines per cell)
 - Learning
 - Exploring data
- Visual Studio Code (i.e. an editor with a terminal)
 - “Real” programming
 - Longer
 - Automation scripts
 - Server-side code

Hello, Python!



A circular graphic with a dark, textured, cracked stone or metal appearance, set against a blue background with radial light effects. Inside the circle, there is a block of C programming language code. The code is related to character input processing, including functions for reading characters and managing state variables like state, nc, nl, nw, and state.

v1 001_hello.py

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

Try It

- Type them on
 - Jupyter Notebook, ignore
 - if __name__ == '__main__':
 - def ...
 - Visual Studio Code
- and execute it.

```
01_hello.py
def say_hello(name):
    print('Hello, {}'.format(name))

if __name__ == '__main__':
    say_hello('Mosky')

● ● ●
3. bash
Last login: Sun Nov  1 23:17:44 on ttys001
air ~ $ cd Projects/Practicing_Python_3/examples/
air ~/Projects/Practicing_Python_3/examples $ py3 01_hello.py
Hello, Mosky!
air ~/Projects/Practicing_Python_3/examples $
```

In [1]:

```
def say_hello(name):
    print('Hello, {}'.format(name))
```

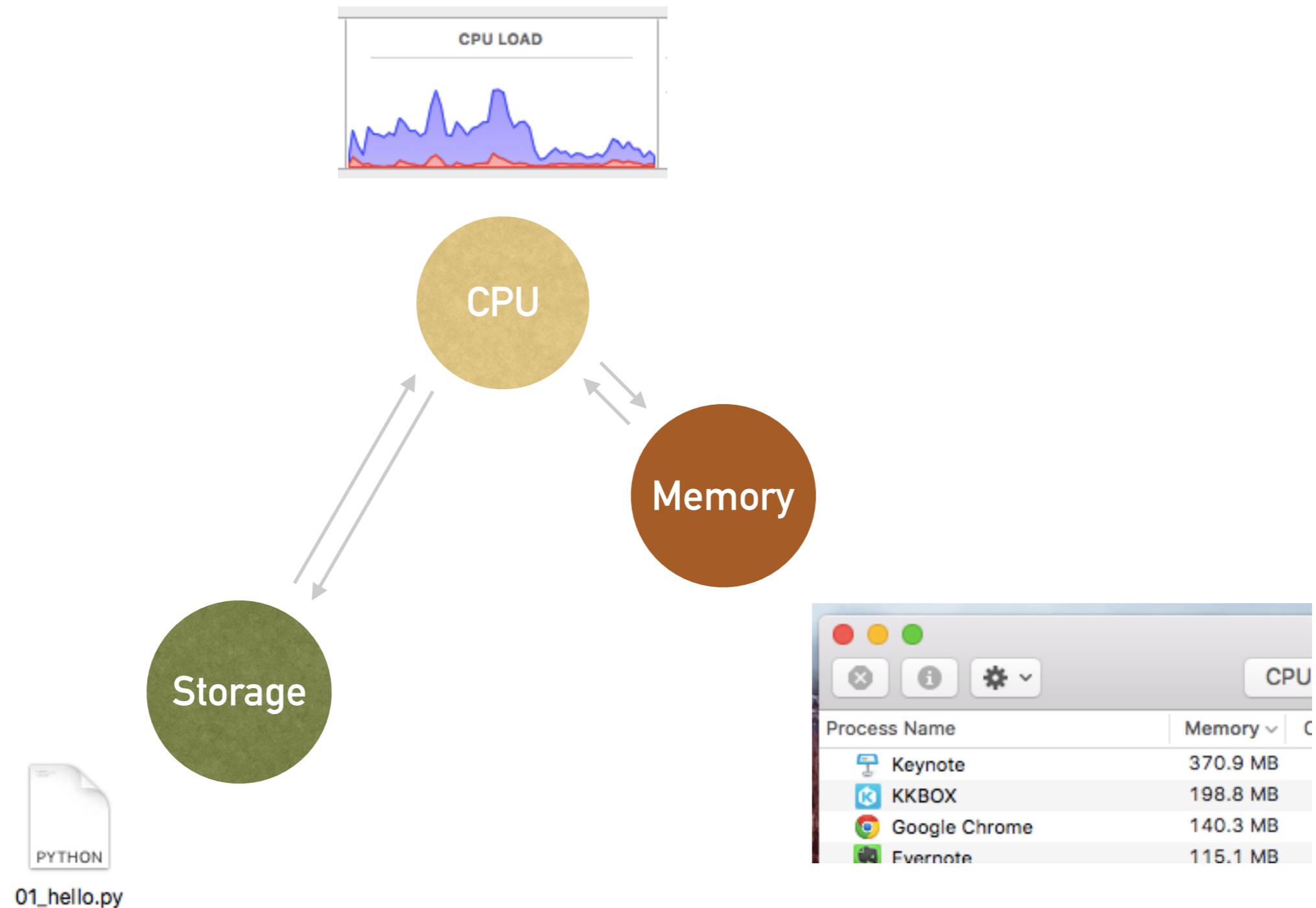
In [2]:

```
say_hello('Mosky')
```

Hello, Mosky!

2. Python

```
air ~/Projects/Practicing_Python_3/examples $ ipython3 notebook
[I 23:28:45.901 NotebookApp] Serving notebooks from local directory: /User
y/Projects/Practicing_Python_3/examples
[I 23:28:45.901 NotebookApp] 0 active kernels
[I 23:28:45.901 NotebookApp] The IPython Notebook is running at: http://lo
c:8888/
[I 23:28:45.901 NotebookApp] Use Control-C to stop this server and shut do
wn kernels (twice to skip confirmation).
```



v1 002_formal_hello.py

```
def say_hello(name):
    print('Hello, {}!'.format(name))

