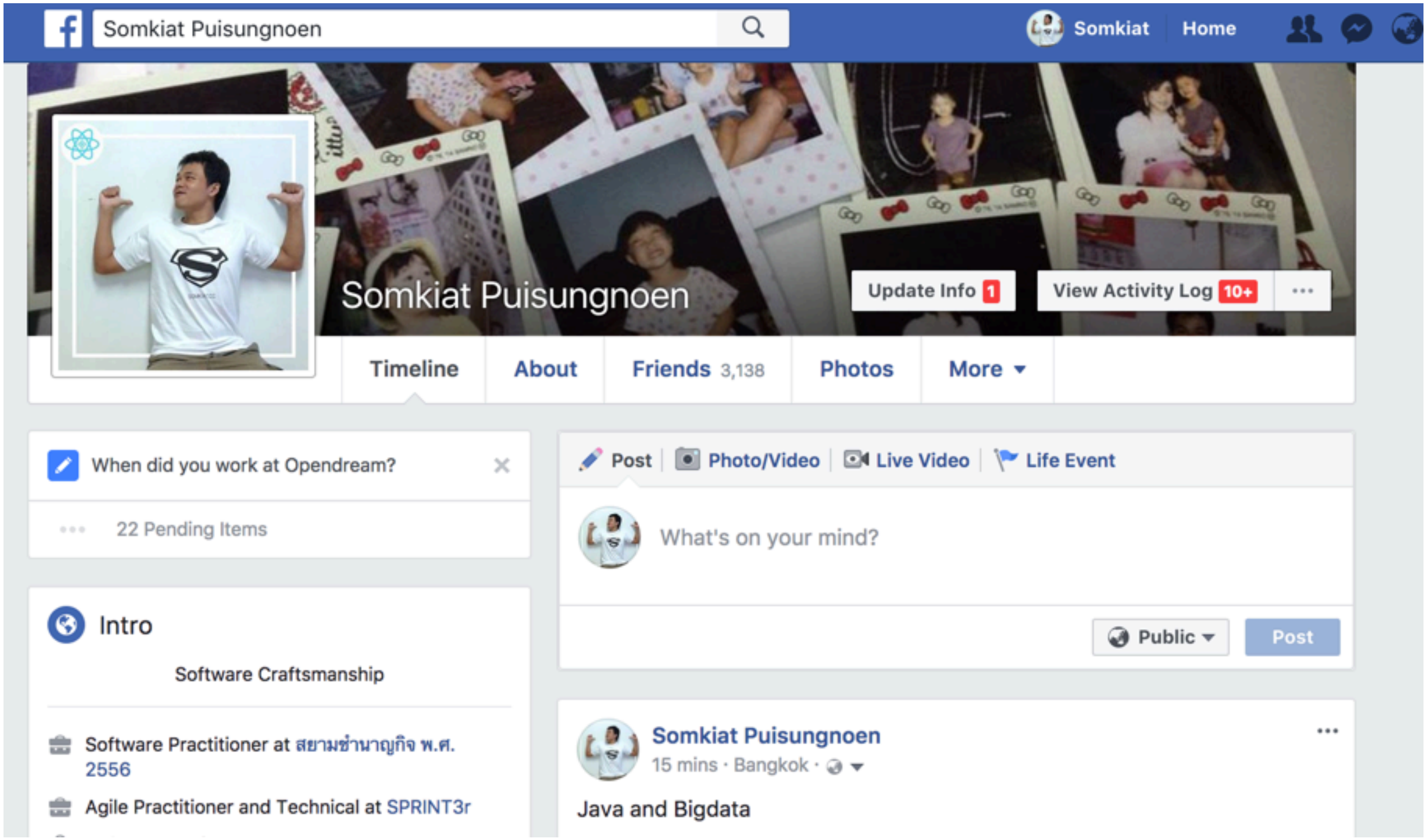
python



When did you work at Opendream?

