Py2K20 - Transitioning from Python2 to Python3

Author: Joff Thyer, March 2019

BLACK HILLS
Information Security
• 2008 •

PENETRATION TESTING & RED TEAMING

www.blackhillsinfosec.com

Who is this Joff guy?

- Joff Thyer
  - Penetration Tester, Researcher, Developer at Black Hills Information Security
  - SANS Certified Instructor
  - SEC573: Automating Information Security with Python
  - Security Weekly Co-Host
  - Twitters: @joff_thyer

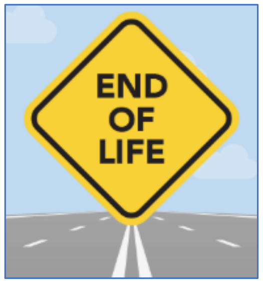
https://www.sans.org/course/automating-information-security-with-python

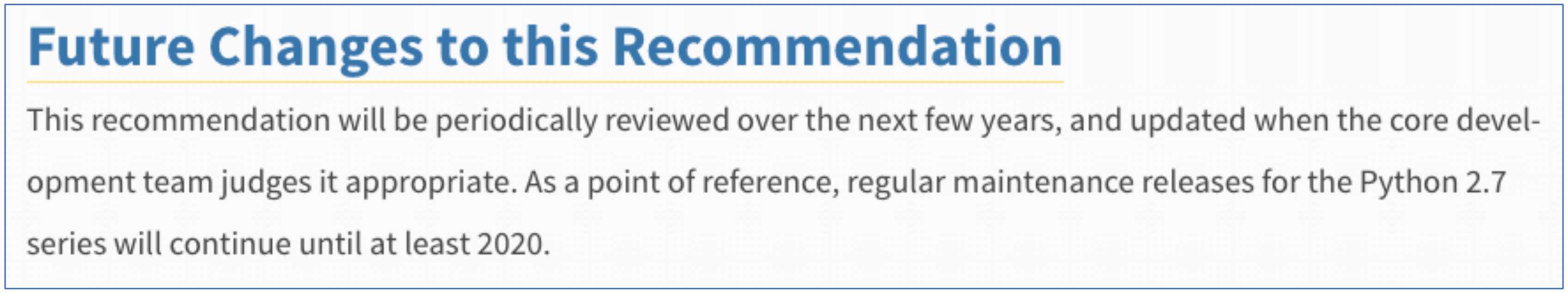
Agenda for today!

- What is happening, Linux Distros, and Potential Impact
- The “__future__” module and Python 2.7.x
- Language Differences
  - Keywords
  - Mathematical Division
  - Print Function
  - Format Strings
  - Keyboard Input
  - String Object Changes (UTF-8)
  - File Handling
  - Memory Saving Enhancements
  - Variable Scope
- Wrapping Up, Summary and Questions

Python2 becomes EOL – What changes?

- PEP394 states that the "python" command on Linux invokes Python2.
- Maintenance releases of Python 2.7 will probably cease in 2020.



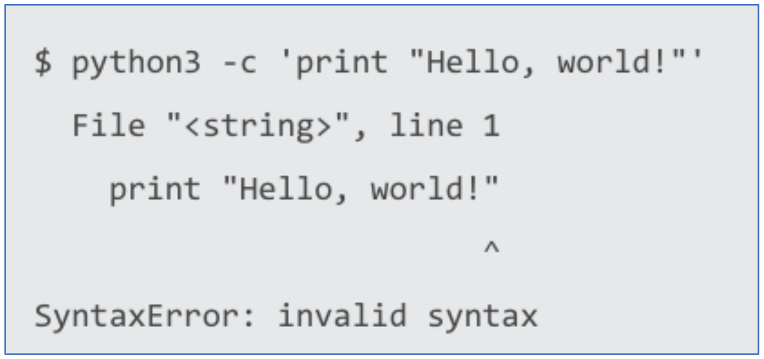


Future Changes to this Recommendation

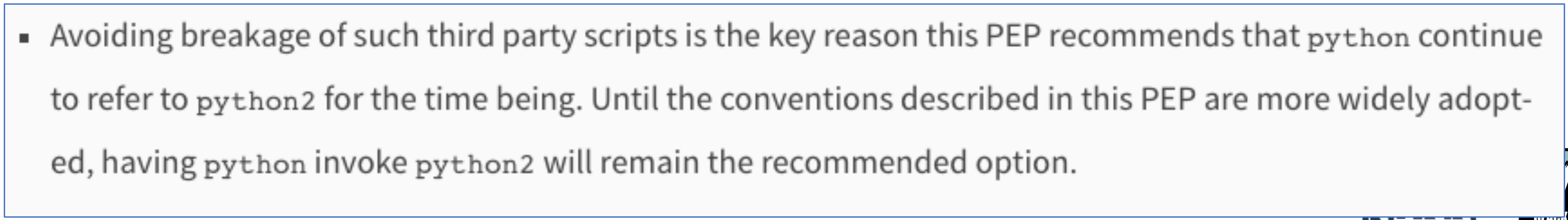
This recommendation will be periodically reviewed over the next few years, and updated when the core development team judges it appropriate. As a point of reference, regular maintenance releases for the Python 2.7 series will continue until at least 2020.

What will Linux distros do?

- I suspect *nothing*
- *Why?* The main barrier to changing to Python3 is breakage of 3rd party scripts, and packages



```
$ python3 -c 'print "Hello, world!"'
  File "<string>", line 1
    print "Hello, world!"
                        ^
SyntaxError: invalid syntax
```



- Avoiding breakage of such third party scripts is the key reason this PEP recommends that `python` continue to refer to `python2` for the time being. Until the conventions described in this PEP are more widely adopted, having `python` invoke `python2` will remain the recommended option.

