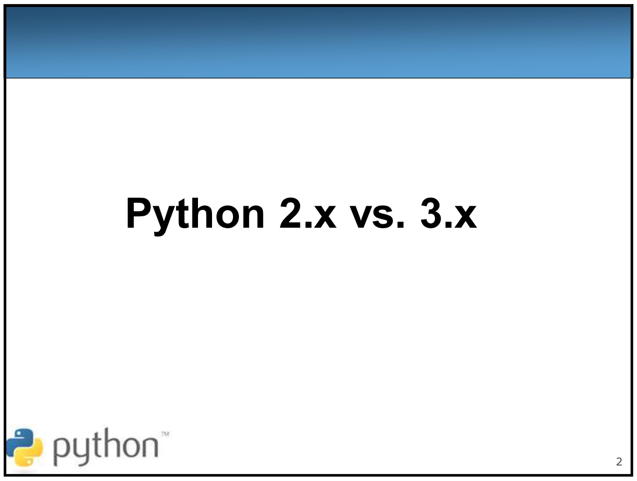
Programming with Python

Dr. Soharab Hossain Shaikh
BML Munjal University

https://github.com/soharabhossain/Python

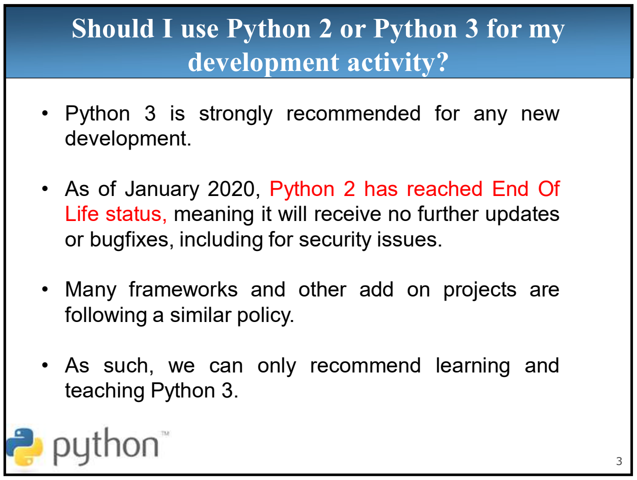
1



Python 2.x vs. 3.x

2

2

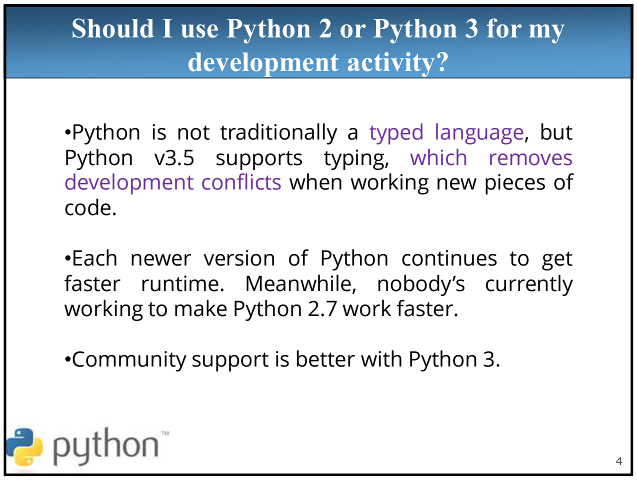


Should I use Python 2 or Python 3 for my development activity?

- Python 3 is strongly recommended for any new development.
- As of January 2020, Python 2 has reached End Of Life status, meaning it will receive no further updates or bugfixes, including for security issues.
- Many frameworks and other add on projects are following a similar policy.
- As such, we can only recommend learning and teaching Python 3.

3

3



Should I use Python 2 or Python 3 for my development activity?

•Python is not traditionally a typed language, but Python v3.5 supports typing, which removes development conflicts when working new pieces of code.

•Each newer version of Python continues to get faster runtime. Meanwhile, nobody's currently working to make Python 2.7 work faster.

•Community support is better with Python 3.

4

4

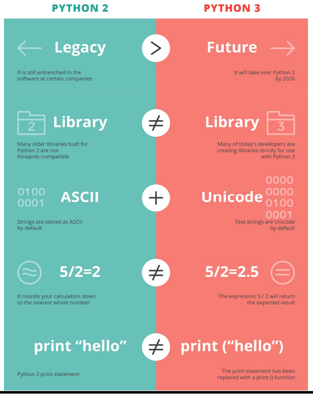
What are the differences?

History of Python 2

- Python 2.0 - October 16, 2000
- Python 2.1 - April 17, 2001
- Python 2.2 - December 21, 2001
- Python 2.3 - July 29, 2003
- Python 2.4 - November 30, 2004
- Python 2.5 - September 19, 2006
- Python 2.6 - October 1, 2008
- Python 2.7-July 3, 2010

History of Python 3

- Python 3.0 - December 3, 2008
- Python 3.1 - June 27, 2009
- Python 3.2 - February 20, 2011
- Python 3.3 - September 29, 2012
- Python 3.4-March 16, 2014
- Python 3.5 - September 13, 2015
- Python 3.6- October 2016
- Python 3.7- June 2018.