22 Pending Items

Intro

Software Craftsmanship

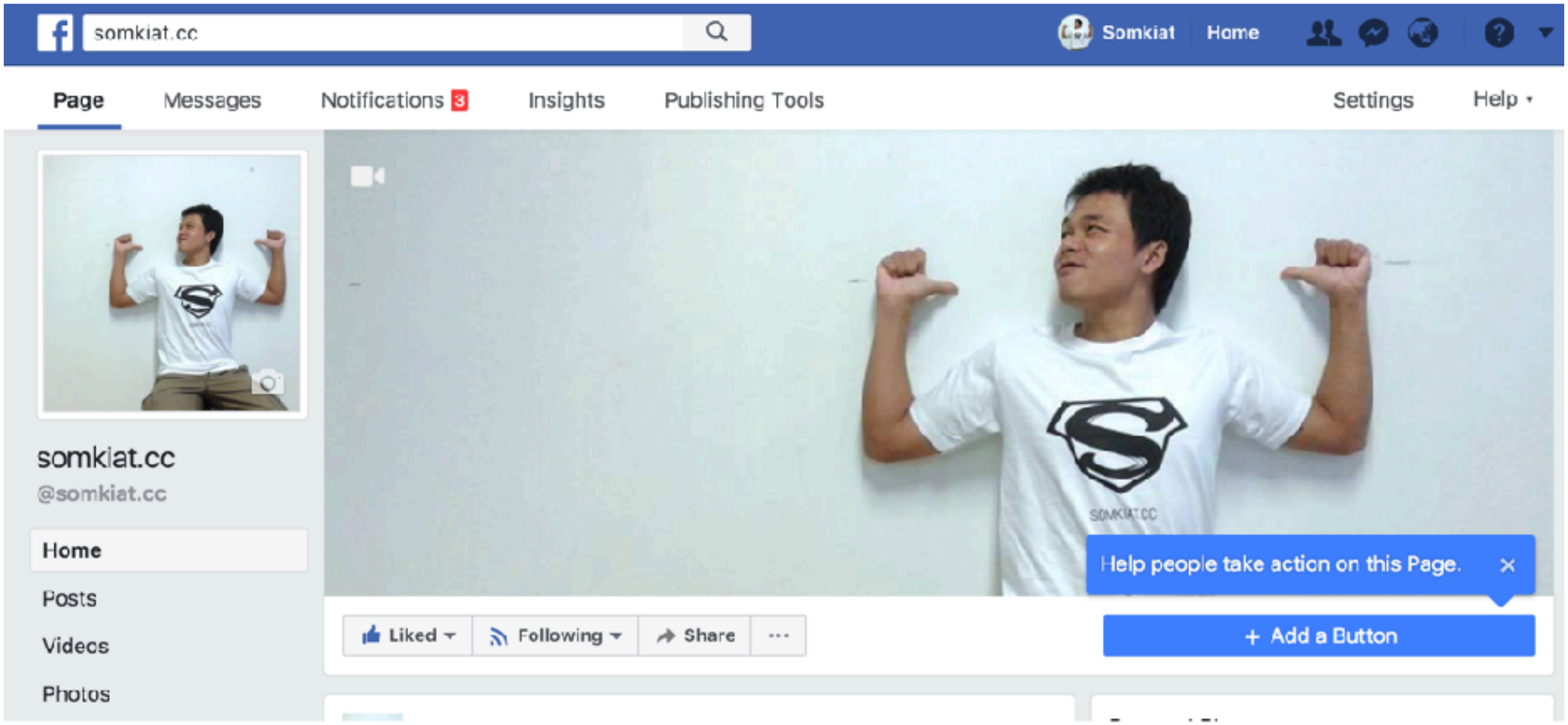
Software Practitioner at สยามชำนาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