If Linux distros make the move

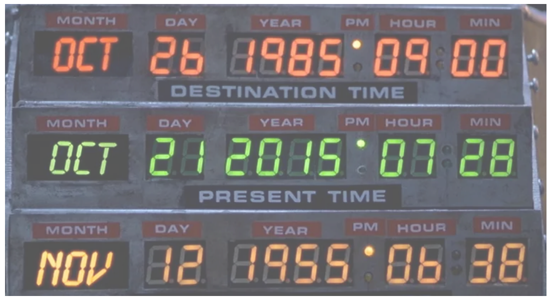
- What does it really mean?
- A symbolic link will change such that:
  - python -> python3

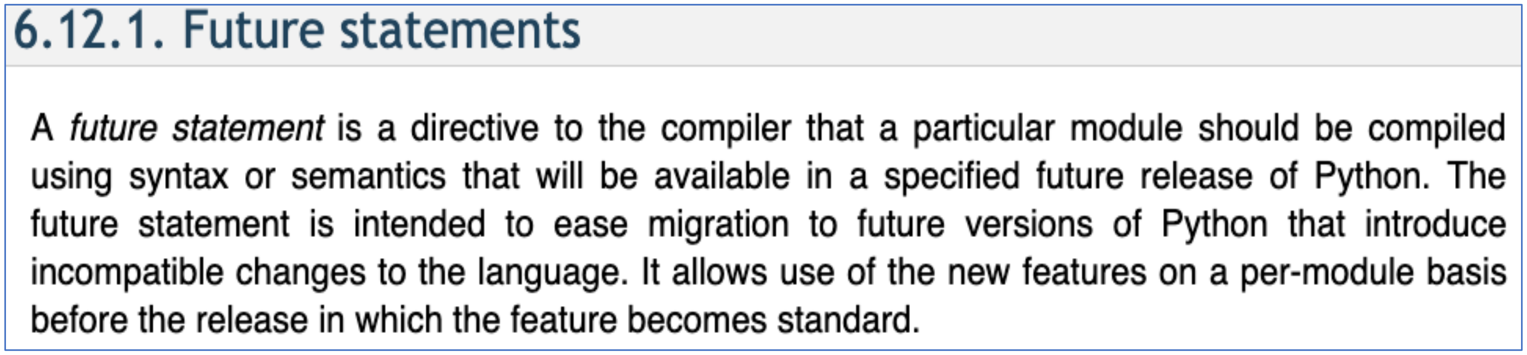
- How much do you care?
  - Change symbolic link back to python2
  - Problem solved. *Until some package update changes it again.*

Back to the __future__

- Example Python statements:

```
from __future__ import division
from__future__ import print_function
```



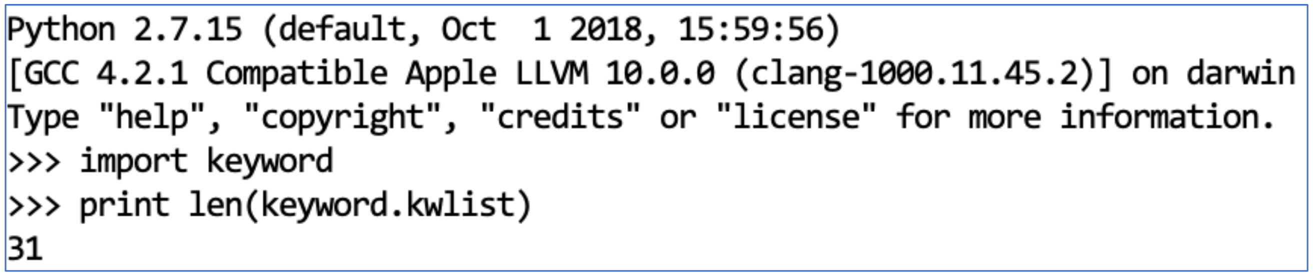


6.12.1. Future statements

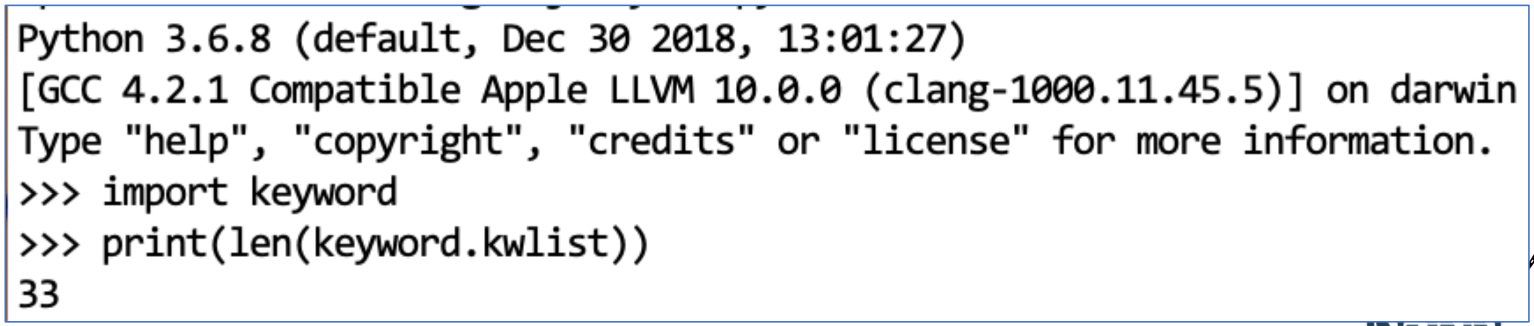
A *future statement* is a directive to the compiler that a particular module should be compiled using syntax or semantics that will be available in a specified future release of Python. The future statement is intended to ease migration to future versions of Python that introduce incompatible changes to the language. It allows use of the new features on a per-module basis before the release in which the feature becomes standard.

Core Language Keywords

- True, False, and None become Keywords in Python3
- ”print” becomes a function rather than Keyword.



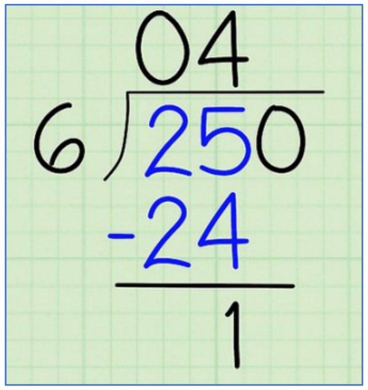
```
Python 2.7.15 (default, Oct  1 2018, 15:59:56)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import keyword
>>> print len(keyword.kwlist)
31
```

