Python3的新特性和改进

杭州美登科技 杜逸先

目录 CONTENTS

1 Python的现状

Python的现状

Python2.7将于2020年1月1日停止维护

Python的现状

Python2.7将于2020年1月1日停止维护

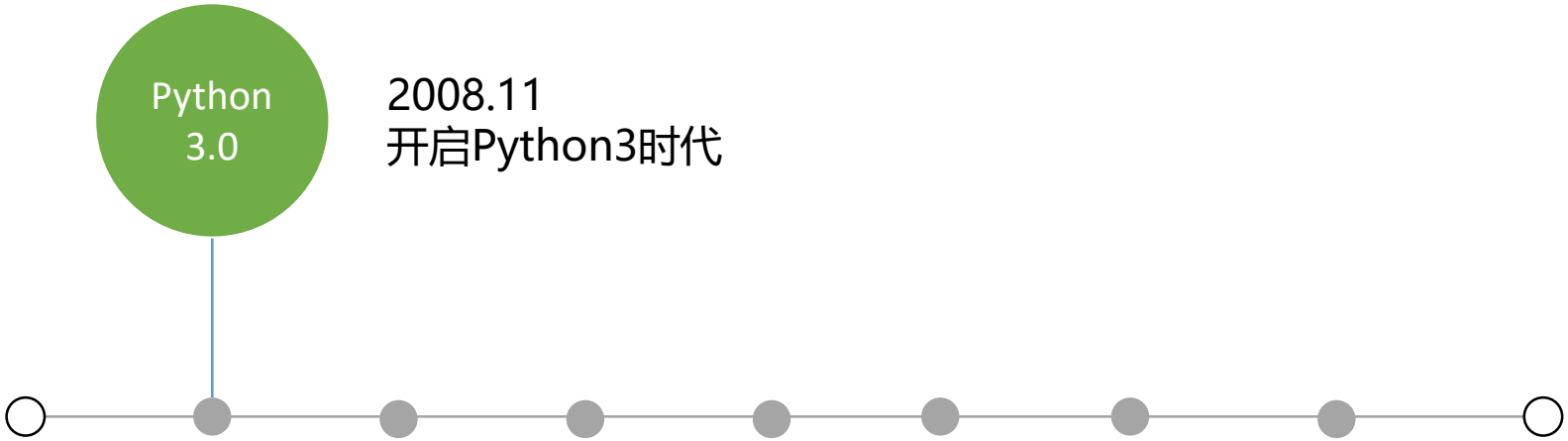
主流Python包陆续终止对Python2的支持

Python的现状

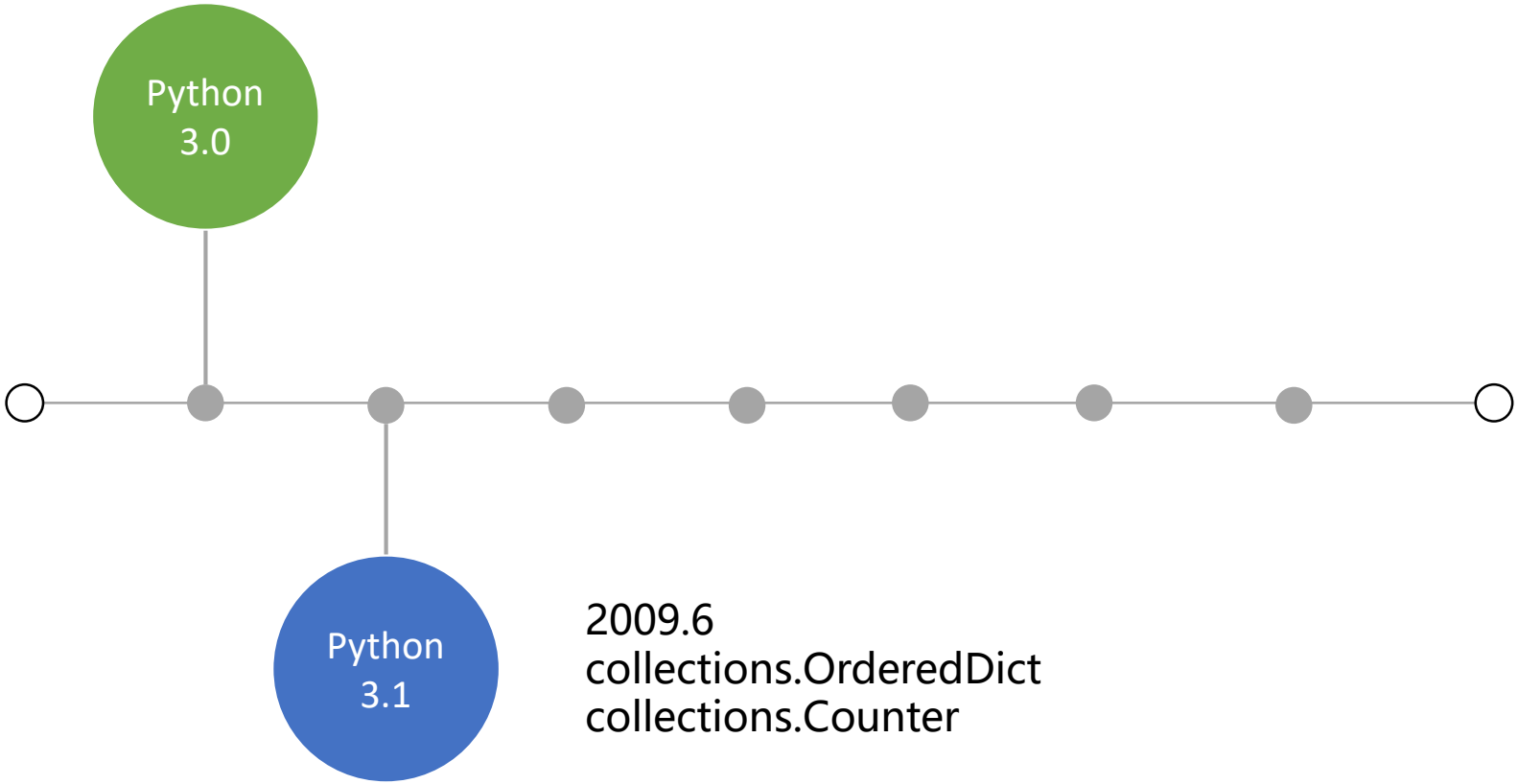
Python2.7将于2020年1月1日停止维护

主流Python包陆续终止对Python2的支持

Python的现状

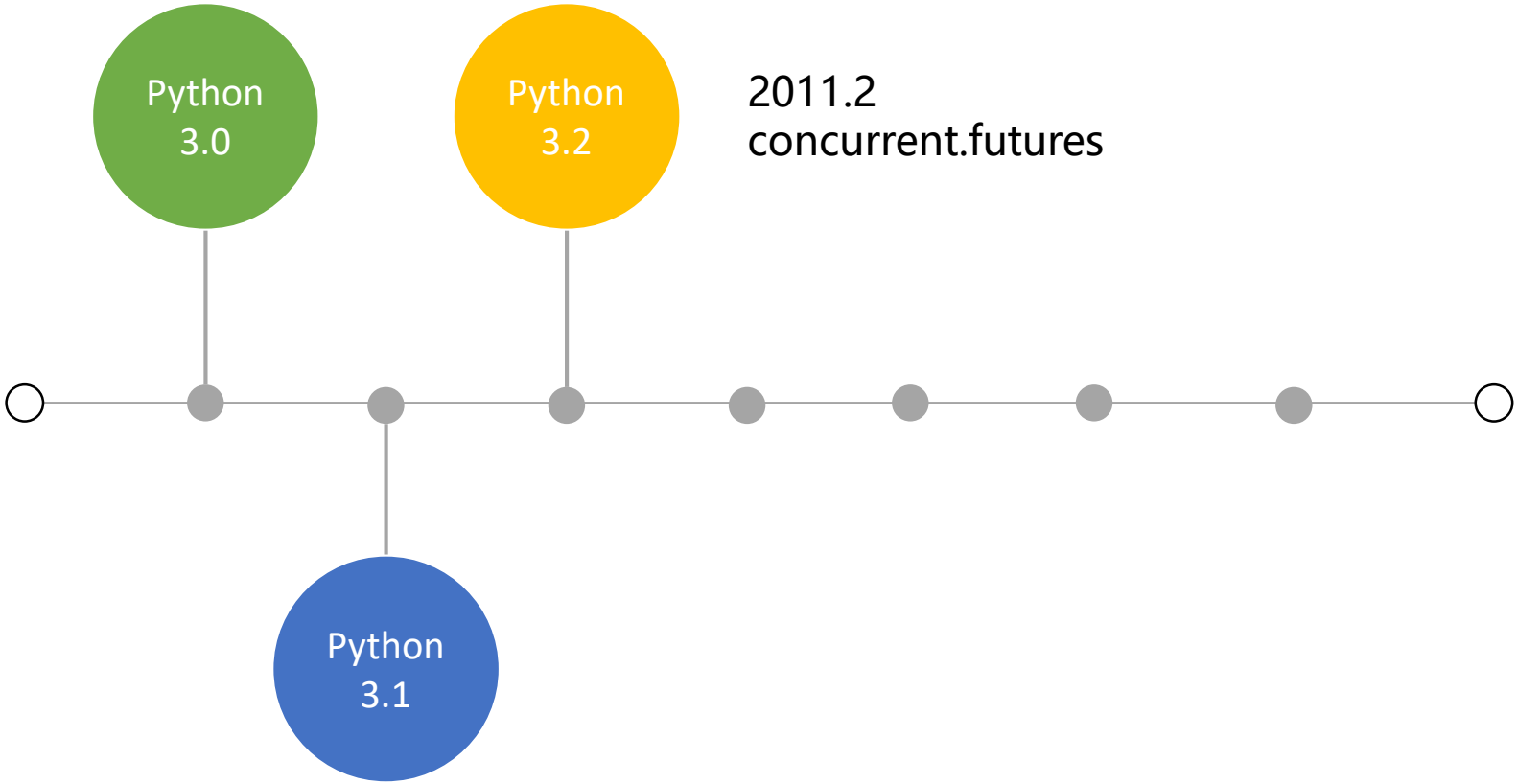


Python的现状


Python 3.0
Python 3.1
2009.6
collections.OrderedDict
collections.Counter