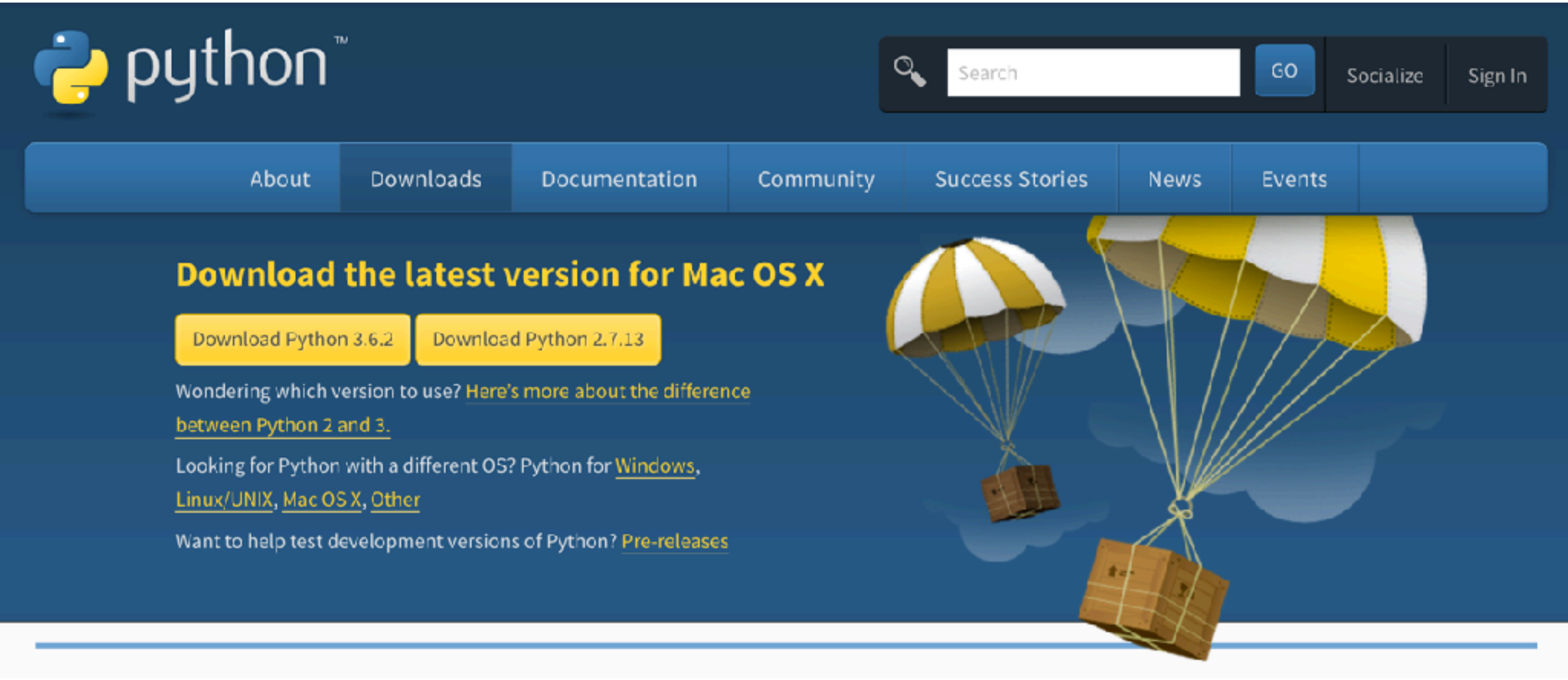
What's on your mind?

Somkiat Puisungnoen
15 mins · Bangkok ·

Java and Bigdata


somkiat.cc
Somkiat
Home
Page
Messages
Notifications 3
Insights
Publishing Tools
Settings
Help
somkiat.cc
@somkiat.cc
Home
Posts
Videos
Photos
Liked
Following
Share
Help people take action on this Page.
+ Add a Button

Installation



Download the latest version for Mac OS X

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

https://www.python.org/

First Question ?

https://wiki.python.org/moin/Python2orPython3

Overall Picture

Python 2.x is **Legacy**

Python 3.x is the **present and future**

For beginner ?

You should learn Python 2

More documents
More libraries and frameworks

Style Guide

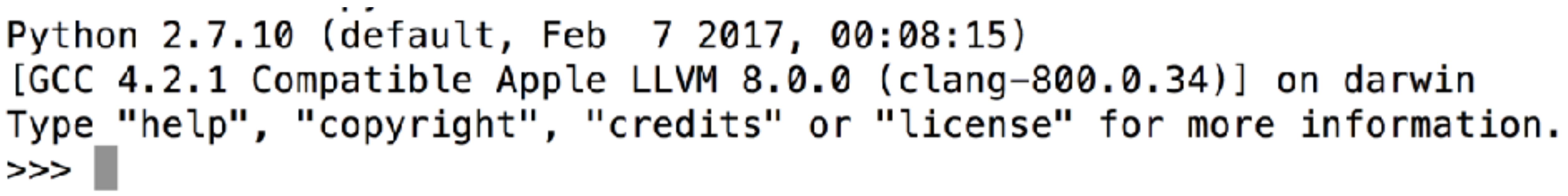
https://www.python.org/dev/peps/pep-0008/

long_function_name()
Classname
_private

Basic Python

Interactive mode

$python



```
Python 2.7.10 (default, Feb  7 2017, 00:08:15)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Interactive mode

```
>>help(list)
>>dir(list)
```

Online Help

1. Google with “python <name>”
2. Official Python Doc
3. Stackoverflow
4. Quora

Script mode

```
$python <file>.py
```

Jupyter notebook

Web application that allows you
to create and share documents
live code, document and visualization

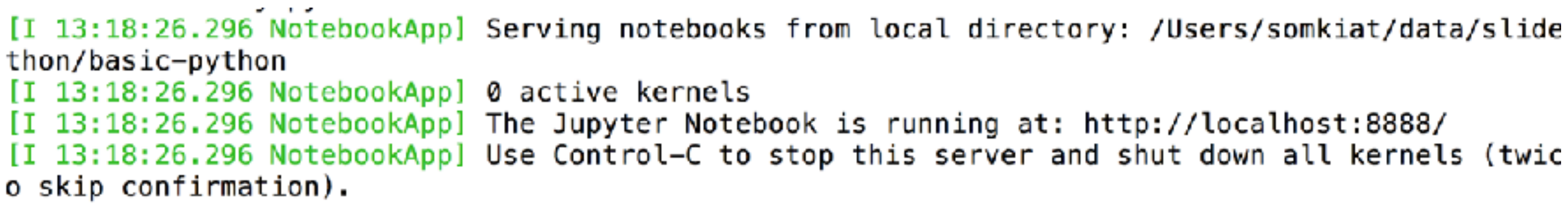
http://jupyter.org/

Install Jupiter notebook