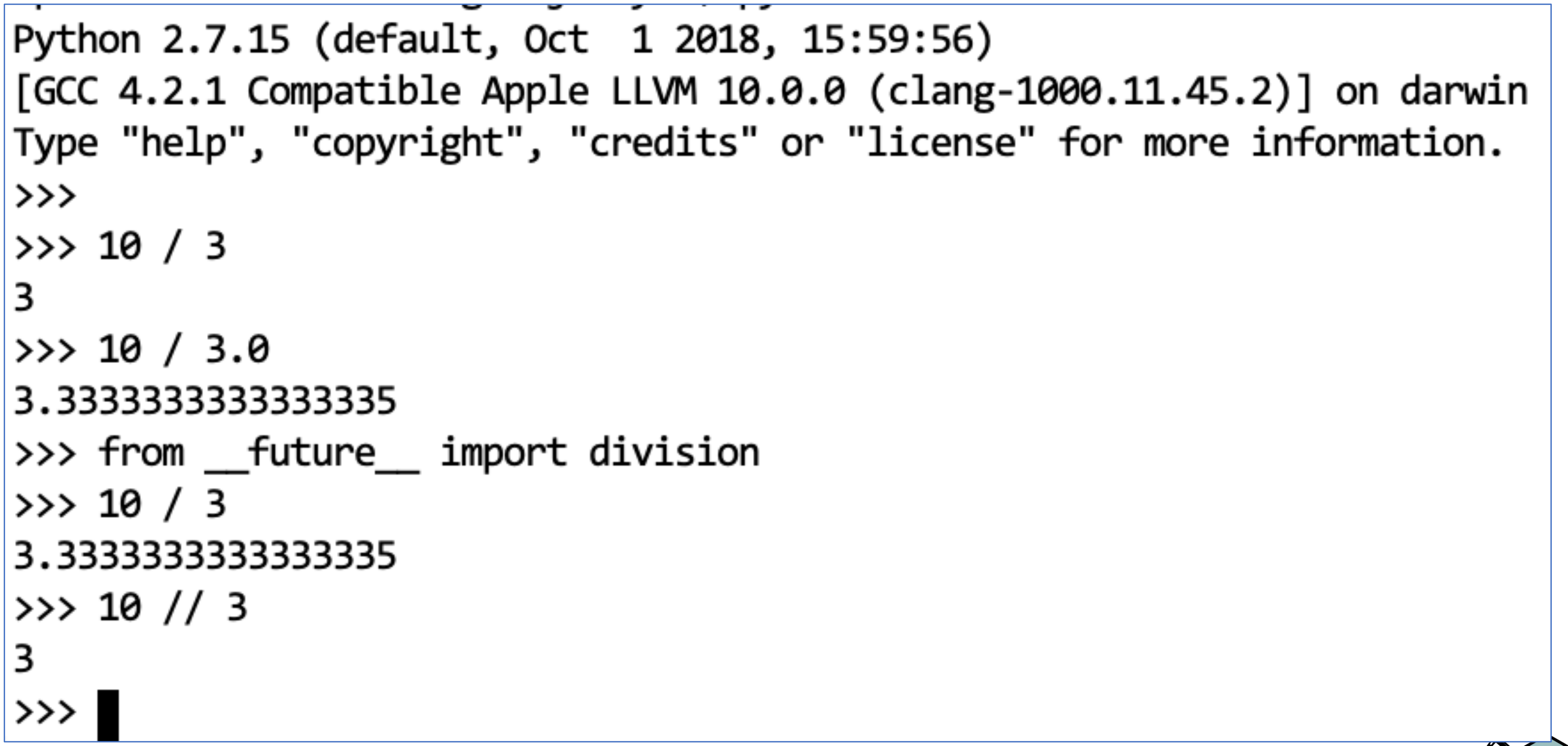


```
Python 3.6.8 (default, Dec 30 2018, 13:01:27)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import keyword
>>> print(len(keyword.kwlist))
33
```

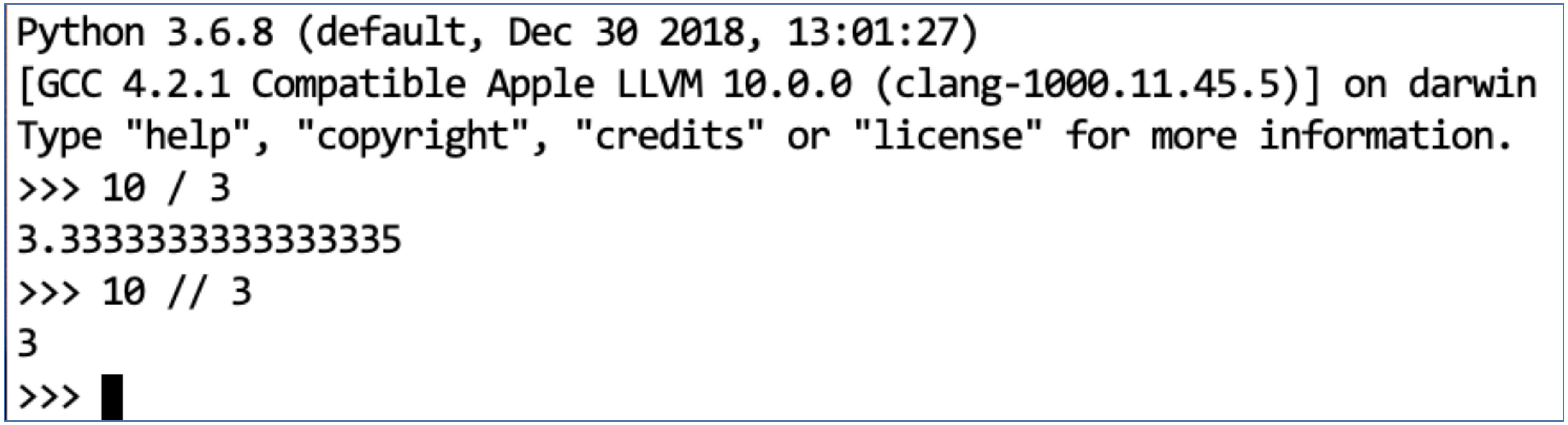
Division is not what you think it is…

- Python2
  - Division is the same as the “floor” operator
  - If either dividend or divisor are float(), then result is float
  - If only INTEGERS used, then result is integer.
- Python3
  - All division is floating point unless you explicitly use the floor operator.





```
Python 2.7.15 (default, Oct  1 2018, 15:59:56)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 10 / 3
3
>>> 10 / 3.0
3.3333333333333335
>>> from __future__ import division
>>> 10 / 3
3.3333333333333335
>>> 10 // 3
3
>>>
```



```
Python 3.6.8 (default, Dec 30 2018, 13:01:27)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 / 3
3.3333333333333335
>>> 10 // 3
3
>>> 
```

PRINT becomes a function().

- In Python2, print was a “keyword” as part of the core interpreter
- In Python3, print() has become a function

- What does this mean?
  - The syntax: print “hello”   <<- this no longer works
  - Replace with: print(“hello”)

- But in Python2 that means we are printing a tuple.
  - So what do we do for “forward compatibility”

    from future import print_function

Print Function and End of Line Character

- In Python2 you could suppress line endings by appending a comma to the end of the print statement.
- In Python3, you have flexibility on line endings using the “end=“ parameter to the print function.