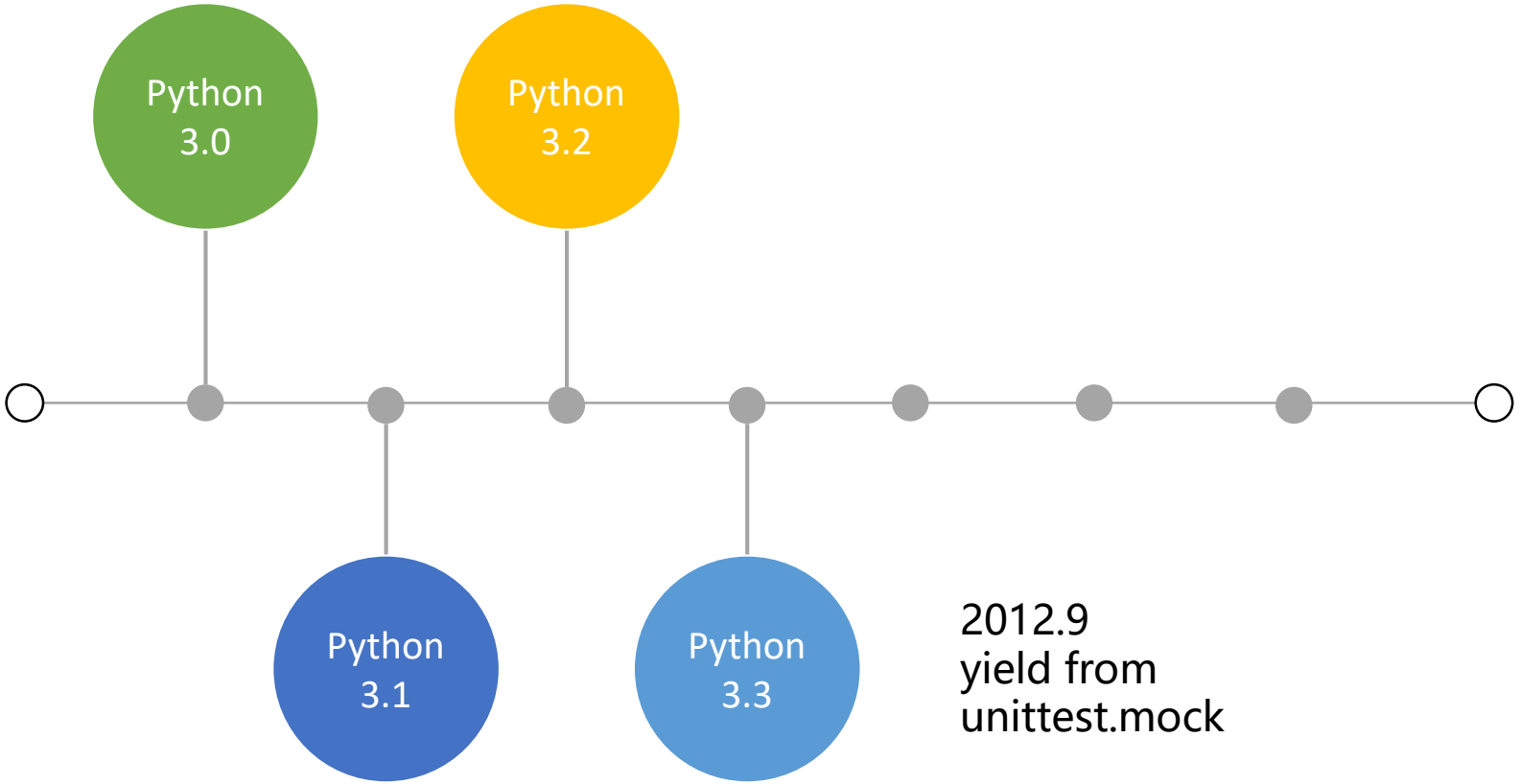

PYTHON 30th

Python的现状


Python 3.0
Python 3.2
2011.2
concurrent.futures
Python 3.1

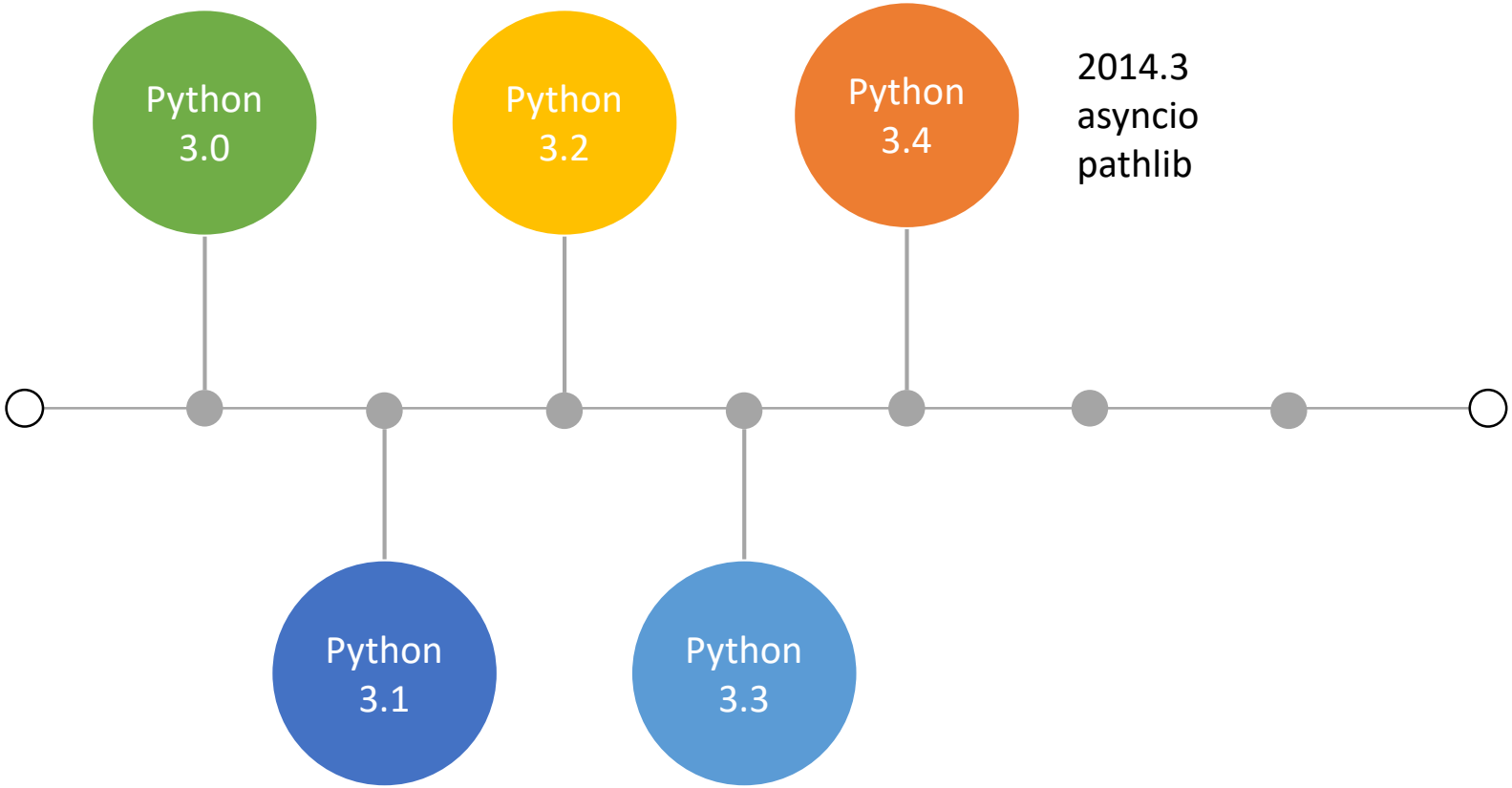

PYTHON 30th

Python的现状


Python 3.0
Python 3.2
Python 3.1
Python 3.3
2012.9
yield from
unittest.mock

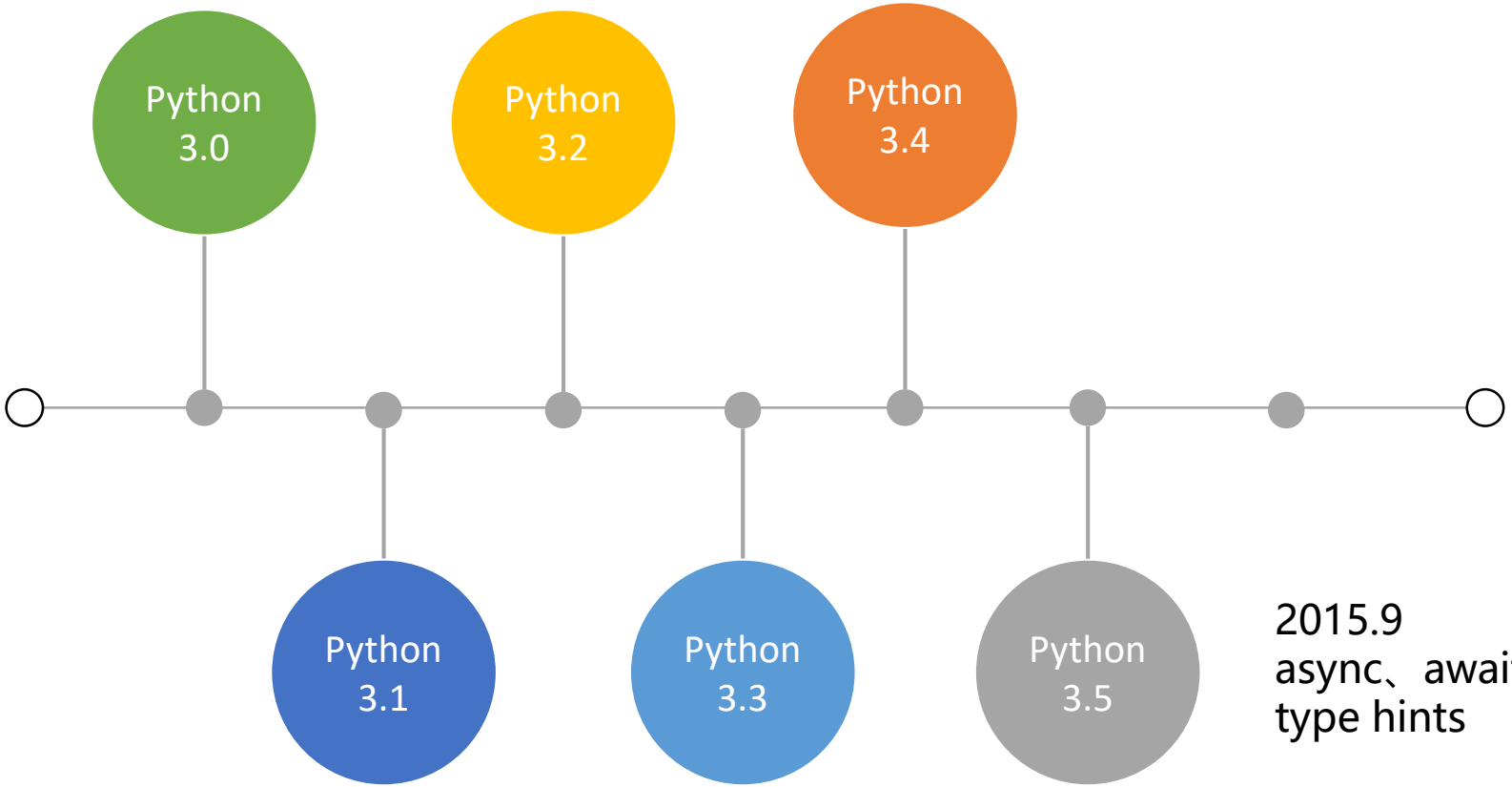

PYTHON 30th

Python的现状


Python 3.0
Python 3.2
Python 3.4
2014.3
asyncio
pathlib
Python 3.1
Python 3.3

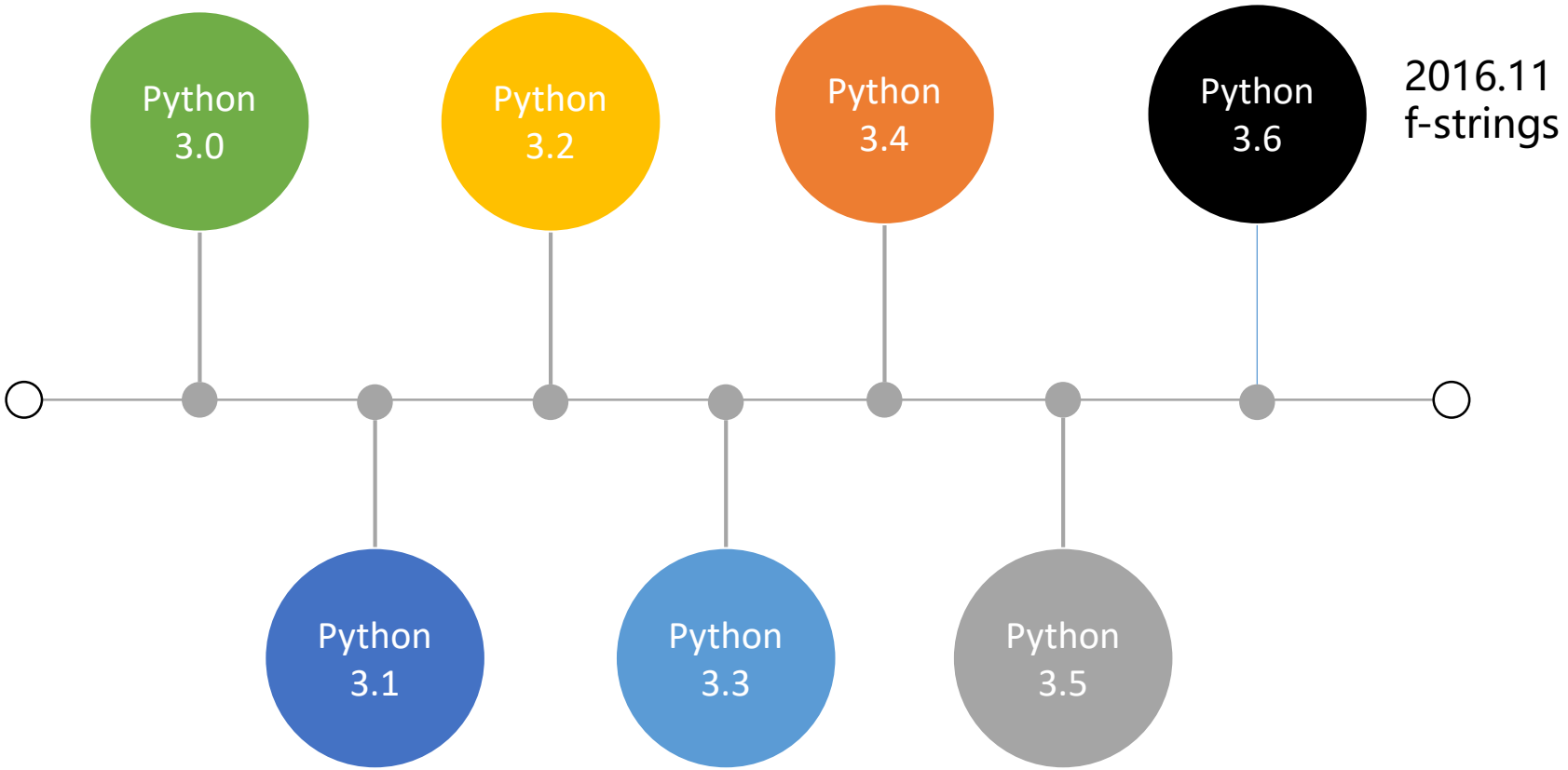

PYTHON 30th

