The evolution of dinosaurs:

Migrating from Python 2 to Python 3

Hello!

I am Marta Gómez

IT Security Developer at Wazuh.

@Mrs_DarkDonado // @mgmacias95

Agenda

- Python 2.7 is almost dead.
- Compatible code with Python 2 & 3.
- What's new in Python 3?
- Integrating Python in your project.

1

Python 2.7 is almost dead

Python 2.7 will be no longer maintained in 2020, which means there’s only one more year of Python 2.7 support.

Being the last of the 2.x series, **2.7** will have an extended period of maintenance. Specifically, 2.7 will receive bugfix support until **January 1, 2020**. After the last release, 2.7 will receive no support.

PEP 373

This is the moment

But, how to do it?

2

Compatible code with Python 2 and 3

A good start to port a Python 2 project to Python 3 is to make code compatible with both. You can start doing that today!

How to do it

- Drop support for Python 2.6 and older.
- Have a good test coverage.
- Update your code.
- Check your dependencies are compatible with Python 3 (caniusepython3).
- Use CI to stay compatible.

Drop support for Python < 2.6

- Python 2.6 is no longer freely supported.
- There are lots of Python 2.7 and 3 syntax that isn't included in Python 2.6. For example:

```
{x: x**2 for x in range(10)}
```

- Required changes will still look like idiomatic Python code.

Update your code

- Use Futurize or Modernize to port the code.
- Manual changes required: depends on the chosen tool.
- Division: Operators / and //.
- Text versus binary data.
- Use feature detection instead of version detection.

Division

Python 3

```
>>> 5 / 2
2.5
>>> 5 // 2
2
```

Python 2

```
>>> 5 / 2
2
>>> 5.0 / 2.0
2.5
>>> from __future__ import division
>>> 5 / 2
2.5
>>> 5 // 2
2
```

Text vs binary

Python 3

```
>>> 'hi' == b'hi'
False
>>> bytes(2)
b'\x00\x00'
>>> b'123'[1]
50
>>> import hashlib
>>> h = hashlib.md5()
>>> h.update('hi')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Unicode-objects must be encoded before hashing
>>> h.update(b'hi')
```

Python 2

```
>>> 'hi' == b'hi'
True
>>> bytes(2)
'2'
>>> b'123'[1]
'2'
>>> import hashlib
>>> h = hashlib.md5()
>>> h.update('hi')
```

Feature detection vs Version detection

Version detection

```
>>> import sys
>>> if sys.version_info[0] > 2:
...     from queue import Queue
... else:
...     from Queue import Queue
```

Feature detection

```
>>> try:
...     from queue import Queue
... except ImportError:
...     from Queue import Queue
```

More differences

Python 3

```
>>> filter(lambda x: x % 2, range(10))
<filter object at 0x7f8f308625f8>
>>> map(lambda x: x * 2, range(10))
<map object at 0x7f6ba9fa25f8>
>>> {'a': 1, 'b': 2}.keys()
dict_keys(['a', 'b'])
```

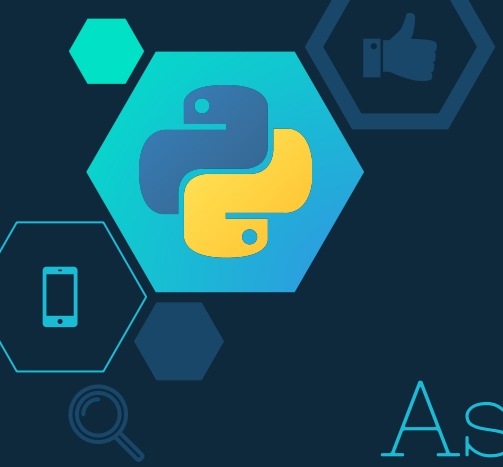
Python 2

```
>>> filter(lambda x: x % 2, range(10))
[1, 3, 5, 7, 9]
>>> map(lambda x: x * 2, range(10))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> {'a': 1, 'b': 2}.keys()
['a', 'b']
```

3

What's new in Python 3?

You're missing lots of new interesting features only available in Python 3 when you make compatible code with both versions.



Asyncio

Library added in Python 3.4, highly improved in Python 3.7, to write concurrent code using async/await syntax.

Good fit for applications using lots of IO or network protocols.

Type Hints

Added in Python 3.5.

Helps IDEs, type checkers and makes code easier to read.

```
>>> from typing import List
>>> def example(parameter: str) -> List:
...     return [parameter]
...
>>> example('a')
```

Yield from

A shortened version of `for item in iterable: yield item`

But it also allows subgenerators to receive sent and thrown values directly from the calling scope, and return a final value to the outer generator:

```
>>> def accumulate():
...     tally = 0
...     while True:
...         next = yield
...         if next is None:
...             return tally
...         tally += next
...
>>> def gather_tallies(tallies):
...     while True:
...         tally = yield from accumulate()
...         tallies.append(tally)
...
```

```
>>> tallies = []
>>> acc = gather_tallies(tallies)
>>> next(acc)
>>> for i in range(4):
...     acc.send(i)
...
>>> acc.send(None)
>>> for i in range(5):
...     acc.send(i)
...
>>> acc.send(None)
>>> tallies
[6, 10]
```

Modern Dictionaries

```
{'sarah': 'banana',
 'barry': 'orange',
 'rachel': 'pear',
 'tim': 'peach',
 'guido': 'apple'}
```

Version	Dict Size	Dict Ordering	Notes
Python 2.7	280	`['sarah', 'barry', 'rachel', 'tim', 'guido']`	Scrambled
Python 3.5	196	`dict_keys(['guido', 'sarah', 'barry', 'tim', 'rachel'])`	Randomized
Python 3.6	112	`dict_keys(['guido', 'sarah', 'barry', 'rachel', 'tim'])`	Ordered

4

Integrating Python in your project

Integrating python directly in your project allows to develop for a specific python without forcing users to install it.

Possible options

- Create standalone binaries.
- Add a python interpreter in your install directory.

Standalone binaries

- Integrates all the Python application in a single binary.
- End users don’t need a Python interpreter.
- Doesn’t speed up.
- Dynamic compiled python libraries aren’t correctly added to the binary. The interpreter needs to be compiled with specific flags.
- Final binaries are very heavy.
- Freeze, cx_Freeze, pyinstaller, nukita

Add a python interpreter in your install directory

- Compile Python configuring a custom install directory:
  ```
  $ ./configure --prefix=/var/ossec/python
  ```
- The interpreter can be used by multiple scripts.
- Approach used by Splunk.

Thanks!

Any questions?

You can find me at:

- @Mrs_DarkDonado // @mgmacias95
- marta@wazuh.com

Links

- PEP 373 -- Python 2.7 Release Schedule
- Porting Python 2 Code to Python 3
- Modern Python Dictionaries: A confluence of a dozen great ideas
- Freeze
- Python docs.

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival