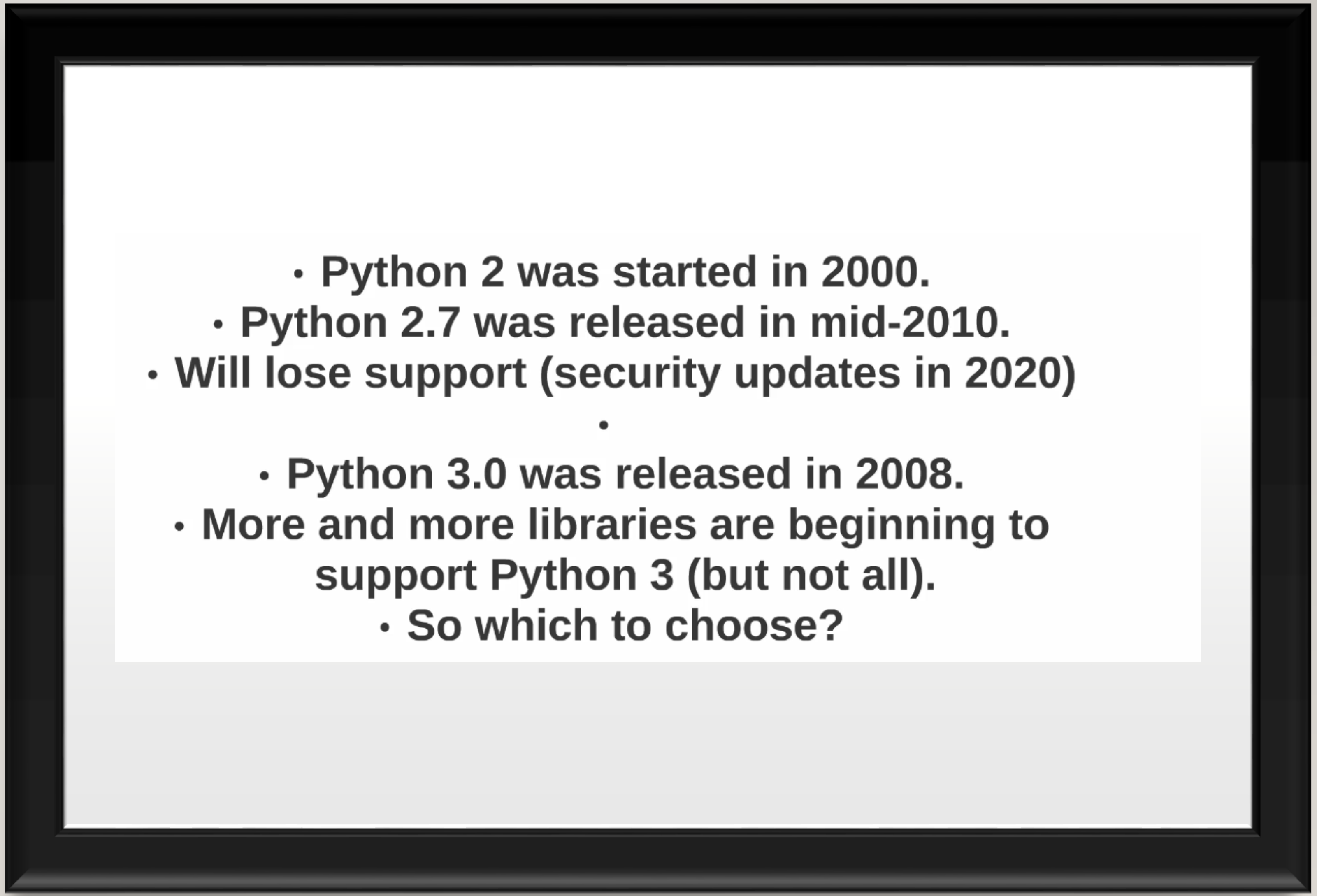
TechnoStrom

PYTHON 2 VS PYTHON 3

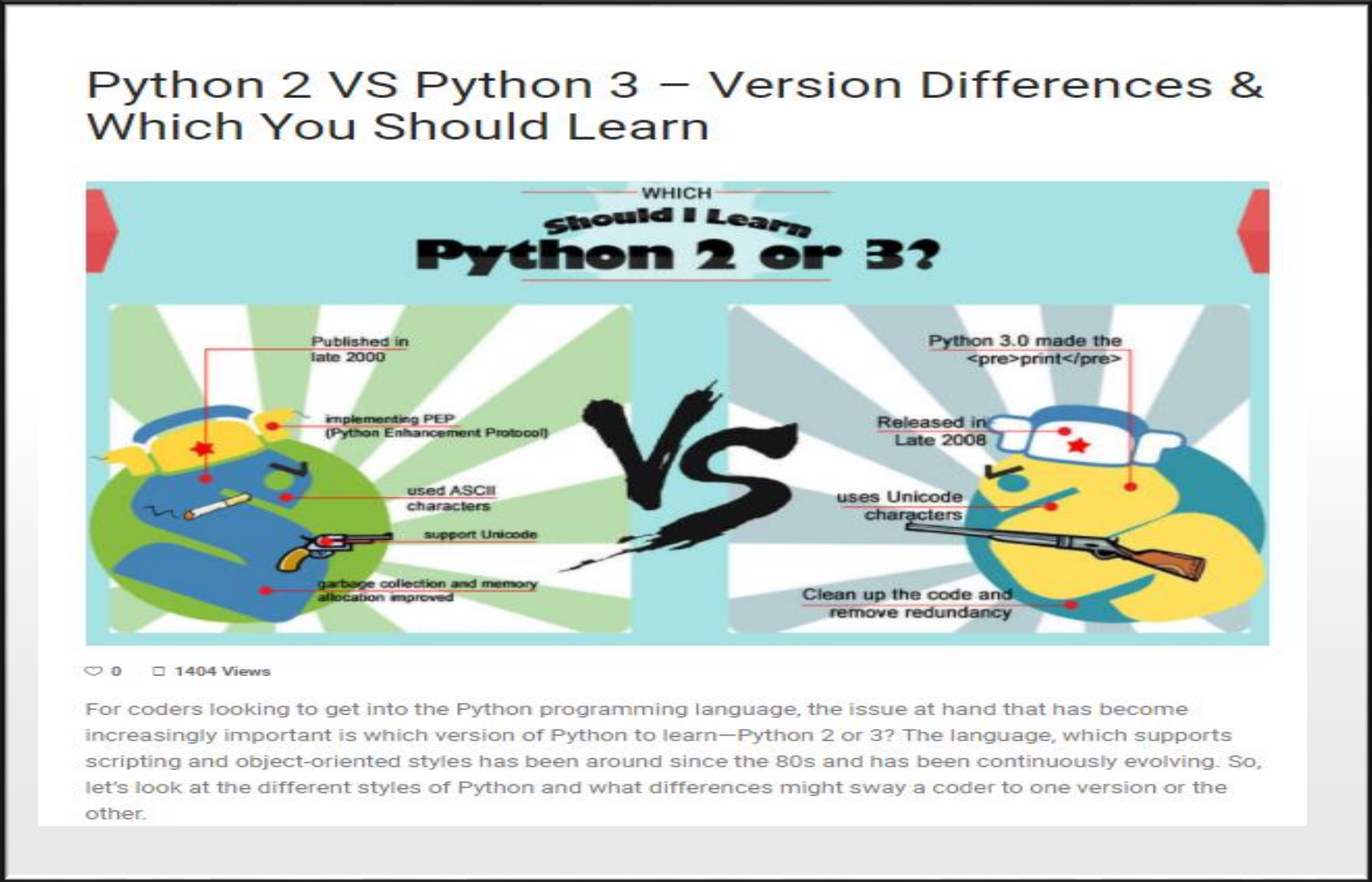
RAVINDRA KUDACHE :- 9730551007

PYTHON 2 VS PYTHON 3

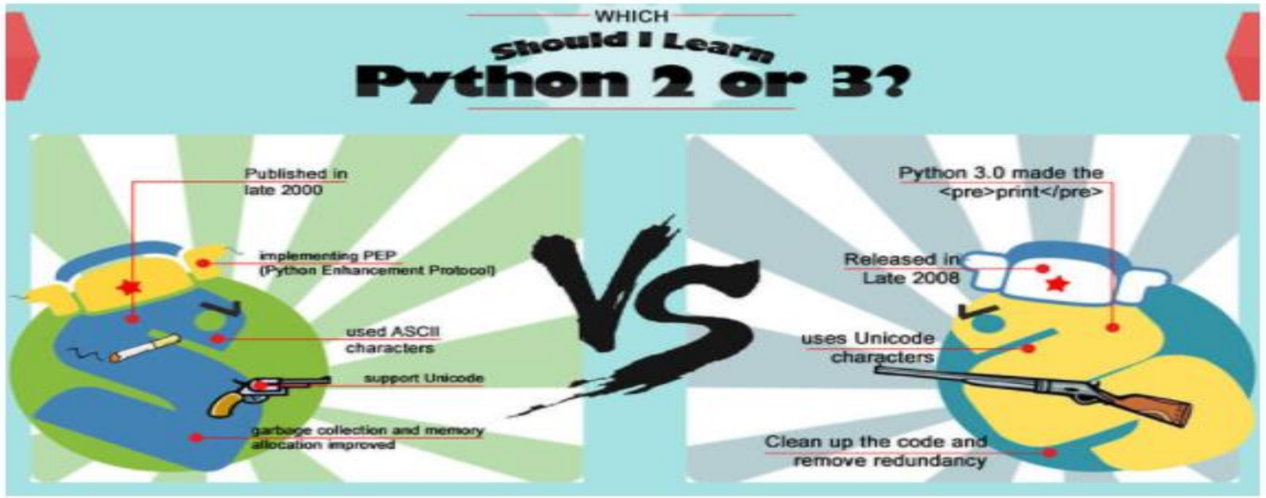


- Python 2 was started in 2000.
- Python 2.7 was released in mid-2010.
- Will lose support (security updates in 2020)
-
- Python 3.0 was released in 2008.
- More and more libraries are beginning to support Python 3 (but not all).
- So which to choose?