if __name__ == '__main__':
    say_hello('Mosky')
```

Python Tutor

► <http://www.pythontutor.com/visualize.html>

[Start shared session](#) [Online Python Tutor: Visualize Python, Java, JavaScript, TypeScript, and Ruby code execution](#)
[What are shared sessions?](#)

Python 3.3

```
1 def say_hello(name):
2     print('Hello, {}!'.format(name))
3
4 if __name__ == '__main__':
5     say_hello('Mosky')
```

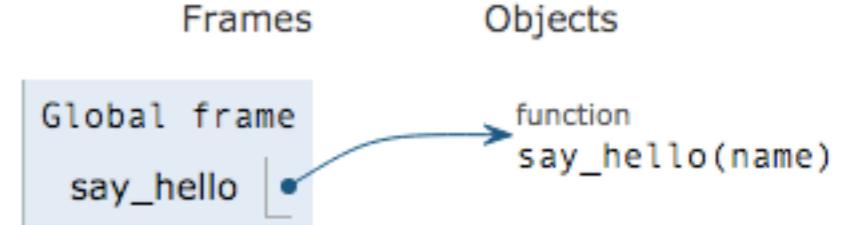
[Edit code](#)

Step 2 of 6 Forward > [Last >>](#)

Green arrow: line that has just executed
Red arrow: next line to execute

Frames Objects

Global frame	function say_hello
--------------	-----------------------



Syntax

- Each statement ends with a newline.
- A compound statement ends with a colon (:).
- Indentation () identifies a suite, not curly brace ({}).
- An indentation is 4-space, not tab.
- Spaces and blank lines are ignored usually.

v1 003_formal_hello_with_comments.py

```
# the part after `#` is "comment".
# here we define a `say_hello` "function" which accepts an "argument" `name`.
def say_hello(name):

    # the indents distinguish the lines in or out of the function.
    # in this function, we call a bulit-in function `print` with an argument.

    # `'Hello, {}!'` is a string.
    # ` `.format` is its "method", method is a function belonging an "object".
    # we give `name` as an argument of the format.

    print('Hello, {}!'.format(name))

# `if` is used for conditional execution.
# `__name__` is a bulit-in "variable", it will be the name of a "module".
# if it is executed directly, it will be ` '__main__'`.
if __name__ == '__main__':

    # call the function we just defined
    say_hello('Mosky')
```

Checkpoint: Say Hi to Python

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

Programming?