```
$pip3 install --upgrade pip
$pip3 install jupyter
```

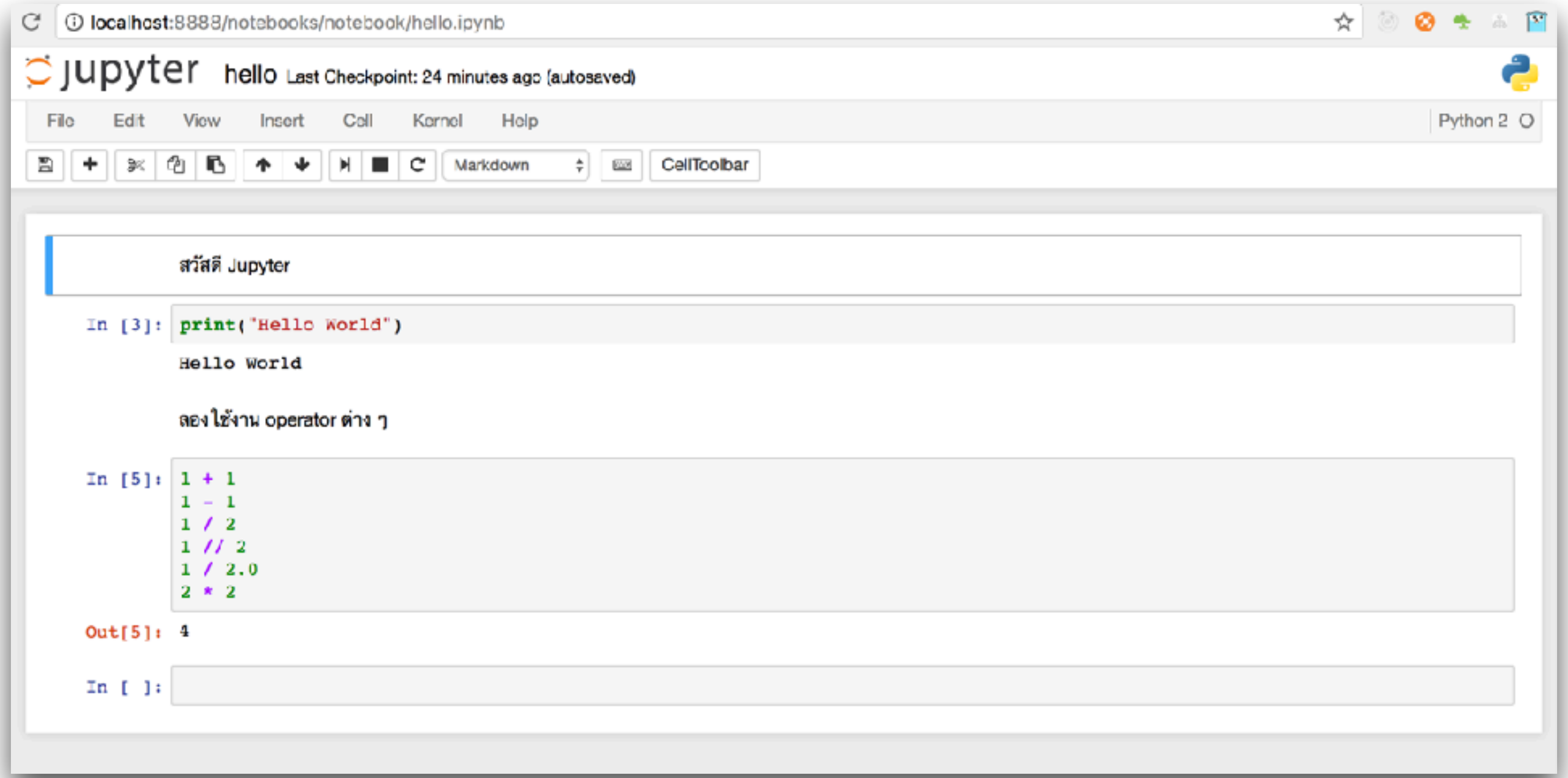
Starting Jupyter

$jupyter notebook