PYTHON 2 VS PYTHON 3



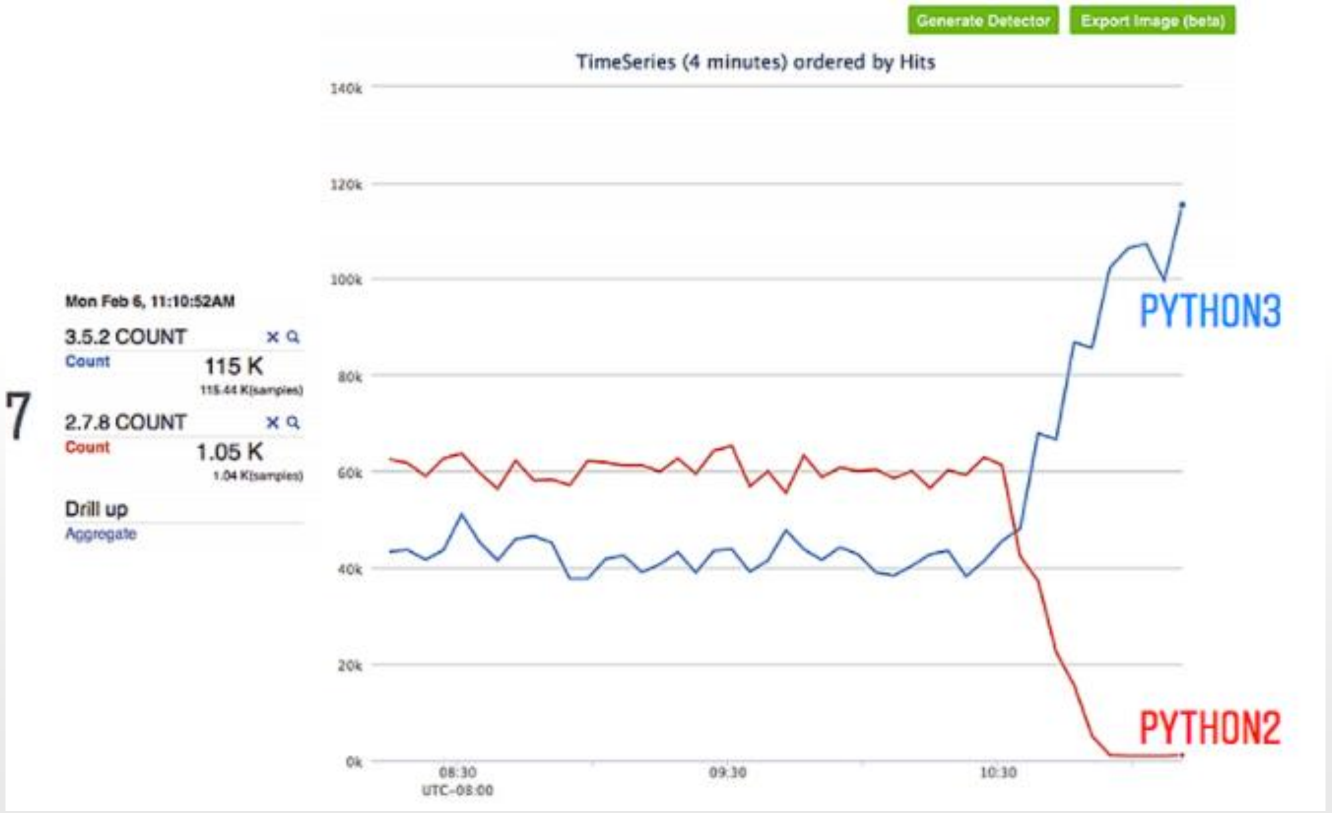
Python 2 VS Python 3 – Version Differences & Which You Should Learn


WHICH
Should I Learn
Python 2 or 3?
Published in late 2000
implementing PEP (Python Enhancement Protocol)
used ASCII characters
support Unicode
garbage collection and memory allocation improved
VS
Python 3.0 made the <pre>print</pre>
Released in Late 2008
uses Unicode characters
Clean up the code and remove redundancy

♡ 0 ☐ 1404 Views

For coders looking to get into the Python programming language, the issue at hand that has become increasingly important is which version of Python to learn—Python 2 or 3? The language, which supports scripting and object-oriented styles has been around since the 80s and has been continuously evolving. So, let's look at the different styles of Python and what differences might sway a coder to one version or the other.