Python2 Style Format Strings

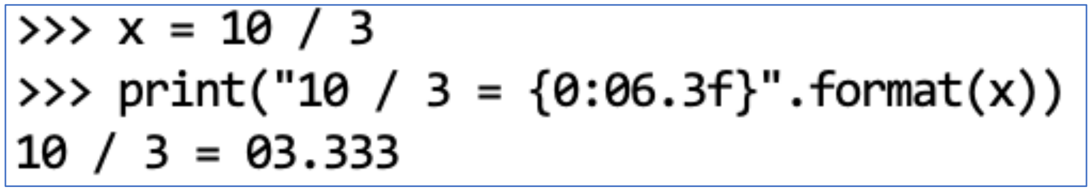
- Python2 uses the “C” and “C++” language style format strings.
- These also work in Python3.
  - %d = decimal / integer
  - %f = floating point
  - %x = hex lowercase
  - %X = hex uppercase
  - %s = string
- Example code:

```
a = “Joff Thyer”
x = “Hello %s, What is your favorite color?” % (a)
print(x)
```

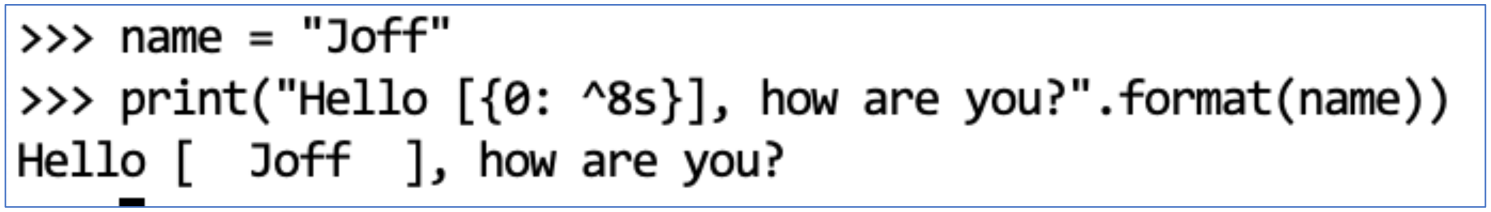
Python3 Style Format Strings

- Adopts the C# style/syntax
- Syntax is as follows:
  - {Arg#: <fill character><alignment><length><type>}
  - Where:
    - Arg# is the argument number in the format() method
    - Fill character is the character to fill the string when aligning
    - Alignment can be either “^”, “<“, “>” for center, left, and right
    - Length is the length of the string when aligned
    - Type is the SAME as Python2 style format string types:
      - s = string
      - d = decimal/integer
      - x = hexadecimal
      - f = float

Python3 Format String Examples



```
>>> x = 10 / 3
>>> print("10 / 3 = {0:06.3f}".format(x))
10 / 3 = 03.333
```



```
>>> name = "Joff"
>>> print("Hello [{0: ^8s}], how are you?".format(name))
Hello [  Joff  ], how are you?
```

raw_input() is absent in Python3

- Python3 only has the input() function
- It is the same as “raw_input()” in Python2.

- NOTE: input() in Python2 works also but will eval() what ever is entered.
  - Using this will result in Python2 code injection vulnerabilities.

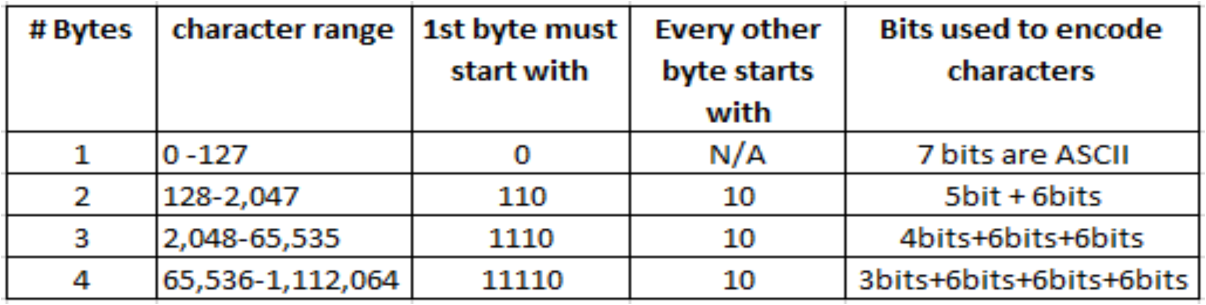
Strings are no longer just byte sequences

- Python3 treats all strings as UTF-8 encoded
- Python2 treats all strings as just sequences of bytes

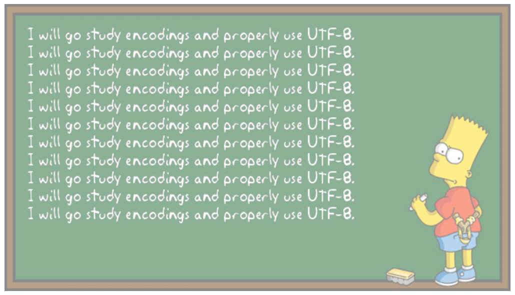
- This one aspect of Python3 is typically what holds people back from adopting the new version!

What is UTF-8?

- A space saving encoding for strings
- Can store characters from integer 0 up through 1,112,064
  - Remember, everything is stored as bytes on disk and in memory!!
  - Think: “man ascii” but now we have unicode



# Bytes	character range	1st byte must start with	Every other byte starts with	Bits used to encode characters
1	0 -127	0	N/A	7 bits are ASCII
2	128-2,047	110	10	5bit + 6bits
3	2,048-65,535	1110	10	4bits+6bits+6bits
4	65,536-1,112,064	11110	10	3bits+6bits+6bits+6bits


I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.