Python的现状


Python 3.0
Python 3.2
Python 3.4
Python 3.1
Python 3.3
Python 3.5
2015.9
async、await
type hints


PYTHON 30th

Python的现状


Python 3.0
Python 3.2
Python 3.4
Python 3.6
2016.11
f-strings
Python 3.1
Python 3.3
Python 3.5


PYTHON 30th

Python的现状

当前版本

Python3.7(published at 2018.6)

dataclasses、breakpoint()

Python3.8(published at 2019-10-14本周一)

assignment expressions、positional-only parameters

2 Python3的新特性和改进
——重要变化

Python3的新特性和改进——重要变化

Text Vs. Data Instead Of Unicode Vs. 8-bit

	内容	类型	混合使用
Python3	Text Vs. Data	str Vs. bytes	✗
Python2	Unicode Vs. 8-bit	unicode Vs. str	✓

Python3的新特性和改进——重要变化

Python2

```
In [1]: a = u'你好'

In [2]: b = 'hello'

In [3]: type(a), type(b)
Out[3]: (unicode, str)

In [4]: unicode.mro(), str.mro()
Out[4]: ([unicode, basestring, object], [str, basestring, object])

In [5]: c = a + b
        print(c)
        你好hello

In [6]: type(c)
Out[6]: unicode
```