PYTHON 2 | PYTHON 3

Legacy > Future

It is still entrenched in the software at certain companies | It will take over Python 2 by 2020

Library ≠ Library

Many older libraries built for Python 2 are not forwards-compatible | Many of today's developers are creating libraries strictly for use with Python 3

ASCII + Unicode

Strings are stored as ASCII by default | Text strings are Unicode by default

5/2=2 ≠ 5/2=2.5

It rounds your calculation down to the nearest whole number | The expression 5 / 2 will return the expected result

print "hello" ≠ print ("hello")

Python 2 print statement | The print statement has been replaced with a print () function

What are the differences?

- xrange() of Python 2.x doesn't exist in Python 3.x.
- In Python 2.x, range returns a list i.e. range(3) returns [0, 1, 2] while xrange returns a xrange object i. e., xrange(3) returns iterator object which works similar to Java iterator and generates number when needed.
- If we need to iterate over the same sequence multiple times, we prefer range() as range provides a static list. xrange() reconstructs the sequence every time. xrange() doesn't support slices and other list methods.
- The advantage of xrange() is, it saves memory when the task is to iterate over a large range.
- In Python 3.x, the range function now does what xrange does in Python 2.x, so to keep our code portable, we might want to stick to using a range instead.
- So, Python 3.x's range function *is* xrange from Python 2.x.

The __***future***__ module

__future__ module

- The idea of the __future__ module is to help migrate to Python 3.x.
- If we are planning to have Python 3.x support in our 2.x code, we can use **_future_** imports in our code.

__future__ module

For example, in the Python 2.x code below, we use Python 3.x's integer division behavior using the __future__ module.

```
# In below python 2.x code, division works

# same as Python 3.x because we use  __future__

from __future__ import division

print 7 / 5

print -7 / 5
```

Output :

```
1.4

-1.4
```

__future__ module

Another example where we use brackets in Python 2.x using __future__ module

```
from __future__ import print_function

print("Hello BMU")
```

https://docs.python.org/2/library/__future__.html

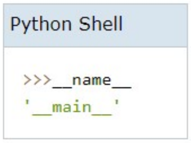
__name__ & __main__

__name__ & __main__

- A Program written in languages of C family (C, C++, Java, C# etc.) needs the main() function to indicate the starting point of execution.
- In Python, on the other hand, there is no concept of the main() function, as it is an interpreter-based language and can be equally used in an interactive shell.
- The Python program file with .py extension contains multiple statements. The execution of the Python program file starts from the first statement.
- Python includes the special variable called **__name__** that contains the scope of the code being executed as a string.
- **__main__** is the name of the top-level scope in which top-level code executes.

__name__ & __main__

- For example, the scope of the code executed in the interpreter shell will be **__main__,** as shown below.


Python Shell

```
>>>__name__
'__main__'
```

__name__ & __main__

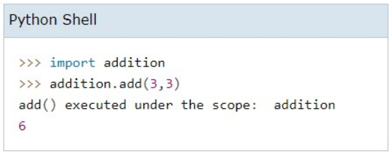
- The value of the **__name__** allows the Python interpreter to determine whether a module is intended to be an executable script or not.
- If its value is **__main__,** the statements outside function definitions will be executed.
- If not, the contents of the module are populated in top-level module (or interpreter namespace) without the executable part.

- Note: The Python script file executing from the command prompt/terminal will be executed under the top-level scope **__main__** scope.
- However, importing a module will be executed under the module's own scope. So, the top-level scope will be **__main__**, and the second scope would be module's scope.

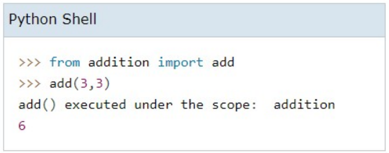
__name__ & __main__

```
# Script stored as addition.py

def add(x, y):
    z=x+y
    print('add() executed under the scope: ', __name__)
    return z

if __name__ == '__main__':
    x=input('Enter the first number to add: ')
    y=input('Enter the secode number to add: ')
    result = add(int(x),int(y))
    print(x, '+', y,'=', result)
    print('Code executed under the scope: ', __name__)
```

Executing from Python Shell

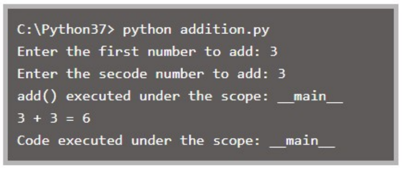

Python Shell

```
>>> import addition
>>> addition.add(3,3)
add() executed under the scope:  addition
6
```


Python Shell

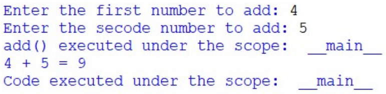
```
>>> from addition import add
>>> add(3,3)
add() executed under the scope:  addition
6
```

Executing from Command Prompt



```
C:\Python37> python addition.py
Enter the first number to add: 3
Enter the secode number to add: 3
add() executed under the scope: __main__
3 + 3 = 6
Code executed under the scope: __main__
```

Similar output is obtained when we run the code from any code editor like IDLE with F5.



```
Enter the first number to add: 4
Enter the secode number to add: 5
add() executed under the scope:  __main__
4 + 5 = 9
Code executed under the scope:  __main__
```

__name__ & __main__

- Thus, using the special variable **__name__** and the top-level scope **__main__** increases the reusability.
- The Python script file can be executed from the command prompt/terminal as an independent script as well as when imported as a module.

End of Presentation