PYTHON 2 VS PYTHON 3

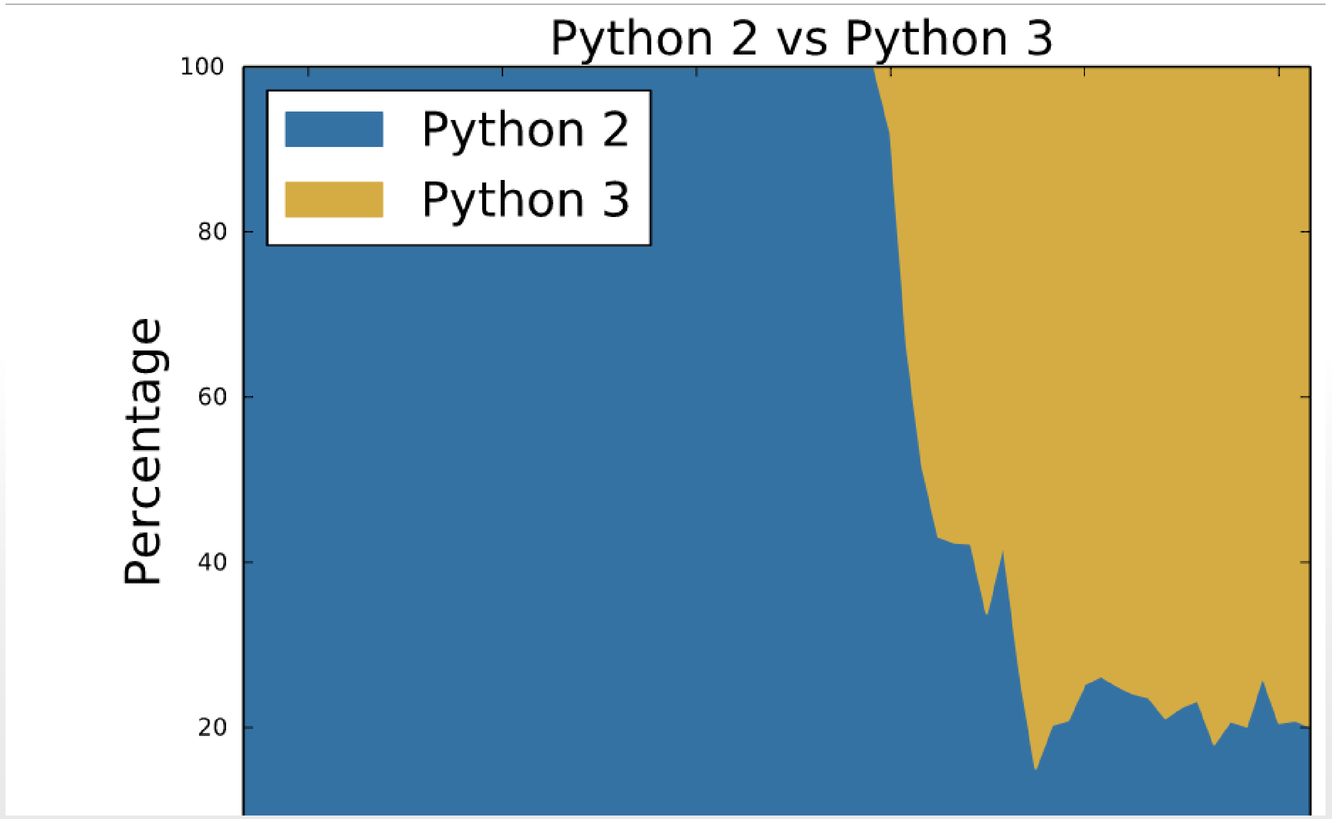


Mon Feb 6, 11:10:52AM

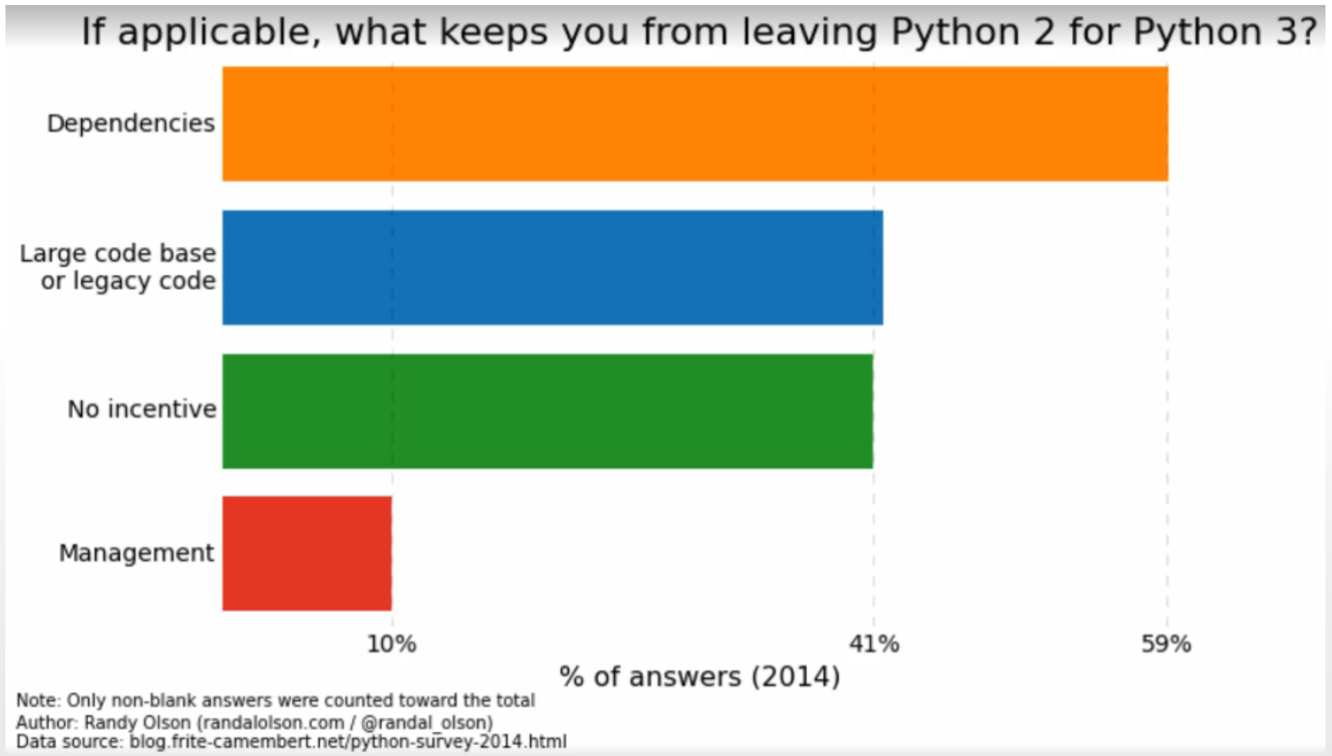
3.5.2 COUNT
Count 115 K
115.44 K(samples)