Python3的新特性和改进——重要变化

Python3

```
In [1]: a = '你好'

In [2]: b = b'hello'

In [3]: type(a), type(b)
Out[3]: (str, bytes)

In [4]: str.mro(), bytes.mro()
Out[4]: ([str, object], [bytes, object])

In [8]: a + b
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-bd58363a63fc> in <module>
----> 1 a + b

TypeError: must be str, not bytes

In [9]: c = a + b.decode()
        c, type(c)
Out[9]: ('你好hello', str)

In [12]: d = a.encode() + b
         d, type(d), d.decode()
Out[12]: (b'\xe4\xbd\xa0\xe5\xa5\xbdhello', bytes, '你好hello')
```

Python3的新特性和改进——重要变化

Python3的str和bytes不能混用！

Python3的新特性和改进——重要变化

求一段文本的MD5:Python2

```
In [1]: text = 'Welcome to Pycon China 2019, Hanzhou station.'

In [2]: from hashlib import md5

In [3]: md5_hash = md5(text)
        md5_hash.hexdigest()

Out[3]: '0a970d432f7a69fd2bdf0c3fc2559bbd'
```

Python3的新特性和改进——重要变化

求一段文本的MD5:Python3

```
In [1]: text = 'Welcome to Pycon China 2019, Hanzhou station.'

In [2]: from hashlib import md5

In [3]: md5_hash = md5(text)
        md5_hash.hexdigest()

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-bee2ef4a3e9f> in <module>
----> 1 md5_hash = md5(text)
      2 md5_hash.hexdigest()

TypeError: Unicode-objects must be encoded before hashing

In [4]: md5_hash = md5(text.encode())
        md5_hash.hexdigest()

Out[4]: '0a970d432f7a69fd2bdf0c3fc2559bbd'
```

Python3的新特性和改进——重要变化

Views And Iterators Instead Of Lists

	Python2	Python3
dict.keys(),dict.items(), dict.values()	list	view
map,filter	list	generator(iterator)
range	list	range(iterator)*
xrange	xrange(iterator)	/
zip	list	zip(iterator)

Python3的新特性和改进——重要变化

Python2:dict

```
In [1]: a = {'a': 1, 'b': 2, 'c': 3}

In [2]: keys = a.keys()
        keys
Out[2]: ['a', 'c', 'b']

In [3]: a['d'] = 4

In [4]: keys
Out[4]: ['a', 'c', 'b']

In [5]: a.keys()
Out[5]: ['a', 'c', 'b', 'd']
```

Python3的新特性和改进——重要变化

Python3:dict

```
In [1]: a = {'a': 1, 'b': 2, 'c': 3}

In [2]: keys = a.keys()
        keys
Out[2]: dict_keys(['a', 'b', 'c'])

In [3]: a['d'] = 4

In [4]: keys
Out[4]: dict_keys(['a', 'b', 'c', 'd'])
```

Python3的新特性和改进——重要变化

Python2:range, map, zip

```
In  [1]: range(5)
Out[1]: [0, 1, 2, 3, 4]

In  [2]: map(str.upper, 'abcde')
Out[2]: ['A', 'B', 'C', 'D', 'E']

In  [3]: zip(range(5), 'abcde')
Out[3]: [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

Python3的新特性和改进——重要变化

Python3:range, map, zip

```
In  [1]: range(5)
Out[1]: range(0, 5)

In  [2]: map(str.upper, 'abcde')
Out[2]: <map at 0x7f2c76a401d0>

In  [3]: zip(range(5), 'abcde')
Out[3]: <zip at 0x7f2c76a94308>

In  [4]: list(range(5))
Out[4]: [0, 1, 2, 3, 4]

In  [5]: list(map(str.upper, 'abcde'))
Out[5]: ['A', 'B', 'C', 'D', 'E']

In  [6]: list(zip(range(5), 'abcde'))
Out[6]: [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

Python3的新特性和改进——重要变化

Python3:range, map, zip

```
In [1]: m = map(str.upper, 'abcde')

In [2]: list(m)
Out[2]: ['A', 'B', 'C', 'D', 'E']

In [3]: list(m)
Out[3]: []

In [4]: l = range(5)

In [5]: list(l)
Out[5]: [0, 1, 2, 3, 4]

In [6]: list(l)
Out[6]: [0, 1, 2, 3, 4]
```

Python3的新特性和改进——重要变化

使用迭代器的优势：惰性计算、节约内存、无限长度

```
In [1]: from itertools import combinations

In [2]: c = combinations('ABCDEF', 3)

In [3]: next(c)
Out[3]: ('A', 'B', 'C')

In [4]: next(c)
Out[4]: ('A', 'B', 'D')

In [5]: next(c)
Out[5]: ('A', 'B', 'E')
```

Python3的新特性和改进——重要变化

使用迭代器的优势：惰性计算、节约内存、无限长度

```
In  [1]: from sys import getsizeof

In  [2]: getsizeof(range(100000))
Out[2]: 800072
```

Python2

```
In  [1]: from sys import getsizeof

In  [2]: getsizeof(range(100000))
Out[2]: 48
```

Python3

Python3的新特性和改进——重要变化

使用迭代器的优势：惰性计算、节约内存、无限长度

```
In [1]: from itertools import count

In [2]: c = count()

In [3]: next(c)
Out[3]: 0

In [4]: for _ in range(10000):
            next(c)

In [5]: next(c)
Out[5]: 10001
```

Python3的新特性和改进——重要变化

Exception in Python2

In [2]:
```
try:
    raise Exception, 'Error'
except Exception, error:
    print error
```

```
Error
```

Python3的新特性和改进——重要变化

Exception in Python3

1. 所有自定义异常类要继承于BaseException
2. rasie 1个异常对象
3. except … as …

In [1]:
```
class MyException(BaseException):
    def __init__(self, code, msg):
        self.code = code
        self.msg = msg
    def __str__(self):
        return f'MyException(code={self.code}, msg={self.msg})'
```

In [2]:
```
try:
    raise MyException(2, 'Error Message.')
except MyException as error:
    print(error)
```

```
MyException(code=2, msg=Error Message.)
```

Python3的新特性和改进——重要变化

Python2: int and long

```
In  [1]: type(0xFFFFFFFF)
Out[1]: int

In  [2]: type(0xFFFFFFFFFFFFFFFF)
Out[2]: long

In  [3]: type(123L)
Out[3]: long
```

Python3的新特性和改进——重要变化

Python3: int

```
In  [1]: type(0xFFFFFFFFFFFFFFFF)
Out[1]: int

In  [2]: 123L
  File "<ipython-input-2-540d80edec42>", line 1
    123L
       ^
SyntaxError: invalid syntax
```