```
[I 13:18:26.296 NotebookApp] Serving notebooks from local directory: /Users/somkiat/data/slide
thon/basic-python
[I 13:18:26.296 NotebookApp] 0 active kernels
[I 13:18:26.296 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 13:18:26.296 NotebookApp] Use Control-C to stop this server and shut down all kernels (twic
o skip confirmation).
```

Using Jupyter


localhost:8888/notebooks/notebook/hello.ipynb
jupyter hello Last Checkpoint: 24 minutes ago (autosaved)
File Edit View Insert Cell Kernel Help
Python 2
Markdown
CellToolbar

สวัสดี Jupyter

```
In [3]: print("Hello World")

Hello World
```

ลองใช้งาน operator ต่าง ๆ

```
In [5]: 1 + 1
        1 - 1
        1 / 2
        1 // 2
        1 / 2.0
        2 * 2

Out[5]: 4

In [ ]:
```

Hello world

```
print('Hello World')
print('Hello' + ' ' + 'World')
print("Hello World")
print("Hello" + " " + "World")
```

print is a function in Python 3

Indentation

Python standard is **4 spacebars**

Indentation

Python 3 **disallows mixing** the use of tabs and spaces for indentation.

Comment

Start with **#**

Muli-line comment with """

Reserved words

and	exec	not	else
as	finally	or	lambda
assert	for	pass	yield
break	from	print	except
class	global	raise	del
continue	if	return	in
def	import	try	with

Waiting for the user

input(“Please any key to continue”)

Variable and Type

Variable and Type

Completely Object Oriented

Every variable in Python is an object

Not statically typed

Standard data types

Number
String
List
Tuple
Dictionary

Data type conversion

Function	Description
int(x [,base])	Converts x to an integer
float(x)	Converts x to a floating-point number
complex(real [,imag])	Creates a complex number
str(x)	Converts object x to a string representation
tuple(s)	Converts s to a tuple
list(s)	Converts s to a list
set(s)	Converts s to a set
dict(d)	Creates a dictionary
chr(x)	Converts an integer to a character

Basic Operator

Number

int
float
complex

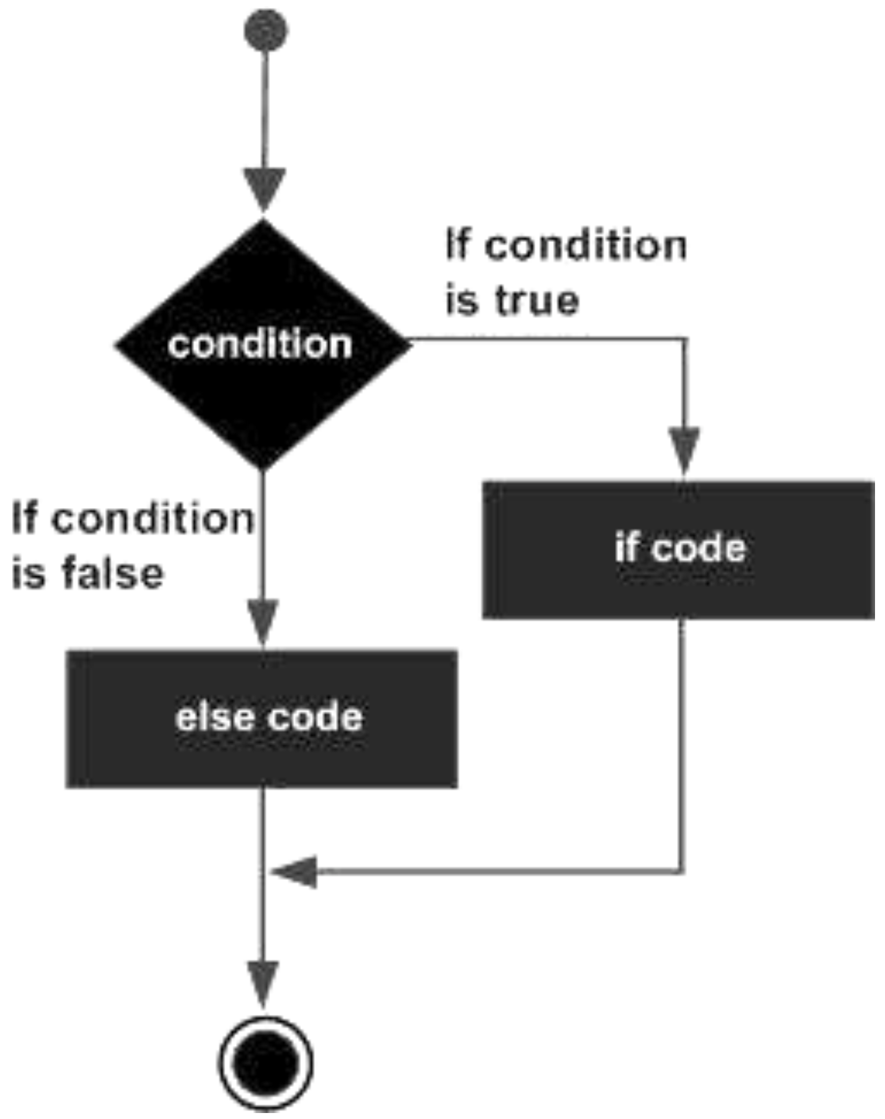
Int

In Python 3 => Unlimited size

Operation for number

Operation	Symbol	Example
Power (exponentiation)	**	5 ** 2 == 25
Multiplication	*	2 * 3 == 6
Division	/	14 / 3 == 4.666666666666667
Integer Division	//	14 / 3 == 4
Remainder (modulo)	%	14 % 3 == 2
Addition	+	1 + 2 == 3
Substraction	-	4 - 3 == 1

Decision making


If condition
is true
condition
If condition
is false
if code
else code