2.7.8 COUNT
Count 1.05 K
1.04 K(samples)

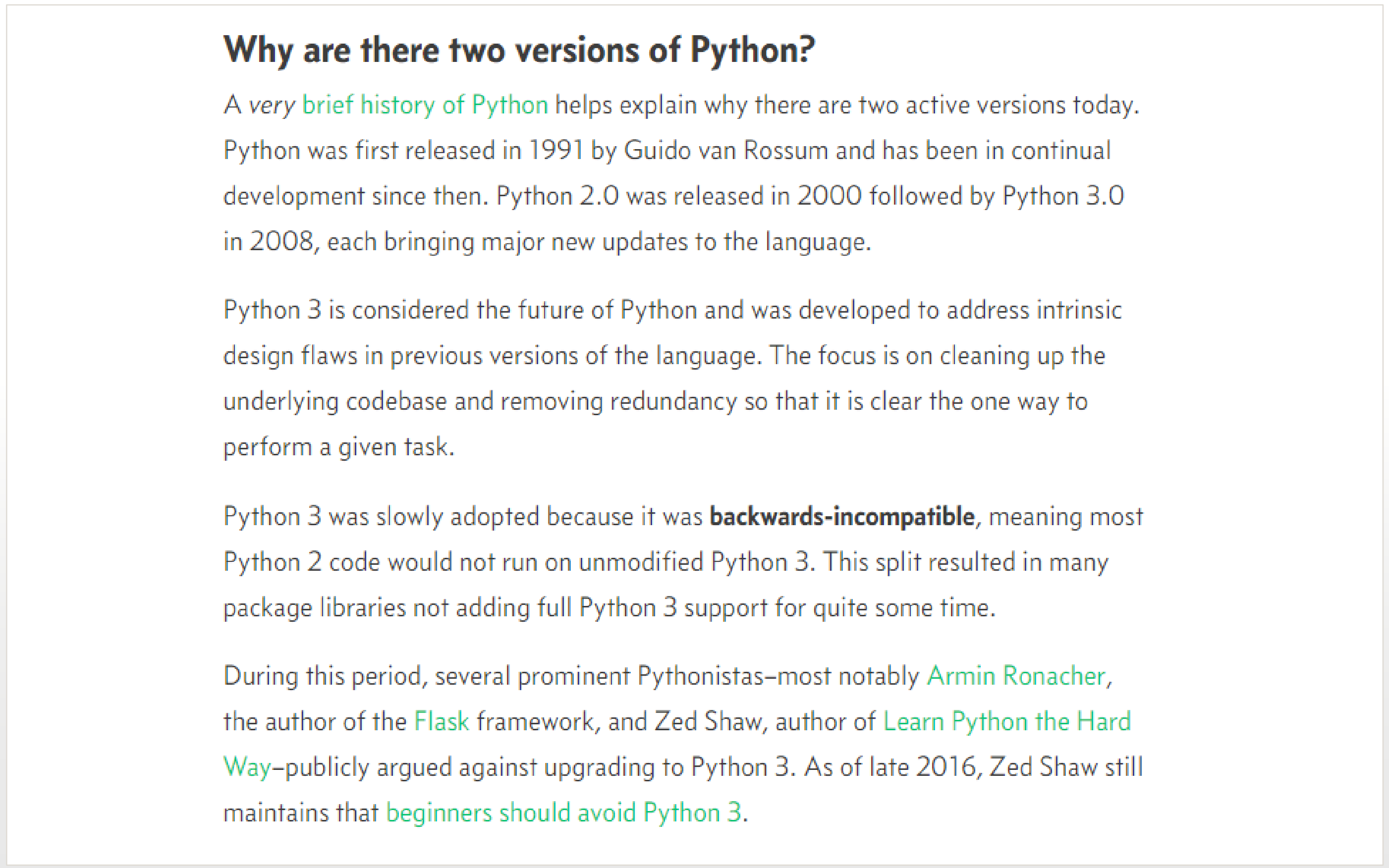
PYTHON 2 VS PYTHON 3



PYTHON 2 VS PYTHON 3



PYTHON 2 VS PYTHON 3



Why are there two versions of Python?