Abstraction

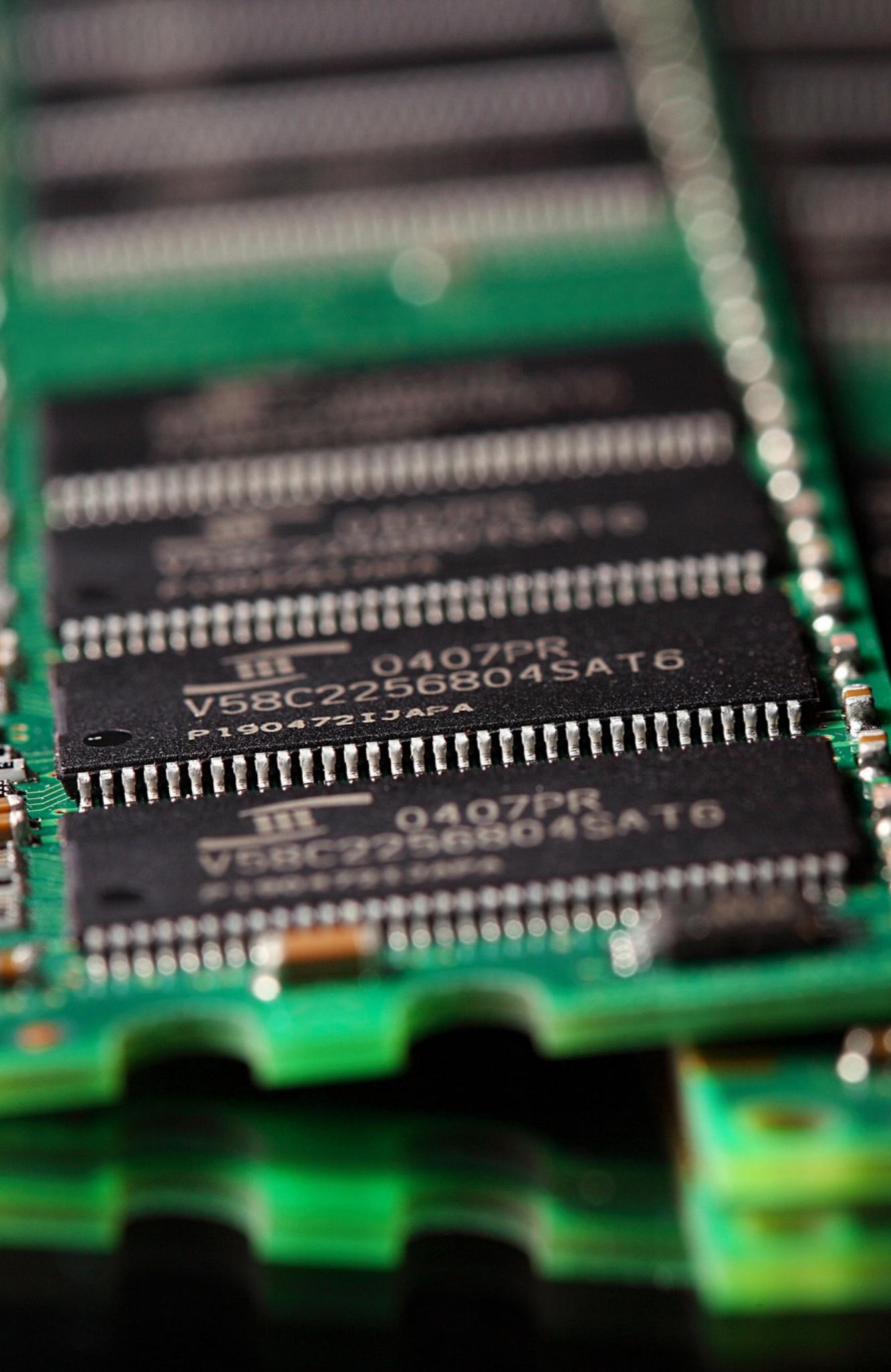
- Programming is abstracting.
- Abstract the world with:
 - Data types: 1.
 - Operations: $1 + 1$.
 - Control flow: if ... else.
- They are from:
 - Libraries.
 - Your own code.

Primitives

Data Types

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

Del

- `del x`
- Remove the pointing.