```python
score = int(input("Enter score: "))
if score >= 80:
    print("A")
elif score >= 70:
    print("B")
elif score >= 60:
    print("C")
elif score >= 50:
    print("D")
else:
    print("F")
```

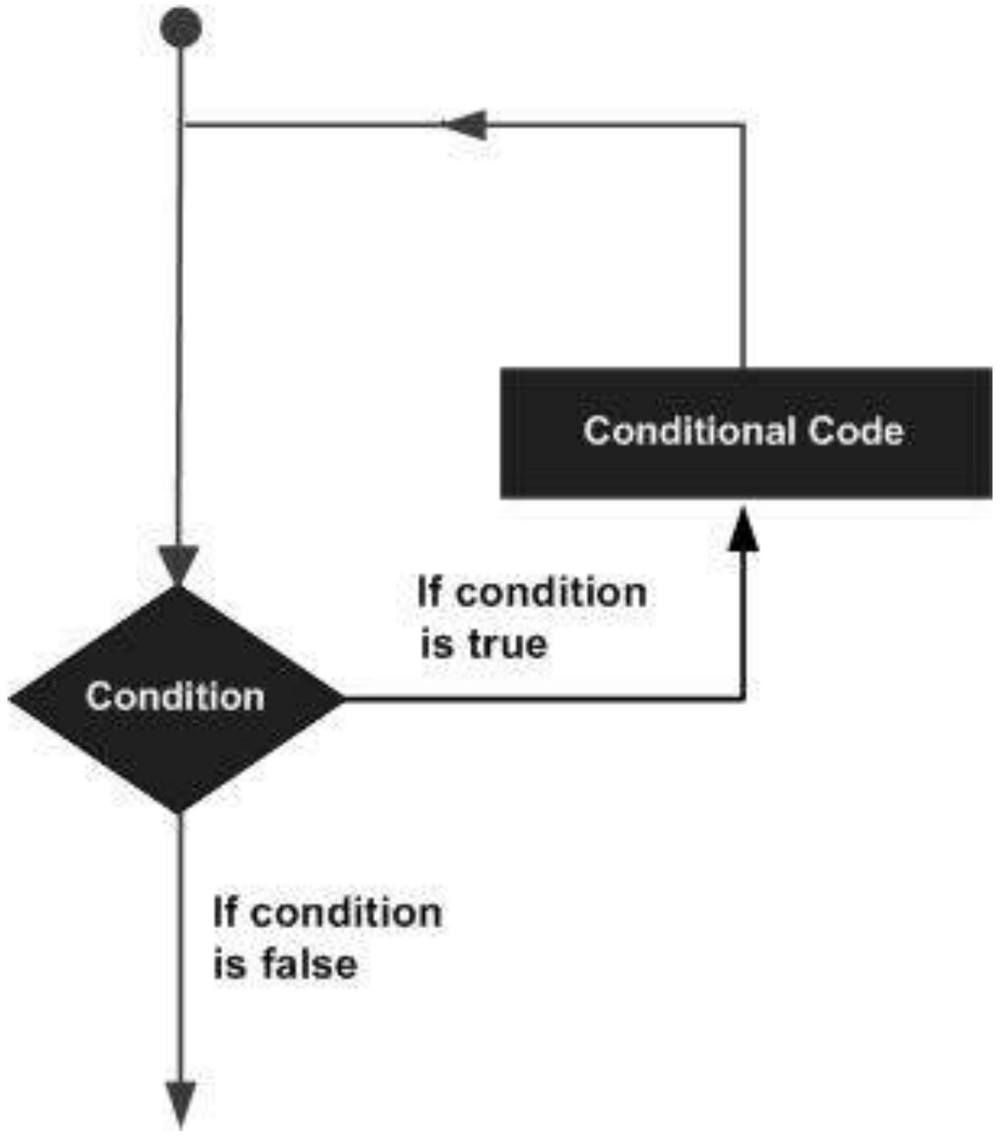
More operator

in operator

is operator

not operator

Loop


Conditional Code
If condition
is true
Condition
If condition
is false

Loop types

While loop
For loop
Nested loop

For loop

