Python best practices, tools and libs

Lviv 09/09/2017

Taras Rudnyk

I am …

- **Taras Rudnyk**
- Software Engineer at SoftServe
- From Kyiv, Ukraine
- Follow up: @TarasRudnyk,
  https://github.com/TarasRudnyk

Broken windows theory

As result you have

- Try to write good code (don’t broke window)
- If you see some problems in code fix it immediately (fix broken window as fast as you can)

Optimize your imports

```
import sqlalchemy as sa
from sqlalchemy.sql.expression import false

from cems_api.resources.deliverables.models import (
  DeliverableDocument,
  DocumentWorkitem,
  WorkLog,
)
from cems_api.mixins.resourse_manager import (
    QueryHelperMixin, ManagerMixin
)
from cems_api.resources.project.helpers.global_attributes import \
    GlobalAttributesHelper
from ..helpers.deliverable_state_models import \
    DeliverableChangeModelHelperQueries
```

isort

isort will sort imports for you

Installing isort is as simple as:

```
pip install isort
```

or if you prefer

```
easy_install isort
```

Using isort

From the command line:

```
isort mypythonfile.py mypythonfile2.py
```

or recursively:

```
isort -rc .
```

or to see the proposed changes without applying them:

```
isort mypythonfile.py --diff
```

isort settings

```
[settings]
multi_line_output = 3
lines_after_imports = 2
```

PEP8

PEP8 checkers:

- flake8
- pyflakes
- pylint
- pylama
- flake8-isort
- flake8-import-order

pre-commit hook

flake8 --install-hook git
git config --bool flake8.strict true

Formatters:

- YAPF
- autopep8
- pep8ify

Note: flake8-isort defers all logic to isort,
the flake8-import-order comes bundled with it's own logic

Type annotations, mypy

Module typing supports type hints as specified by **PEP 484** and **PEP 526** (Variable Annotations). Type annotations designed on top of PEP-3107 at September 2014 and became part of standard library in Python 3.5

Pros

- Specifying author intent
- Like a doctoring, without the code rot
- Like a doctest, but compact
- Very useful when you do refactoring
- Easier code reviews
- @enforce.runtime_validation (pip install enforce)

Cons

- Most libraries aren't annotated
- You need to ignore (# type: ignore) missing annotations and use Any
- Wait for new version of mypy and upgrade
- enforce is much slower

Don’t write what you can skip

```
if doc_check:
    return False
if log_check:
    return False
if doc_check or log_check:
    return False
```

```
prefix = prefix + '_'
prefix += '_'
```

```
for _ in d.keys():
    pass
for _ in d:
    pass
```