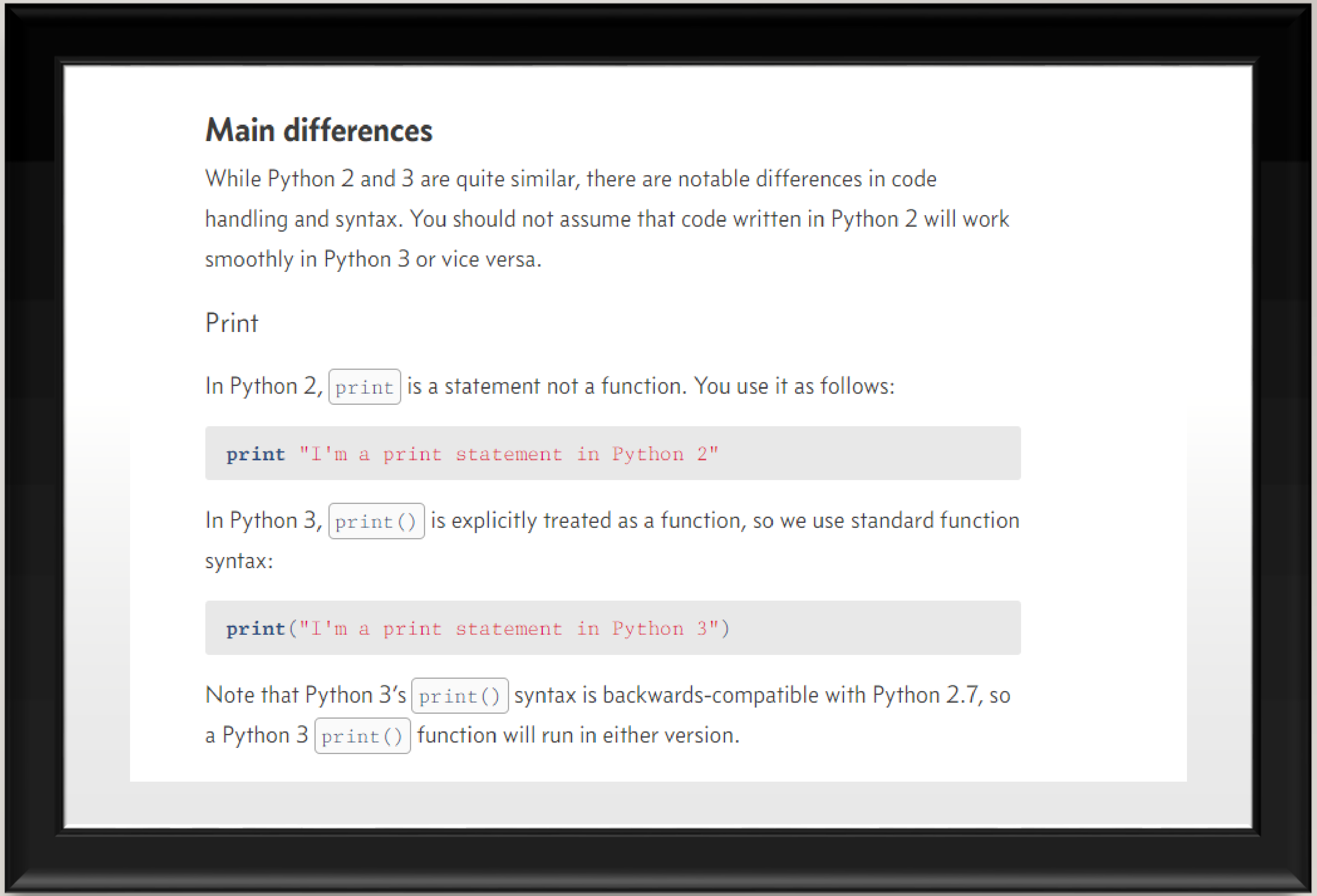
A *very* brief history of Python helps explain why there are two active versions today. Python was first released in 1991 by Guido van Rossum and has been in continual development since then. Python 2.0 was released in 2000 followed by Python 3.0 in 2008, each bringing major new updates to the language.

Python 3 is considered the future of Python and was developed to address intrinsic design flaws in previous versions of the language. The focus is on cleaning up the underlying codebase and removing redundancy so that it is clear the one way to perform a given task.

Python 3 was slowly adopted because it was **backwards-incompatible**, meaning most Python 2 code would not run on unmodified Python 3. This split resulted in many package libraries not adding full Python 3 support for quite some time.

During this period, several prominent Pythonistas–most notably Armin Ronacher, the author of the Flask framework, and Zed Shaw, author of Learn Python the Hard Way–publicly argued against upgrading to Python 3. As of late 2016, Zed Shaw still maintains that beginners should avoid Python 3.

PYTHON 2 VS PYTHON 3



Main differences

While Python 2 and 3 are quite similar, there are notable differences in code handling and syntax. You should not assume that code written in Python 2 will work smoothly in Python 3 or vice versa.

Print

In Python 2, `print` is a statement not a function. You use it as follows:

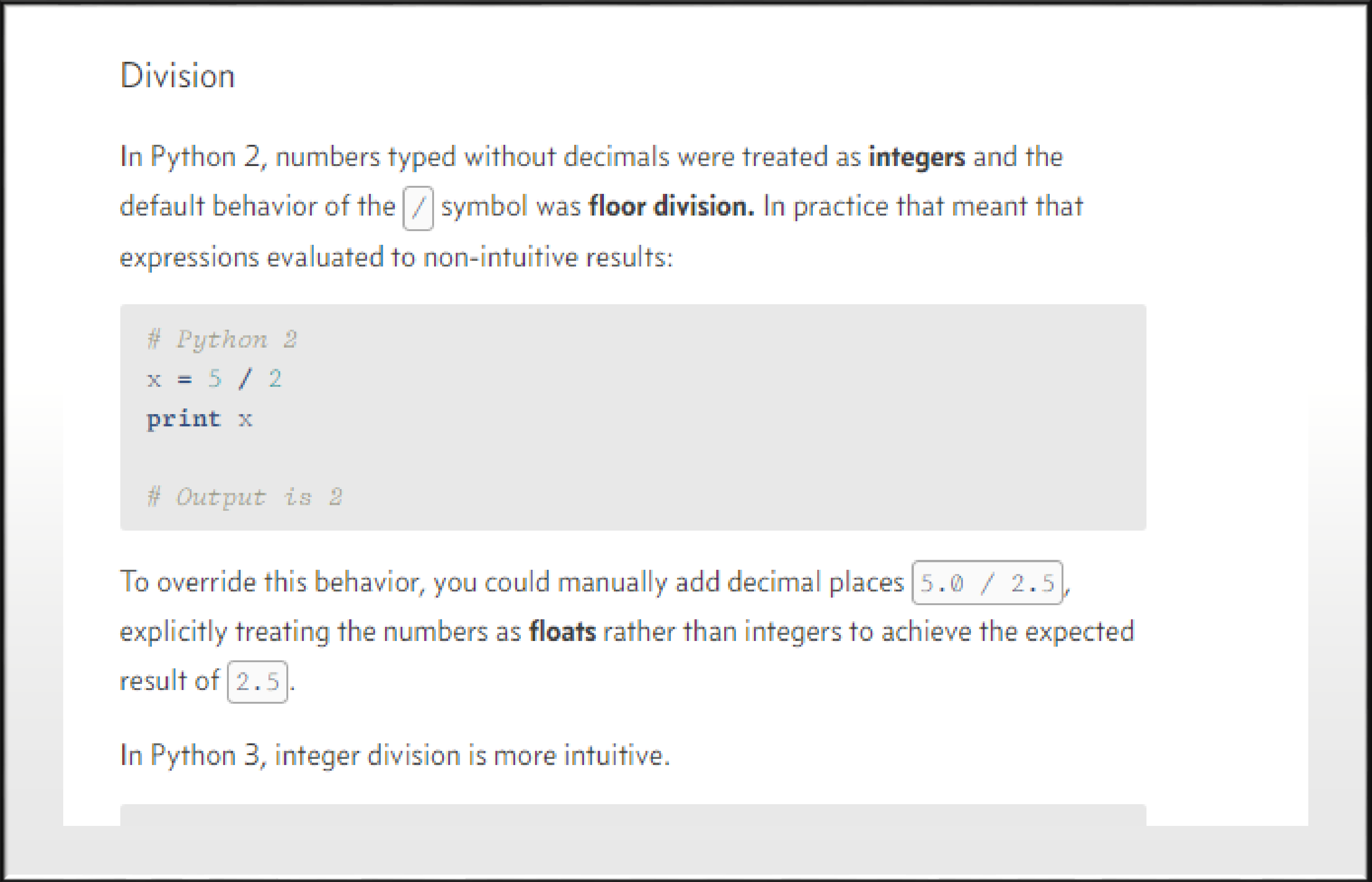
```
print "I'm a print statement in Python 2"
```

In Python 3, `print()` is explicitly treated as a function, so we use standard function syntax:

```
print("I'm a print statement in Python 3")
```

Note that Python 3's `print()` syntax is backwards-compatible with Python 2.7, so a Python 3 `print()` function will run in either version.

PYTHON 2 VS PYTHON 3



Division

In Python 2, numbers typed without decimals were treated as **integers** and the default behavior of the `/` symbol was **floor division.** In practice that meant that expressions evaluated to non-intuitive results:

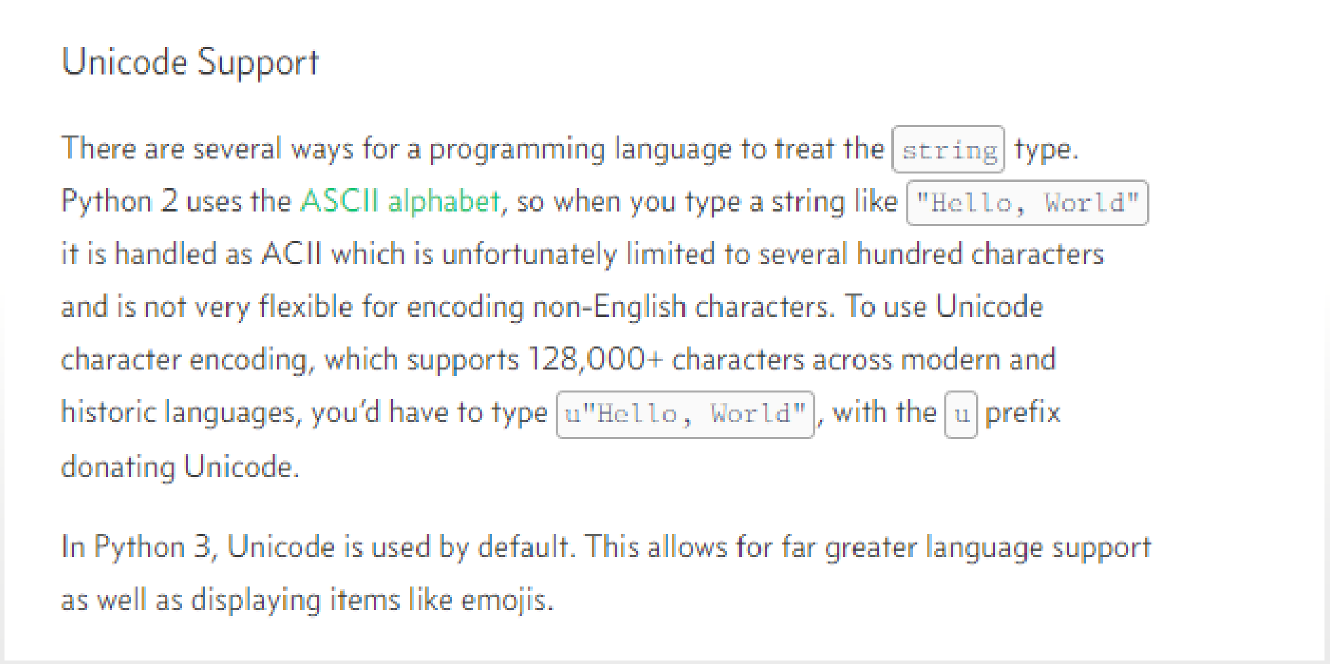
```
# Python 2
x = 5 / 2
print x

# Output is 2
```