Checkpoint: Calculate BMR

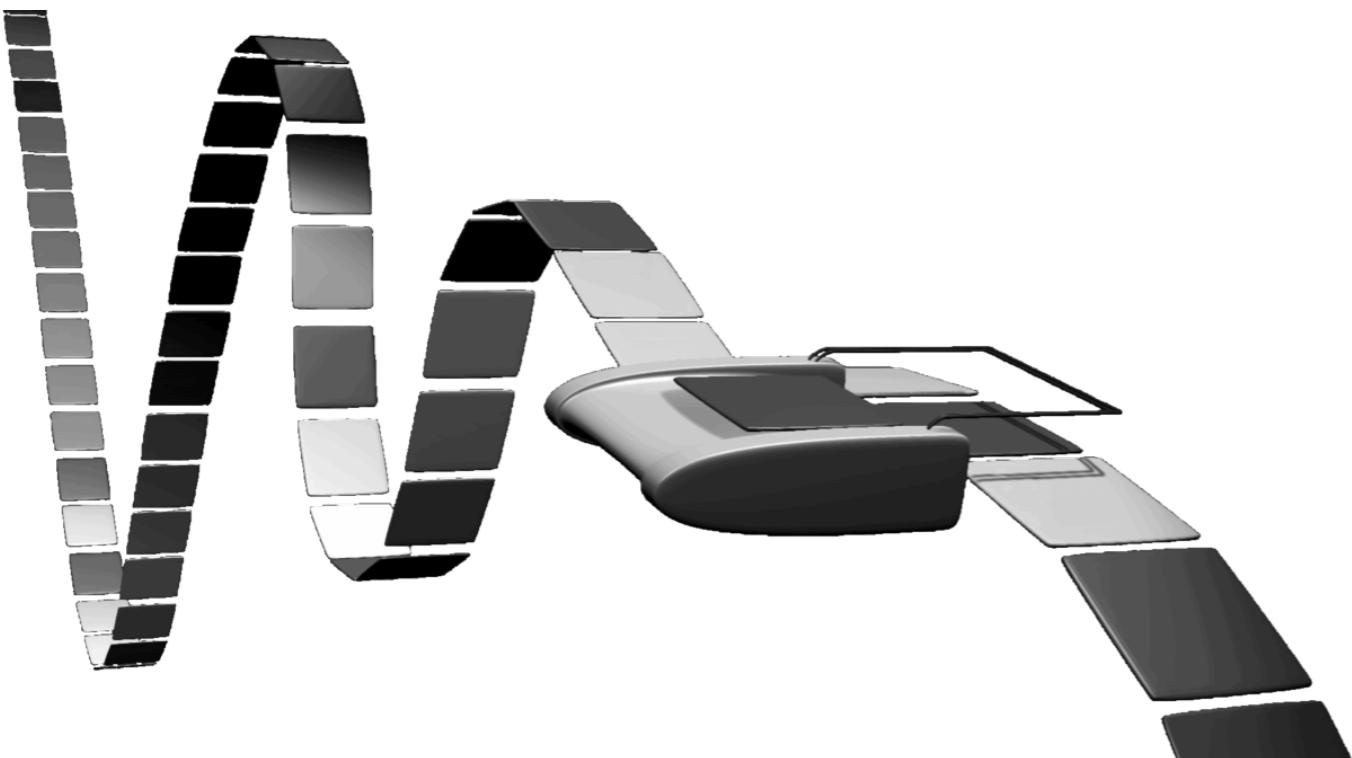
- Search “The Mifflin St Jeor Equation” in Basal metabolic rate – Wikipedia.
- Are you healthy enough?

If & While

Control Flow

If & While

- if
- if ... else
- if ... elif ... else
- while
 - The while-loop.
 - Do while true.



“

02_control_flow_if_while.ipynb

-The Notebook Time

Collections

Data Types

Collections

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_collections.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.

“

04_control_flow_for.ipynb

-The Notebook Time

Checkpoint: Calculate Average BMR by Gender

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

Try & Related Statements

- An exception stops the whole execution.
- Catch the exceptions and handle them.
- Sometimes stop is better than a bad output.

“

05_control_flow_try.ipynb

-The Notebook Time

Raise

- `raise`
 - `raise RuntimeError('should not be here')`
 - Raise an customized exception.
- `raise`
 - Re-raise the last exception.

Function

Control Flow

Function

INPUT x



OUTPUT $f(x)$

- Reuse statements.
- Extend built-in operations.
- def
 - Take the inputs.
- return
 - Give an output.
- If no return, returns None.
- Method
 - Is 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: $def f(a=None): \dots$
 - Arbitrary argument list: $def f(*args, **kwargs): \dots$
- 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

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 by Command Line Arguments

```
➤ $ python3 calc_bmr.py 152 48 63 1  
➤ 1120
```

The Challenges

Find the Largest File

- Input:
 - A path from command line.
- Output:
 - The name of the largest file in the directory.
 - Don't show the traceback to user.
- Snippets:
 - `from os import listdir`
 - `from os.path import join, isfile, getsize`

Find the Largest Directory

- Input and output are same as previous.
- Note the output shall be one of the first level directories.
- Snippets:
 - `from os import walk`
 - `from os.path import isdir, islink`
- Hints:
 - Which directory is the first level directory of a file?