How to write less code and make it more efficient

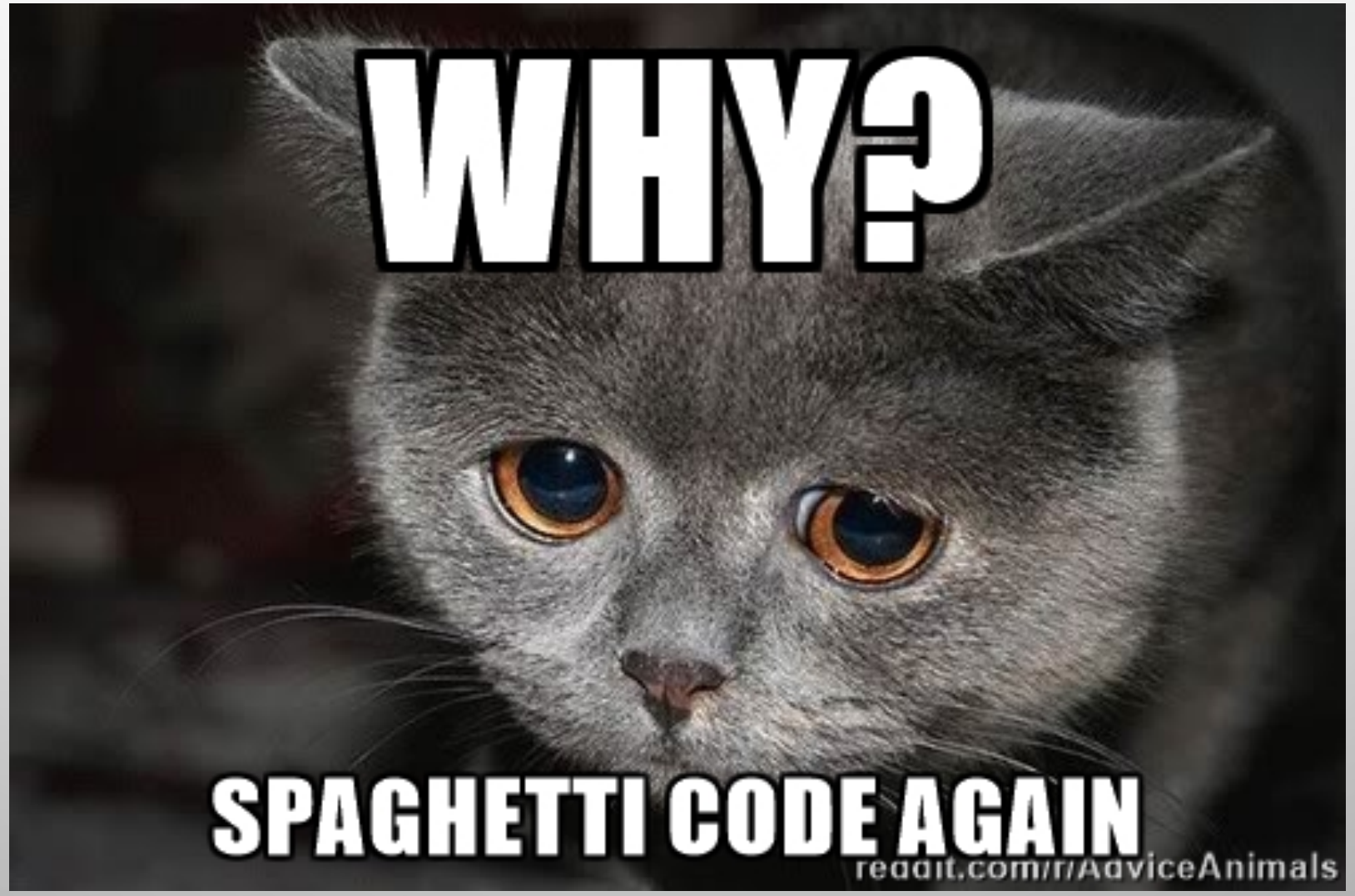
```
result = []
for i in range(10):
    s = i ** 2
    result.append(s)
sum(result)

sum([i ** 2 for i in range(10)])

sum(i ** 2 for i in range(10))
```

https://youtu.be/OSGv2VnC0go?t=47m56s

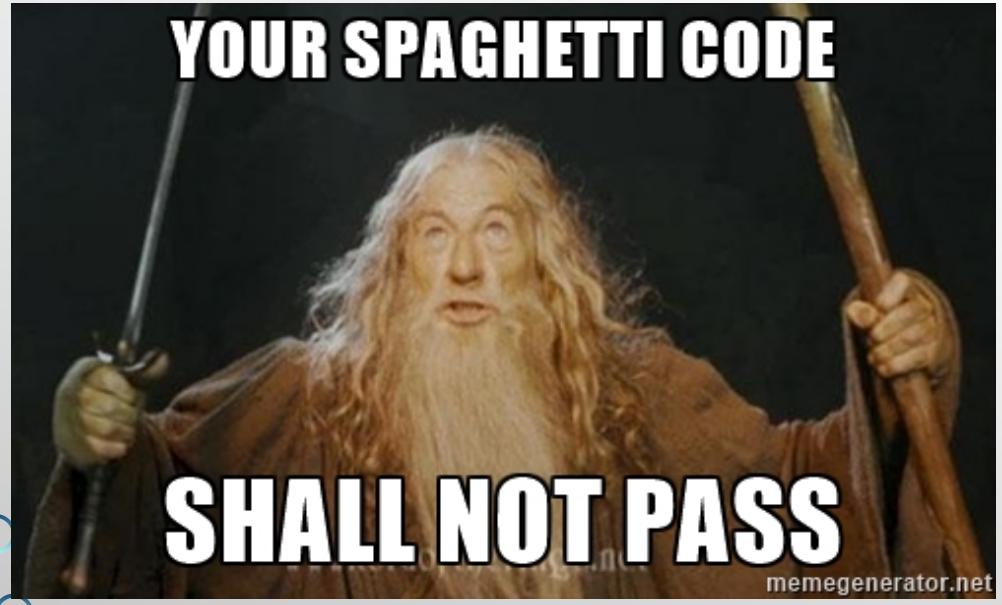
Use namedtuple

```
elif not insert_data and \
     (delete_data[0][0] == client_manager_id or
     delete_data[0][0] == program_manager_id or
     delete_data[0][0] == project_manager_id):
     pass

User = namedtuple('User', ['id', 'role'])
Role = namedtuple('Role', ['id', 'name'])
delete_data = User(1, Role(2, 'Project_manager'))


elif not insert_data and \
     (delete_data.role.id == client_manager_id or
     delete_data.role.id == program_manager_id or
     delete_data.role.id == project_manager_id):
     pass
```