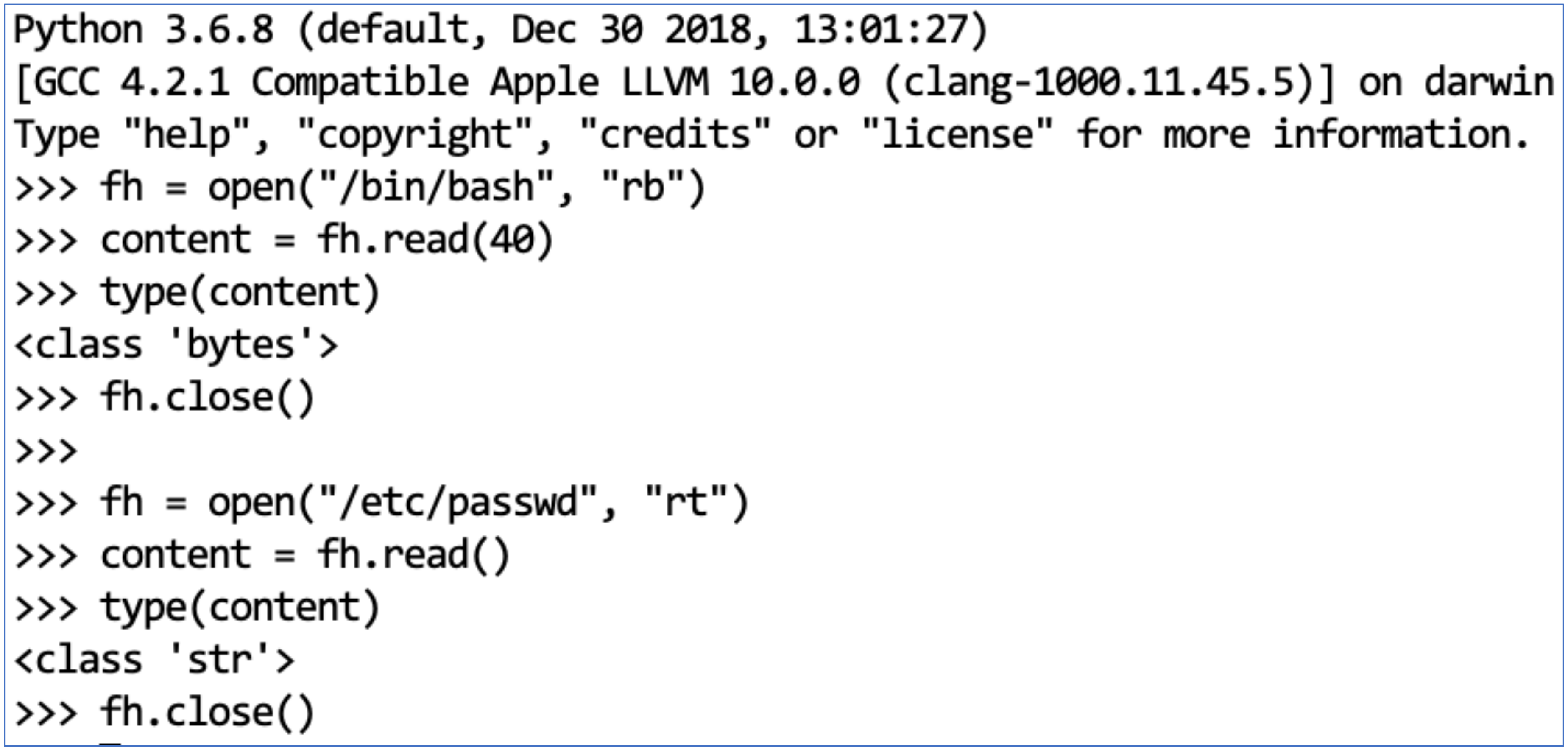
Python3 has a bytes() object now

- Bytes() in Python3 are compatible with strings in Python2
  - There is now a need to convert between bytes() and strings.

- The methods named ”encode()”, and “decode()” that are part of the string object have been re-purposed in Python3.
  - Python2 used then for codecs such as “base64”, “rot13” etc.
  - All of the codecs functionality has now moved to the codecs module
- Python3 string **encode()** method now converts strings to bytes
- The Python3 byte **decode()** method now converts bytes back to strings

File Object Modes

- Normal modes for opening files include:
  - r = read, w = write
  - a = append, r+ = read and write
- Python3 adds a text, and binary mode as additional qualifiers
  - In text mode, the default UTF-8 encoding is assumed for files
  - In binary mode, all file contents are treated as a byte object
- For text mode, Python3 will properly handle line endings
  - UNIX/Linux…: Linefeed only (\n)
  - Windows….. : Carriage Return and Linefeed (\r\n)



```
Python 3.6.8 (default, Dec 30 2018, 13:01:27)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> fh = open("/bin/bash", "rb")
>>> content = fh.read(40)
>>> type(content)
<class 'bytes'>
>>> fh.close()
>>>
>>> fh = open("/etc/passwd", "rt")
>>> content = fh.read()
>>> type(content)
<class 'str'>
>>> fh.close()
```

Memory Savings in Object Generation

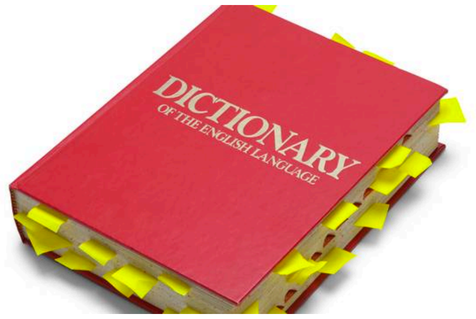
- Python3 has moved away from being so “list intensive” for different built-in function return values
  - Iterable generator objects are usually favored instead
  - Meaning you can loop through return value only.