To override this behavior, you could manually add decimal places `5.0 / 2.5`, explicitly treating the numbers as **floats** rather than integers to achieve the expected result of `2.5`.

In Python 3, integer division is more intuitive.

PYTHON 2 VS PYTHON 3

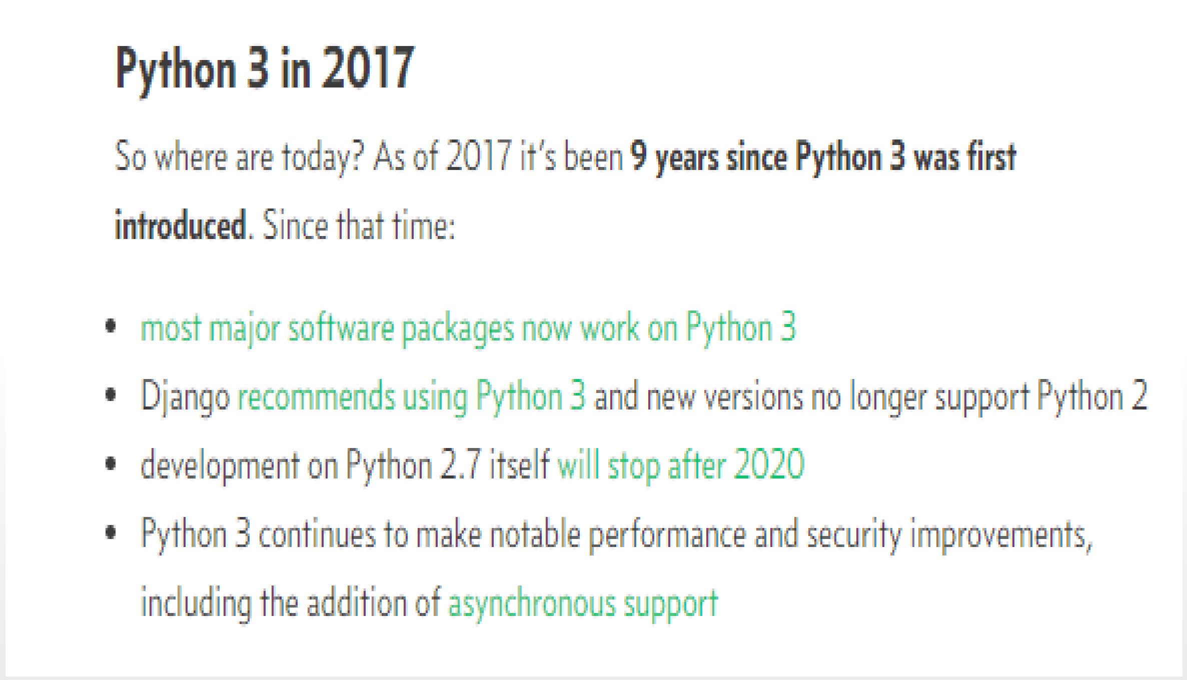


Unicode Support

There are several ways for a programming language to treat the `string` type. Python 2 uses the ASCII alphabet, so when you type a string like `"Hello, World"` it is handled as ACII which is unfortunately limited to several hundred characters and is not very flexible for encoding non-English characters. To use Unicode character encoding, which supports 128,000+ characters across modern and historic languages, you'd have to type `u"Hello, World"`, with the `u` prefix donating Unicode.

In Python 3, Unicode is used by default. This allows for far greater language support as well as displaying items like emojis.

PYTHON 2 VS PYTHON 3



Python 3 in 2017

So where are today? As of 2017 it's been **9 years since Python 3 was first introduced**. Since that time:

- most major software packages now work on Python 3
- Django recommends using Python 3 and new versions no longer support Python 2
- development on Python 2.7 itself will stop after 2020
- Python 3 continues to make notable performance and security improvements, including the addition of asynchronous support

PYTHON 2 VS PYTHON 3

We will See you on Next Session

Thank You