Sometimes our code looks like



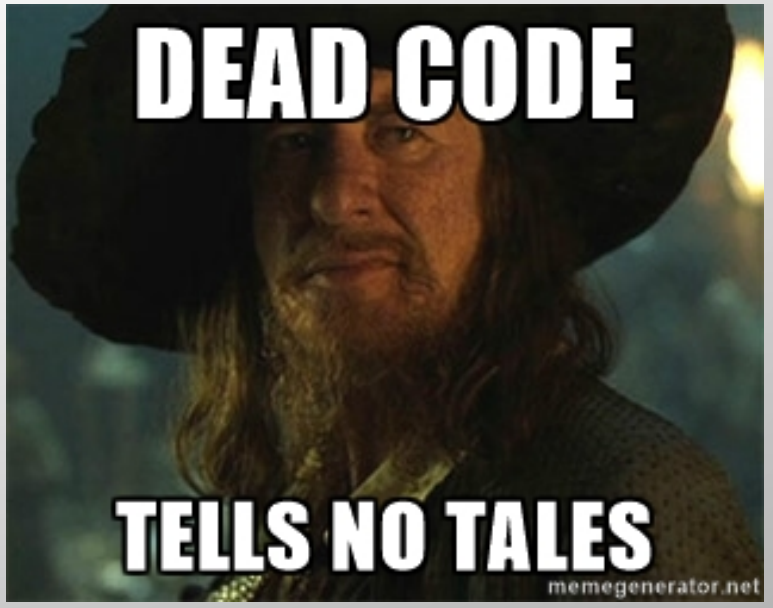
How to prevent spaghetti code



- Convert a code segment into a function that can be reused in future maintenance and refactoring efforts
- Try to write unit tests (if your code is hard to test than you should do refactoring)

Sometimes our code is dead

Remember that:



The best code is code that you don't even have

- Delete dead code as soon as you notice it
- Remember that version control has your back in case you ever need that code again
- Use “vulture” to detect dead code

Few words about pull requests

- **Who** will be the first reviewer of your pull request?
- **Who** takes responsibility for code merged code in your pull request?
- It's the same person
- It's **you**!

Virtualenv

virtualenv is a tool to create isolated Python environments.

virtualenvwrapper - wrappers for creating and deleting virtual environments and otherwise managing your development workflow, making it easier to work on more than one project

autoenv and direnv - automatically execute .env(autoenv) or .envrc(direnv) when you cd into directory.

Use pytest instead of unittest

```python
def incrementor(number):
    return number + 1
```

```python
import unittest


class IncrementorTest(unittest.TestCase):
    def test_incrementing(self):
        self.assertEqual(incrementor(1), 2)
```

```python
def test_incrementor():
    assert incrementor(1) == 2
```

@pytest.mark.parametrize

```
@pytest.mark.parametrize('number, expected', (
        (1, 2),
        (2, 3),
        (9, 10)
))
def test_increment(number, expected):
    assert incrementor(number) == expected
```

```
@pytest.mark.parametrize('number, expected', (
        (1, 2),
        (2, 3),
        (9, 9)
))
def test_increment(number, expected):
    assert incrementor(number) == expected
```

```
tests/test_helpers.py::test_increment[1-2] PASSED
tests/test_helpers.py::test_increment[2-3] PASSED
tests/test_helpers.py::test_increment[9-10] PASSED
```

```
tests/test_helpers.py::test_increment[1-2] PASSED
tests/test_helpers.py::test_increment[2-3] PASSED
tests/test_helpers.py::test_increment[9-9] FAILED

=================================================
_________________________________________________

number = 9, expected = 9

    @pytest.mark.parametrize('number, expected', (
            (1, 2),
            (2, 3),
            (9, 9)
    ))
    def test_increment(number, expected):
>       assert incrementor(number) == expected
E       assert 10 == 9
E        +  where 10 = incrementor(9)
```

Reusable tests

```python
import unittest


class MyTest(unittest.TestCase):
    def setUp(self):
        ...

    def tearDown(self):
        ...

    def my_test(self):
        ...
```

```python
class MyTestMixin:
    def setUp(self):
        ...

    def tearDown(self):
        ...
```

```python
class MyTest(MyTestMixin, unittest.TestCase):
    def setUp(self):
        ...
        super().setUp()

    def tearDown(self):
        ...
        super().tearDown()

    def my_test(self):
        ...
```

```python
@pytest.fixture
def db():
    ...


def my_test(db):
    ...
```

conftest.py

conftest.py - fixtures available in all tests

Conclusion pytest

- Pytest will make your life easier
- DRY when you write tests. Fixtures and conftest will help you with this

Simple is better than complex.