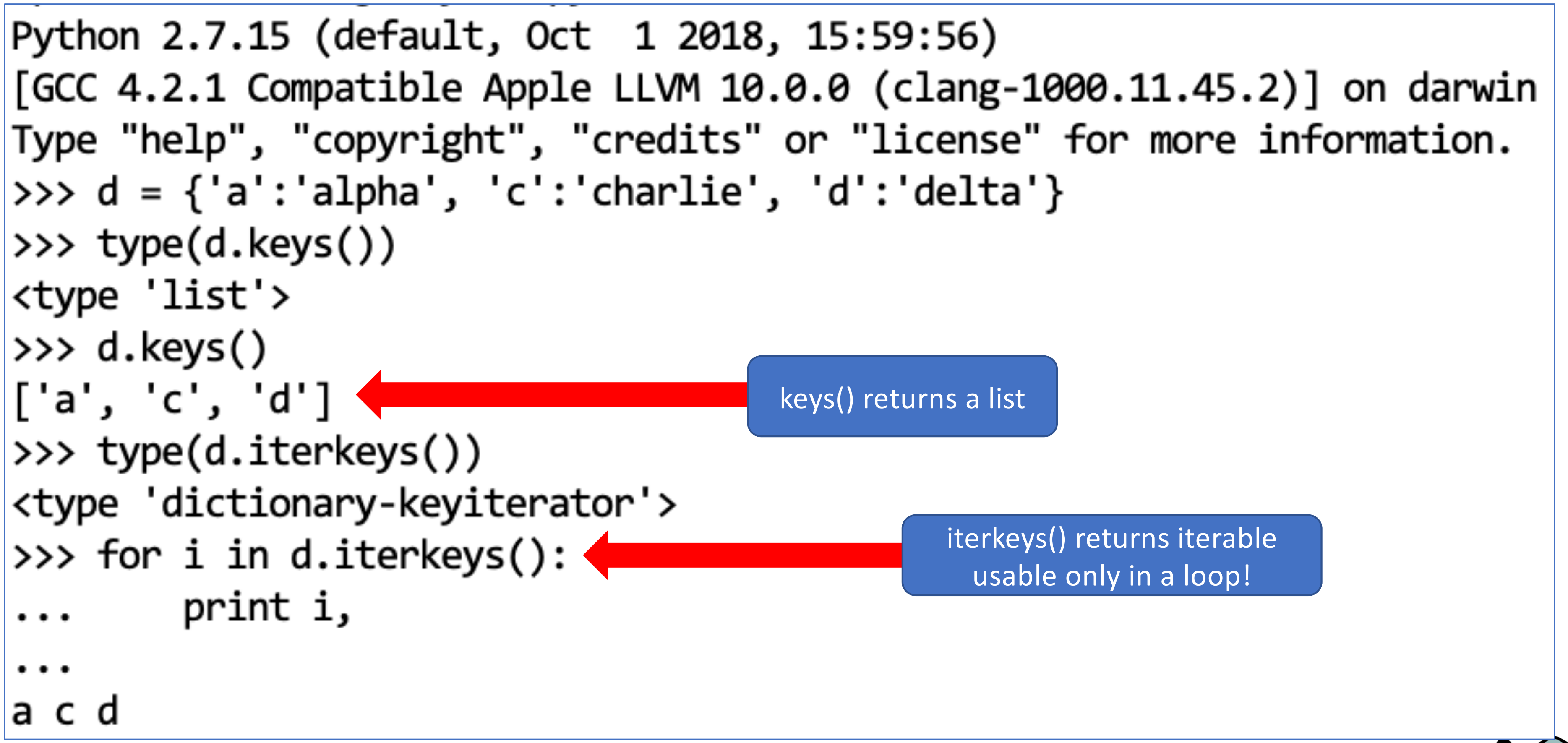
- Examples of functions that used to return lists in Python2:
  - range(start, end, step)
    - Python3: returns an iterable range object
  - map(func, <list1>, <list2>, …)
    - Python3: returns an iterable map object

Dictionary Changes

- Python2 had different methods that return lists or iterable objects in dictionaries.
  - a = dict()
  - a.keys(), a.values(), and a.items() all return lists
  - a.iterkeys(), a.itervalues(), and a.iteritems() all return iterable objects

- Python3 dictionary methods all return an iterable object called a view!
  - a = dict()
  - a.keys(), a.values(), and a.items() all return iterable view objects

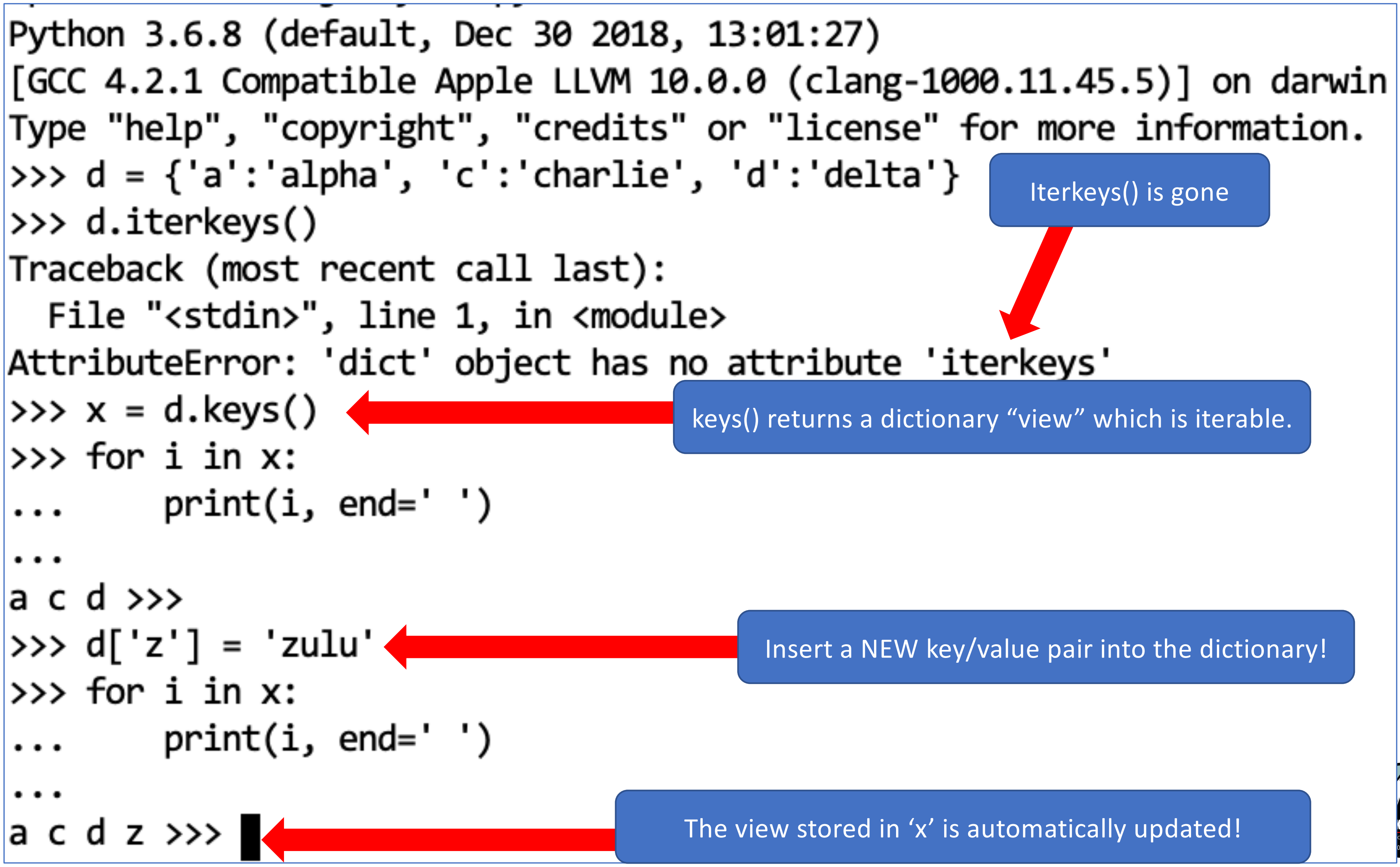




```
Python 2.7.15 (default, Oct  1 2018, 15:59:56)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> d = {'a':'alpha', 'c':'charlie', 'd':'delta'}
>>> type(d.keys())
<type 'list'>
>>> d.keys()
['a', 'c', 'd']
>>> type(d.iterkeys())
<type 'dictionary-keyiterator'>
>>> for i in d.iterkeys():
...     print i,
...
a c d
```