Python3的新特性和改进——重要变化

/ in Python2

```
In  [1]: 1 / 2
Out[1]: 0

In  [2]: 1 / 2.0
Out[2]: 0.5
```

Python3的新特性和改进——重要变化

Python3 区分 truediv(/)与floordiv(//)

```
In  [1]: 1 /  2
Out[1]: 0.5

In  [2]: 1 // 2
Out[2]: 0

In  [3]: 1 // 2.0
Out[3]: 0.0
```

Python3的新特性和改进——重要变化

Python2的比较操作不够严谨

```
In  [1]: None > None
Out[1]: False

In  [2]: None < None
Out[2]: False

In  [3]: 1 > 'a'
Out[3]: False
```

Python3的新特性和改进——重要变化

Python3无法比较未定义对应操作的类型

```
In [1]: None < None
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-0ac550fa0b68> in <module>
----> 1 None < None

TypeError: '<' not supported between instances of 'NoneType' and 'NoneType'

In [2]: 1 < 'a'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-4751c80d99ec> in <module>
----> 1 1 < 'a'

TypeError: '<' not supported between instances of 'int' and 'str'
```

Python3的新特性和改进——重要变化

Python2 round()四舍五入

```
In  [1]: round(1.5)
Out[1]: 2.0

In  [2]: round(2.2)
Out[2]: 2.0

In  [3]: round(2.5)
Out[3]: 3.0
```

Python3的新特性和改进——重要变化

Python3 round()四舍六入五成双

```
In  [1]: round(1.5)
Out[1]: 2

In  [2]: round(2.2)
Out[2]: 2

In  [3]: round(2.5)
Out[3]: 2
```

Python3的新特性和改进——重要变化

Python2 元类的使用

In [2]:
```
from abc import ABCMeta, abstractmethod
```

In [3]:
```
class ICommand(object):
    __metaclass__ = ABCMeta

    @abstractmethod
    def run(self):
        pass
```

Python3的新特性和改进——重要变化

Python3 元类的使用

In [1]:
```
from abc import ABCMeta, abstractmethod
```

In [2]:
```
class ICommand(metaclass=ABCMeta):

    @abstractmethod
    def run(self):
        pass
```

2 Python3的新特性和改进
——易用性的提升

Python3的新特性和改进——易用性的提升

print is a statement in Python2

```
In [1]: a, b, c, d = range(4)

In [2]: print a, '->', b, '->', c, '->', d, '#'
0 -> 1 -> 2 -> 3 #
```

Python3的新特性和改进——易用性的提升

print is a function in Python3

```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

Python3的新特性和改进——易用性的提升

print is a function in Python3

```
In [1]: a, b, c, d = range(4)

In [2]: print(a, b, c, d, sep='->', end='#')
0->1->2->3#
```

Python3的新特性和改进——易用性的提升

print can be replaced and mocked in Python3

In [1]:
```
import builtins

oldprint = print

def myprint(*args):
    oldprint(*args, sep='\n', end='\n******\n')

builtins.print = myprint
```

In [2]:
```
print(1,2,3,4)
```

```
1
2
3
4
******
```

Python3的新特性和改进——易用性的提升

print can be replaced and mocked in Python3

```
In [1]: from unittest import mock

In [2]: with mock.patch('builtins.print') as mock_print:
            print('Test')
        mock_print.assert_called_with('Test')
```

Python3的新特性和改进——易用性的提升

super() in Python2

```
In [1]:
class A(object):
    def __init__(self, a):
        self.a = a

In [2]:
class B(A):
    def __init__(self, a, b):
        super(B, self).__init__(a)
        self.b = b

In [3]:
b = B(1,2)
b.a, b.b

Out[3]:
(1, 2)
```

Python3的新特性和改进——易用性的提升

super() in Python3

```
In [3]:
class A:
    def __init__(self, a):
        self.a = a

In [4]:
class B(A):
    def __init__(self, a, b):
        super().__init__(a)
        self.b = b

In [5]:
b = B(1,2)
b.a, b.b

Out[5]:
(1, 2)
```

Python3的新特性和改进——易用性的提升

f-strings

```
In  [1]: a = 1
         b = 'hello'
         f'{a} {b} {2*3}'
Out[1]: '1 hello 6'

In  [2]: f'{90:x}'
Out[2]: '5a'

In  [4]: f'{1/3:10.4}'
Out[4]: '    0.3333'

In  [5]: f'{1/3:010.4}'
Out[5]: '00000.3333'
```

Python3的新特性和改进——易用性的提升

Iterable unpacking

```
In [1]: a, b, *rest = range(10)
        a, b, rest
Out[1]: (0, 1, [2, 3, 4, 5, 6, 7, 8, 9])

In [2]: a, *_, b = 'hello'
        a, b
Out[2]: ('h', 'o')

In [3]: *_, last = 'abcdefg'
        last
Out[3]: 'g'

In [4]: a = [1,2,3]
        [4, 5, *a]
Out[4]: [4, 5, 1, 2, 3]

In [5]: a = 1
        b = 'hello'
        f'{a} {b} {2*3}'
Out[5]: '1 hello 6'

In [6]: a = {'a':1, 'b':2, 'c':3}
        {'d':4, **a}
Out[6]: {'d': 4, 'a': 1, 'b': 2, 'c': 3}
```

Python3的新特性和改进——易用性的提升

Assignment expressions（Python 3.8）

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

Python3的新特性和改进——易用性的提升

标准库：concurrent.futures

concurrent.futures

In [1]:
```
import requests
```

In [2]:
```
urls = [f'https://news.cnblogs.com/n/page/{n}' for n in range(1,100)]
```

In [3]:
```
def get_url(url):
    return requests.Session().get(url).status_code
```

In [4]:
```
get_url(urls[0])
```

Out[4]: 200

In [5]:
```
%timeit [get_url(url) for url in urls]
```

8.71 s ± 2.67 s per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [6]:
```
from concurrent.futures import ThreadPoolExecutor
```

In [7]:
```
executor = ThreadPoolExecutor()
```

In [8]:
```
%timeit list(executor.map(get_url, urls))
```

1.4 s ± 198 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Python3的新特性和改进——易用性的提升

标准库：pathlib

```
In [1]: from pathlib import Path

In [2]: path = Path('./')

In [3]: path
Out[3]: PosixPath('.')