```
datas = range(0, 9)

for data in datas:
    print(data)
```

For loop

```python
names = ["Tom", "Mike", "Ko"]
for name in names:
    print(name)

for index in range(len(names)):
    print(names[index])
```

String

Operation for String

Operation	Symbol	Example
Repetition	*	"i" * 5 == “iiiii"
Concatenation	+	"Hello, " + "World" == "Hello, World"
Slice	[]	data[0]
Range slice	[:]	data[0:5]
Membership	in not in	"i" in “i love you" == True

List

Create List

```
empty = []
numbers = [1, 2, 3, 4, 5]
string = ["H", "e", "l", "l"]
mix = [1, 2, "three", True]

for data in mix:
    print(type(data))
```

Access List

```
mix = [1, 2, "three", True]
print(mix[0])
print(mix[1])
print(mix[1:3])
```

Operation for List

Operation	Symbol	Example
Length	len()	
Concatenation	+	[1] + [2] = [1, 2]
Repetition	*	[1] * 3 = [1, 1, 1]
Membership	in	1 in [1, 2, 3] = True
Range slice	[:]	

Tuple

Tuple

Immutable list
Can not delete or update data

Create Tuple

```
empty = ()
countries = ("Thai", "Indo", "China")
print(countries)
```

Dictionary

Dictionary

Key:Value data structure
Keys are unique
Keys must be of immutable data type
Values can be of any type

Create Dictionary

```
empty = {}
employee = {"firstname": "Somkiat",
            "lastname": "Pui",
            "age": 30}

print(employee)
```

Duplication key ?

```
data = {"Key1": "first", "Key1": "second"}
print(data)
```

Immutable key ?

```
# Immutable key
data = { 1: "first",
         "two": "second",
         (1, 2): "third" }
print(data)

# Mutable key
data = { [1,2]: "first" }
```

Function

Function

Block of reusable code
Single responsibility
All parameters are passed by reference

Create function

```
def say_hi(name):
    result = "Hello " + name
    return result

print(say_hi("Somkiat"))
```

Pass by reference

```
def try_to_change(data):
    data[2] = 300
    return

input = [0, 0, 0]
print("Before ", input)
try_to_change(input)
print("After ", input)
```

Function arguments

Required arguments
Keyword arguments
Default arguments
Variable-length arguments

Keyword arguments

```
def say_hi( name, age ):
    print("Hello %s, age = %d" %(name, age))

say_hi(name = "somkiat", age = 30)
say_hi(age = 30, name = "somkiat")
```

Default arguments

```
def say_hi2( name, age = 20 ):
    print("Hello %s, age = %d" %(name, age))

say_hi2(name = "somkiat")
say_hi2("somkiat")
```

Variable-length arguments

```
def sum(*numbers):
    result = 0
    for number in numbers:
        result = result + number
    return result

sum()
sum(1)
sum(1, 2)
sum(1, 2, 3)
sum(1, 2, 3, 4)
sum(1, 2, 3, 4, 5)
```

Workshop with factorial

RecursionError

RecursionError: maximum recursion depth exceeded in comparison

RecursionError (1,000)

```python
import sys

def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n-1)

# Max of recursion in python
print(sys.getrecursionlimit())
sys.setrecursionlimit(15000)
print(factorial(5000))
```

Modules

Modules

Organize your Python code
Grouping related code into a module
Easy to understand and use

Create new module

#file hello.py

```python
def say_hi():
    print("Say hi")

if __name__ == "__main__":
    say_hi()
```

Using module with import

#file caller.py