keys() returns a list

iterkeys() returns iterable usable only in a loop!



```
Python 3.6.8 (default, Dec 30 2018, 13:01:27)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> d = {'a':'alpha', 'c':'charlie', 'd':'delta'}
>>> d.iterkeys()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'iterkeys'
>>> x = d.keys()
>>> for i in x:
...     print(i, end=' ')
...
a c d >>>
>>> d['z'] = 'zulu'
>>> for i in x:
...     print(i, end=' ')
...
a c d z >>> █
```

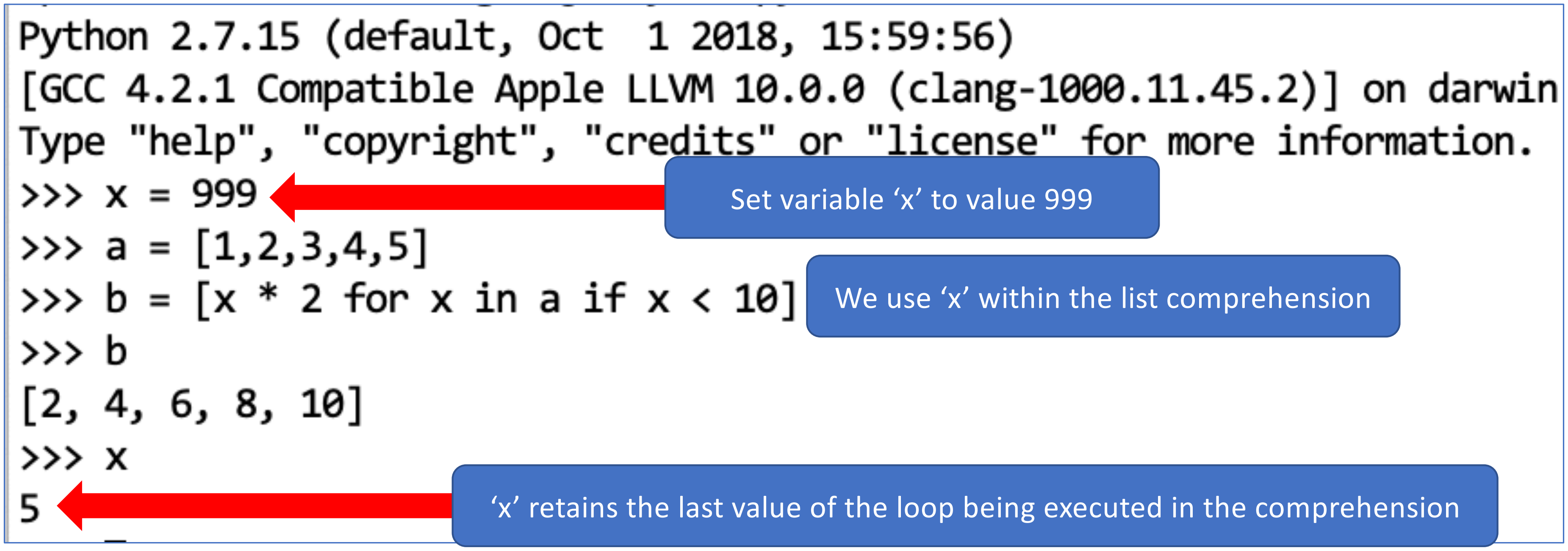
Iterkeys() is gone

keys() returns a dictionary “view” which is iterable.

Insert a NEW key/value pair into the dictionary!

The view stored in ‘x’ is automatically updated!

Variable Scope in Python2 List Comprehension

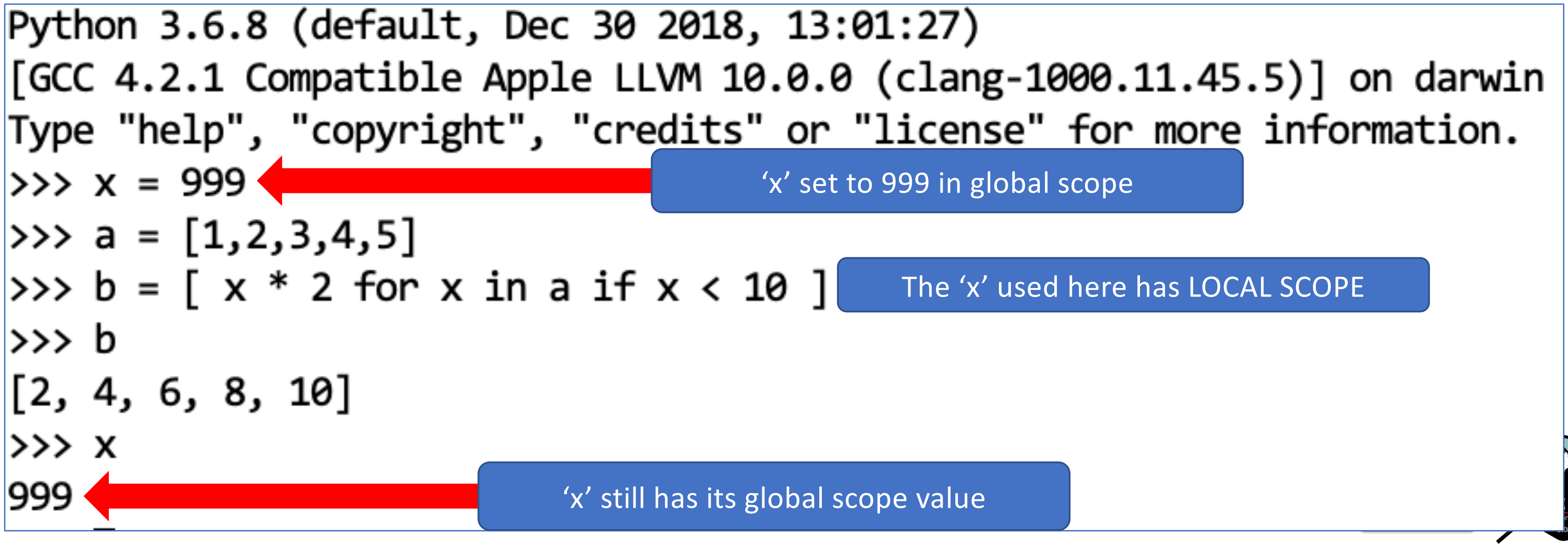


```
Python 2.7.15 (default, Oct  1 2018, 15:59:56)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 999
>>> a = [1,2,3,4,5]
>>> b = [x * 2 for x in a if x < 10]
>>> b
[2, 4, 6, 8, 10]
>>> x
5
```

Set variable ‘x’ to value 999
We use ‘x’ within the list comprehension
‘x’ retains the last value of the loop being executed in the comprehension

Variable Scope in Python3 List Comprehension

- Python3 fixed a bug (feature?) such that the temporary variable used in a list comprehension only has ***local*** scope.
  - When ‘**x**’ is not the same variable as ‘**x**’!