In [4]: path.absolute()
Out[4]: PosixPath('/home/nbuser/library')

In [5]: path.is_dir()
Out[5]: True

In [6]: file = path / 'a.py'

In [7]: file.exists()
Out[7]: True

In [8]: with open(file) as f:
            print(f.readlines())
['print(\'Hello, world\')']
```

Python3的新特性和改进——易用性的提升

标准库：dataclasses

```
In [1]: from dataclasses import dataclass

In [2]: @dataclass
        class Person:
            name: str
            age: int
            female: str
            address: str

In [3]: mike = Person('Mike', 20, 'M', 'Hangzhou')

In [4]: mike
Out[4]: Person(name='Mike', age=20, female='M', address='Hangzhou')

In [5]: print(mike)
        Person(name='Mike', age=20, female='M', address='Hangzhou')
```

2 Python3的新特性和改进
——新的理念和编程方法

Python3的新特性和改进——新的理念和编程方法

async、await与asyncio

Asyncio

In [1]:
```
import asyncio
```

In [2]:
```
async def a():
    print('Coroutine A waitting for 2 seconds.')
    await asyncio.sleep(2)
    print('Coroutine A is executed.')
    return 'a'
```

In [3]:
```
async def b():
    print('Coroutine B waitting for 1 seconds.')
    await asyncio.sleep(1)
    print('Coroutine B is executed.')
    return 'b'
```

In [4]:
```
async def c():
    print('Coroutine C is executed immediately.')
    await asyncio.sleep(0)
    return 'c'
```

In [5]:
```
import time
start = time.time()
results = await asyncio.gather(a(),b(),c())
print(results)
end = time.time()
print(end - start)
```

```
Coroutine A waitting for 2 seconds.
Coroutine B waitting for 1 seconds.
Coroutine C is executed immediately.
Coroutine B is executed.
Coroutine A is executed.
['a', 'b', 'c']
2.0029242038726807
```

Python3的新特性和改进——新的理念和编程方法

async、await与asyncio

In [1]:
```
import asyncio
import aiohttp
import time
```

In [2]:
```
async def download_site(session, url):
    async with session.get(url) as response:
        return response.status
```

In [3]:
```
async def download_all_sites(sites):
    tasks = []
    async with aiohttp.ClientSession() as session:
        for url in sites:
            task = asyncio.ensure_future(download_site(session, url))
            tasks.append(task)
        await asyncio.gather(*tasks, return_exceptions=True)
    return tasks
```

In [10]:
```
urls = [f'https://www.baidu.com/s?wd=python&pn={n}0' for n in range(0,100)]
```

In [15]:
```
start = time.time()
tasks  = await download_all_sites(urls)
end = time.time()
end - start
```

Out[15]:
```
2.7349588871002197
```

In [17]:
```
tasks[0]
```

Out[17]:
```
<Task finished coro=<download_site() done, defined at <ipython-input-2-db2a903890
69>:1> result=200>
```

Python3的新特性和改进——新的理念和编程方法

async、await与asyncio

In [10]:
```
urls = [f'https://www.baidu.com/s?wd=python&pn={n}0' for n in range(0,100)]
```

In [15]:
```
start = time.time()
tasks  = await download_all_sites(urls)
end = time.time()
end - start
```

Out[15]: 2.7349588871002197

In [17]:
```
tasks[0]
```

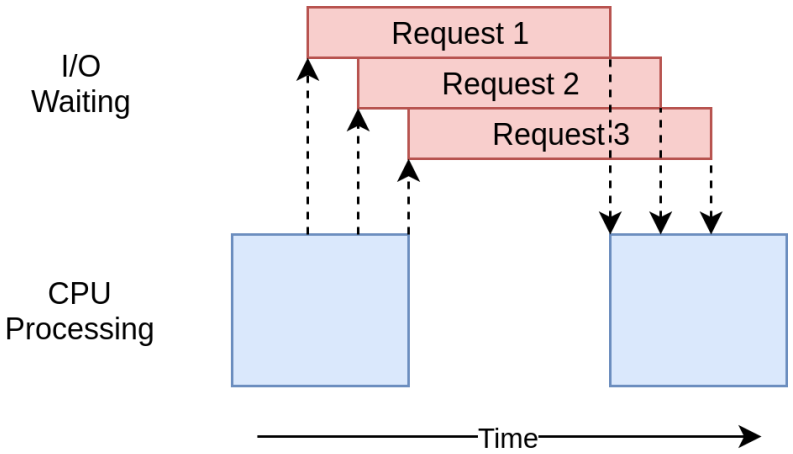
Out[17]: <Task finished coro=<download_site() done, defined at <ipython-input-2-db2a903890
69>:1> result=200>

In [14]:
```
import requests

def get_url(url):
    return requests.get(url).status_code

from concurrent.futures import ThreadPoolExecutor

executor = ThreadPoolExecutor()

start = time.time()
results = list(executor.map(get_url, urls))
end = time.time()
end - start
```

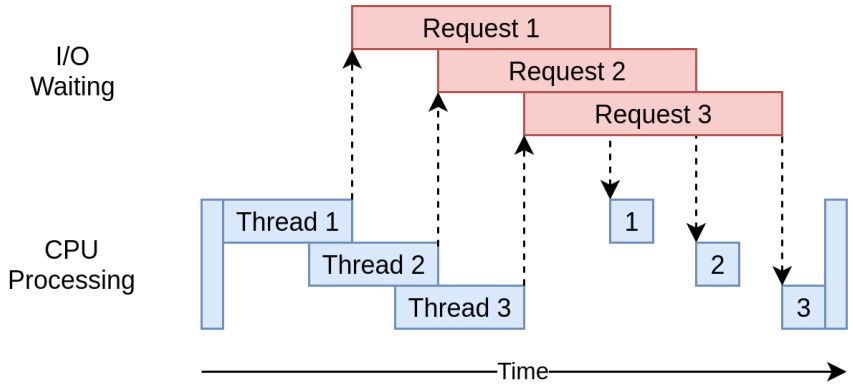
Out[14]: 15.053833246231079

Python3的新特性和改进——新的理念和编程方法

async、await与asyncio



asyncio version



threading version

Python3的新特性和改进——新的理念和编程方法

type hints

type hints

```
In [1]:
def add(a:int, b:int)->int:
    return a + b

In [2]:
add.__annotations__

Out[2]:
{'a': int, 'b': int, 'return': int}

In [3]:
from inspect import signature

In [4]:
signature(add)

Out[4]:
<Signature (a: int, b: int) -> int>

In [5]:
from typing import get_type_hints

In [6]:
get_type_hints(add)

Out[6]:
{'a': int, 'b': int, 'return': int}
```

Python3的新特性和改进——新的理念和编程方法

type hints:代码即文档

type_hints.py

type_hints.py > Python > max_length

```
1  from typing import List
2
3  def max_length(words: List[str]) → int:
4      return max(len(word) for word in words)
```

Python3的新特性和改进——新的理念和编程方法

type hints:使用mypy进行静态类型检查

static_type_check.py

```python
a: int
a = 10
print(a.upper())

def add(a:int, b:str) -> int:
    return a + b

```

```
(share) ➜ share mypy static_type_check.py
static_type_check.py:3: error: "int" has no attribute "upper"
static_type_check.py:6: error: Unsupported operand types for + ("int" and "str")
Found 2 errors in 1 file (checked 1 source file)
```

Python3的新特性和改进——新的理念和编程方法

type hints:更好地利用IDE

app.py

app.py > Python > filter_words