```
from hello import *

say_hi()
```

Locating Modules

Current directory

PYTHONPATH

Python installation directory

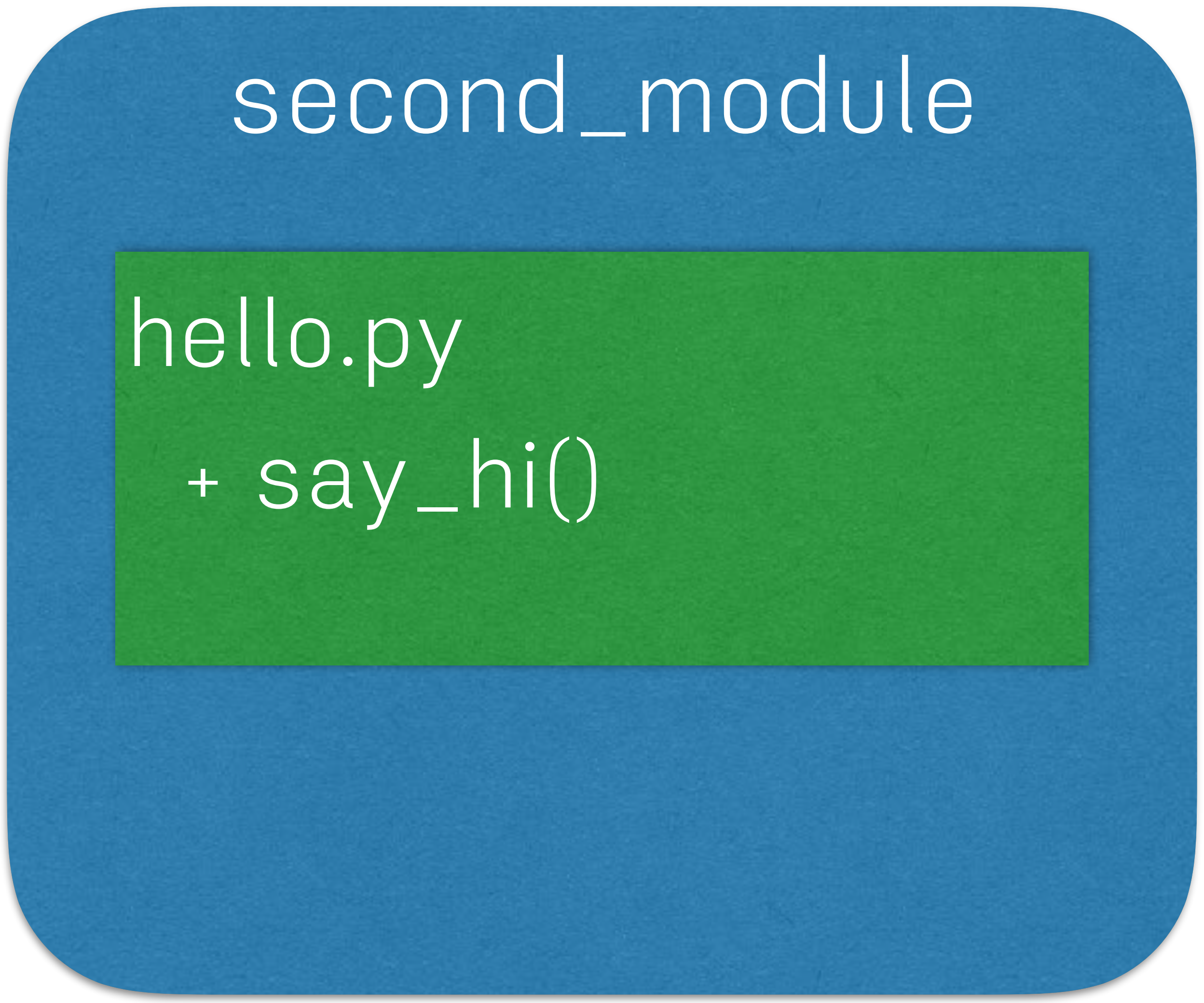
Locating Modules

```
import sys
from pprint import pprint

pprint(sys.path)
```

Workshop module

Workshop module



Module structure

```
.
├── caller.py
└── second_module
    ├── __init__.py
    └── hello.py
```

1. create folder second_module

2. create file hello.py

```python
def say_hi():
    print("From hello 1")
```

3. create caller.py outside the module folder

```
import sub01

sub01.say_hi()
```

4. create __init__.py inside module

```
from .hello import say_hi
```

Class

First class

```python
class Employee:
    class_variable = 0

    def __init__(self, id, name, age):
        self.id = id
        self.name = name
        self.age = age

    def get_data(self):
        return "Data of %s" % self.name


emp1 = Employee(1, "Somkiat", 30)
print(emp1.get_data())
```

Inheritance

```
class Base:
    def say_hi(self):
        print("From base")


class Child(Base):
    def say_hi(self):
        print("From child")

Base().say_hi()
Child().say_hi()
```

Operator overloading

```python
class MyNumber:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __add__(self,other):
        return MyNumber(self.a + other.a, self.b + other.b)

    def __str__(self):
        return 'MyNumber (%d, %d)' % (self.a, self.b)

num1 = MyNumber(1, 2)
num2 = MyNumber(10, 20)
print(num1 + num2)
```

Data hiding

```
class Hello:
    __counter_hiding = 0

    def count(self):
        self.__counter_hiding += 1
        return self.__counter_hiding

h = Hello()
print(h.count())

# print(h.__counter_hiding)
print(h._Hello__counter_hiding)
```

Getting start with test

pytest

Exception handling