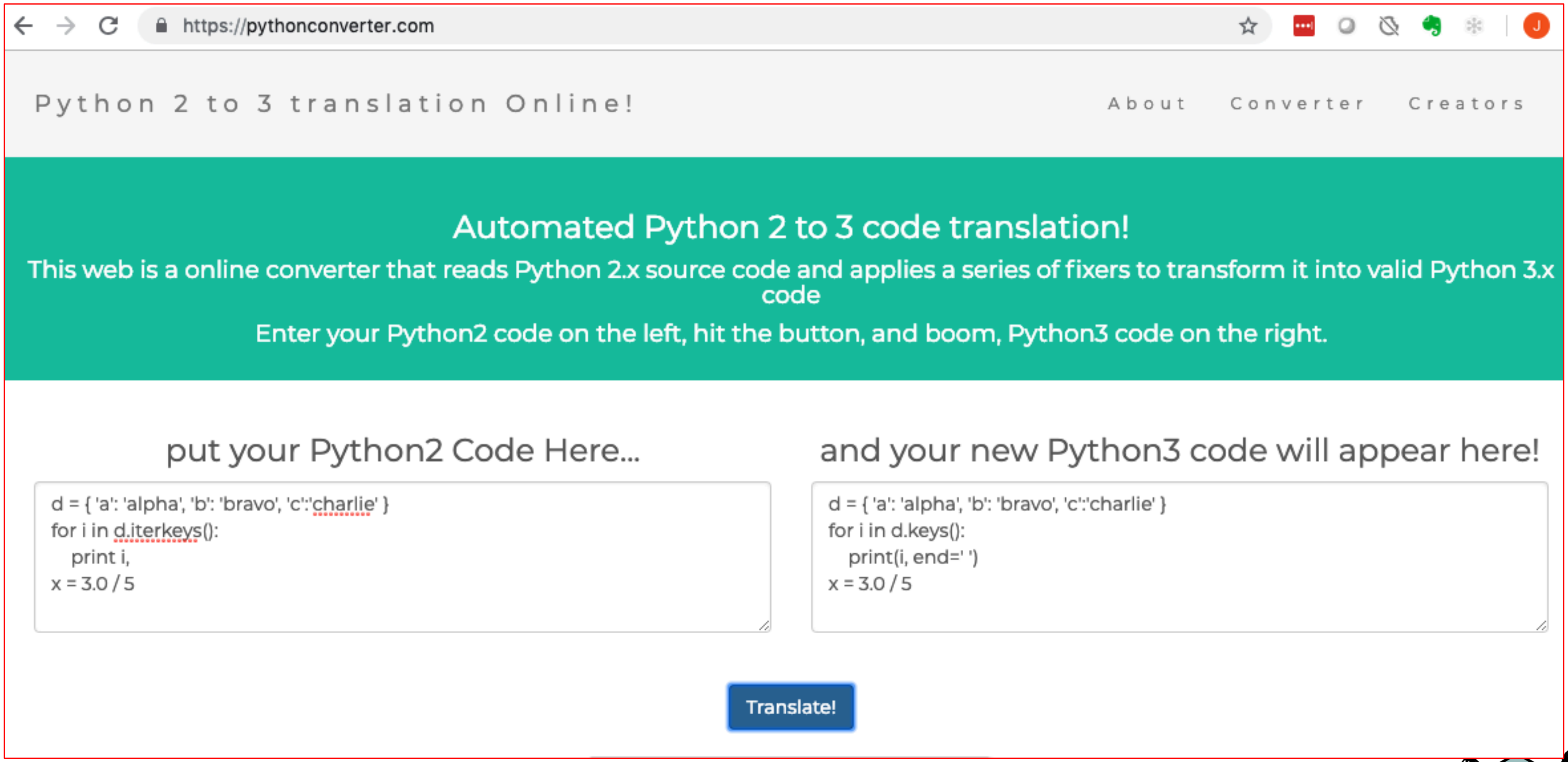
```
Python 3.6.8 (default, Dec 30 2018, 13:01:27)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 999
>>> a = [1,2,3,4,5]
>>> b = [ x * 2 for x in a if x < 10 ]
>>> b
[2, 4, 6, 8, 10]
>>> x
999
```

‘x’ set to 999 in global scope
The ‘x’ used here has LOCAL SCOPE
‘x’ still has its global scope value

Helpful Resources

- https://www.sans.org/course/automating-information-security-with-python

- https://docs.python.org/2/
- https://docs.python.org/3/
- https://docs.python.org/3/howto/pyporting.html
- https://python-3-for-scientists.readthedocs.io/en/latest/python3_transition_guide.html
- https://wiki.python.org/moin/PortingToPy3k/BilingualQuickRef



https://pythonconverter.com

Automated Python 2 to 3 code translation!

This web is a online converter that reads Python 2.x source code and applies a series of fixers to transform it into valid Python 3.x code

Enter your Python2 code on the left, hit the button, and boom, Python3 code on the right.

put your Python2 Code Here...

```
d = { 'a': 'alpha', 'b': 'bravo', 'c':'charlie' }
for i in d.iterkeys():
    print i,
x = 3.0 / 5
```

and your new Python3 code will appear here!

```
d = { 'a': 'alpha', 'b': 'bravo', 'c':'charlie' }
for i in d.keys():
    print(i, end=' ')
x = 3.0 / 5
```

Questions?

- Joff Thyer
  - Penetration Tester, Researcher, Developer at Black Hills Information Security
  - SANS Certified Instructor
  - SEC573: Automating Information Security with Python
  - Security Weekly Co-Host
  - Twitters: @joff_thyer

  https://www.sans.org/course/automating-information-security-with-python