```
1  from typing import List
2
3  def filter_words(words: List[str]) → str:
4      ret = []
5      for word in words:
6          if word.
```

capitalize
casefold
center
count
encode
endswith
expandtabs
find
format
format_map
index
isalnum

str.capitalize() → str

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

2 Python3的新特性和改进
——其他变化

Python3的新特性和改进——其他变化

Python3使用绝对引入(absolute import)*

Python3使用input()代替raw_input()

Python3使用next()函数代替.next方法

Python3增加了很多关键字：True、False、None、with、as、async、await

Python3使用!=代替<>

Python3修改了一些标准库的组织方式和命名：

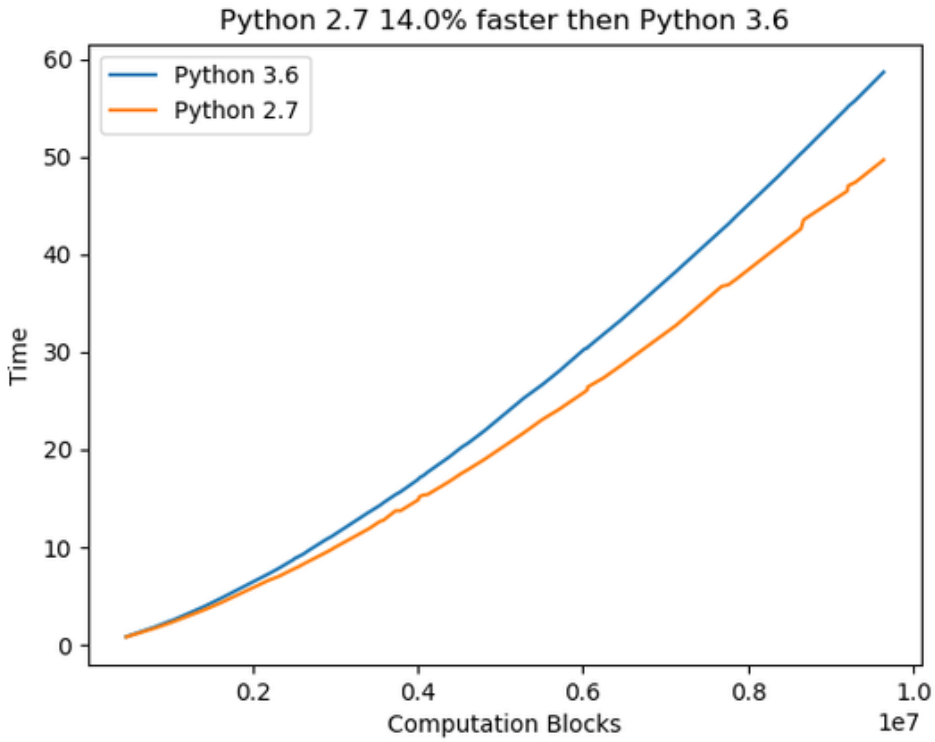
- Queue->queue
- ConfigParser->configparser
- urllib, urllib2, urlparse->urllib

2 Python3的新特性和改进

——性能提升

Python3的新特性和改进——性能提升

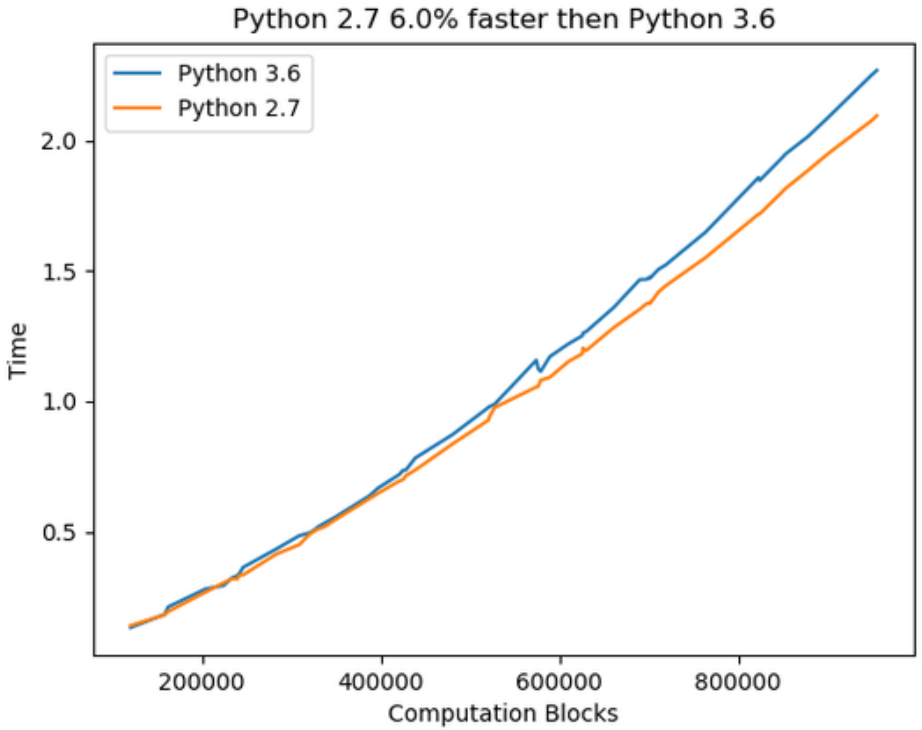
鉴于一些原因（例如Python3不区分int和long）,python3的性能比python2持平（或略差）



计算任务（机器学习等）

Python3的新特性和改进——性能提升

鉴于一些原因（例如Python3不区分int和long）,python3的性能比python2持平（或略差）



日常任务（Web应用、桌面应用）

Python3的新特性和改进——性能提升

Python3在持续的改进

Python3.6使用了新的dict实现方式，节约了大量的内存。

```
➜ ~ python2 -c "from sys import getsizeof;print getsizeof({x: x**2 for x in range(100000)})"
6291736
➜ ~ python3 -c "from sys import getsizeof;print(getsizeof({x: x**2 for x in range(100000)}))"
5242976
```

Python3的新特性和改进——性能提升

Python3在持续的改进

预计在Python3.9(dev阶段)中实现的子解释器（PEP 554），将解放GIL对CPU密集型Python多线程程序的限制。

Python3的新特性和改进——性能提升

鉴于大量的流行包使用C拓展进行加速，Python项目的开发速度和代码质量是更值得关注的指标。

3 迁移到Python3

迁移到Python3

Porting Python 2 Code to Python 3

只需关注Python2.7

保证好的测试覆盖(利用 coverage.py工具)

使用Modernize等工具更新代码

使用Pylint保证代码质量

检查和替换不支持Python3的依赖

使用持续集成保证代码在不同版本下的正常表现

考虑使用静态类型检查

迁移到Python3

Instagram的Python3迁移案例（10亿月活 Python + Django）

节约了 12% 的整体 CPU 使用率（Django/uwsgi）

节约了 30% 的内存使用（celery）

在单个接口中利用 asynio 平行的去做多件事情，降低了 20-30% 的请求延迟

4 问答环节

THANK YOU