Kenneth Reitz

Urllib

```
import urllib.request


password_mgr = urllib.request.HTTPPasswordMgrWithDefaultRealm()
top_level_url = 'https://httpbin.org/basic-auth/user/passwd'
password_mgr.add_password(None, top_level_url, 'user', 'passwd')
handler = urllib.request.HTTPBasicAuthHandler(password_mgr)
opener = urllib.request.build_opener(handler)
response = opener.open(top_level_url)
print(response.read())
```

b'{\n "authenticated": true, \n "user": "user"\n}\n'

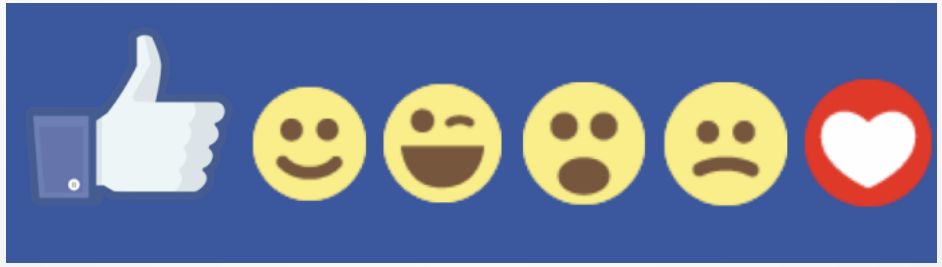
Requests

```
import requests


response = requests.get('https://httpbin.org/basic-auth/user/passwd',
                        auth=('user', 'passwd'))
print(response.content)
print(response.json())
```

b'{\n "authenticated": true, \n "user": “user"\n}\n'

{'authenticated': True, 'user': 'user'}

Scraping Facebook



```
urls = [
    'https://graph.facebook.com/v2.10/67920382572/feed/?limit=100'
    '&access_token={}&since=2017-07-10&until=2017-07-23&fields=reactions.'
    'type(LIKE).limit(0).summary(total_count)'.format(ACCESS_TOKEN),
    'https://graph.facebook.com/v2.10/67920382572/feed/?limit=100&'
    'access_token={}&since=2017-07-10&until=2017-07-23&fields=reactions.'
    ‘type(LOVE).limit(0).summary(total_count)'.format(ACCESS_TOKEN),
                                    …
]
```

From requests to grequests

```
responses = [requests.get(url) for url in urls]
```

Requests + Gevent = <3

GRequests allows you to use Requests with Gevent to make asynchronous HTTP Requests easily.

```
requests_ = (grequests.get(url) for url in urls)
responses = grequests.map(requests_)
```

Benchmarks

```
async def asyncio_benchmark():
    start = time.time()

    for _ in range(number_or_repetitions):
        futures = [loop.run_in_executor(None, requests.get, url)
                   for url in urls]

        for future in futures:
            await future
    print('Average elapsed time asyncio = {}'.format(
        (time.time() - start) / number_or_repetitions))
```

```
async def requests_threads_benchmark():
    start = time.time()
    for _ in range(number_or_repetitions):
        for url in urls:
            await session.get(url)

    print('Average elapsed time requests_threads = {}'.format(
        (time.time() - start) / number_or_repetitions))
```

Benchmarks results

Average elapsed time **requests** = 7.345373201370239

Average elapsed time **asyncio** = 1.6936199188232421

Average elapsed time **grequests** = 2.0158246040344237

Average elapsed time **requests_threads** = 6.250591206550598

aio-libs

Pinned repositories

aiohttp
Async http client/server framework (asyncio)
Python ★ 3.7k 663

aiopg
aiopg is a library for accessing a PostgreSQL database from the asyncio
Python ★ 483 73

aioredis
asyncio (PEP 3156) Redis support
Python ★ 399 67

aiomysql
aiomysql is a library for accessing a MySQL database from the asyncio
Python ★ 354 55

aiobotocore
asyncio support for botocore library using aiohttp
Python ★ 131 32

yarl
Yet another URL library
Python ★ 193 28

And many other asyncio-based libraries built with high quality!

Aiohttp and Sanic

Aiohttp - HTTP client/server for asyncio (**PEP 3156**).

Server example:

```
from aiohttp import web

async def handle(request):
    name = request.match_info.get('name', "Anonymous")
    text = "Hello, " + name
    return web.Response(text=text)

app = web.Application()
app.router.add_get('/', handle)
app.router.add_get('/{name}', handle)

web.run_app(app)
```

Sanic is a Flask-like Python 3.5+ web server that's written to go fast.

Hello World Example

```
from sanic import Sanic
from sanic.response import json

app = Sanic()

@app.route("/")
async def test(request):
    return json({"hello": "world"})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

asvetlov commented on Jul 9 Contributor

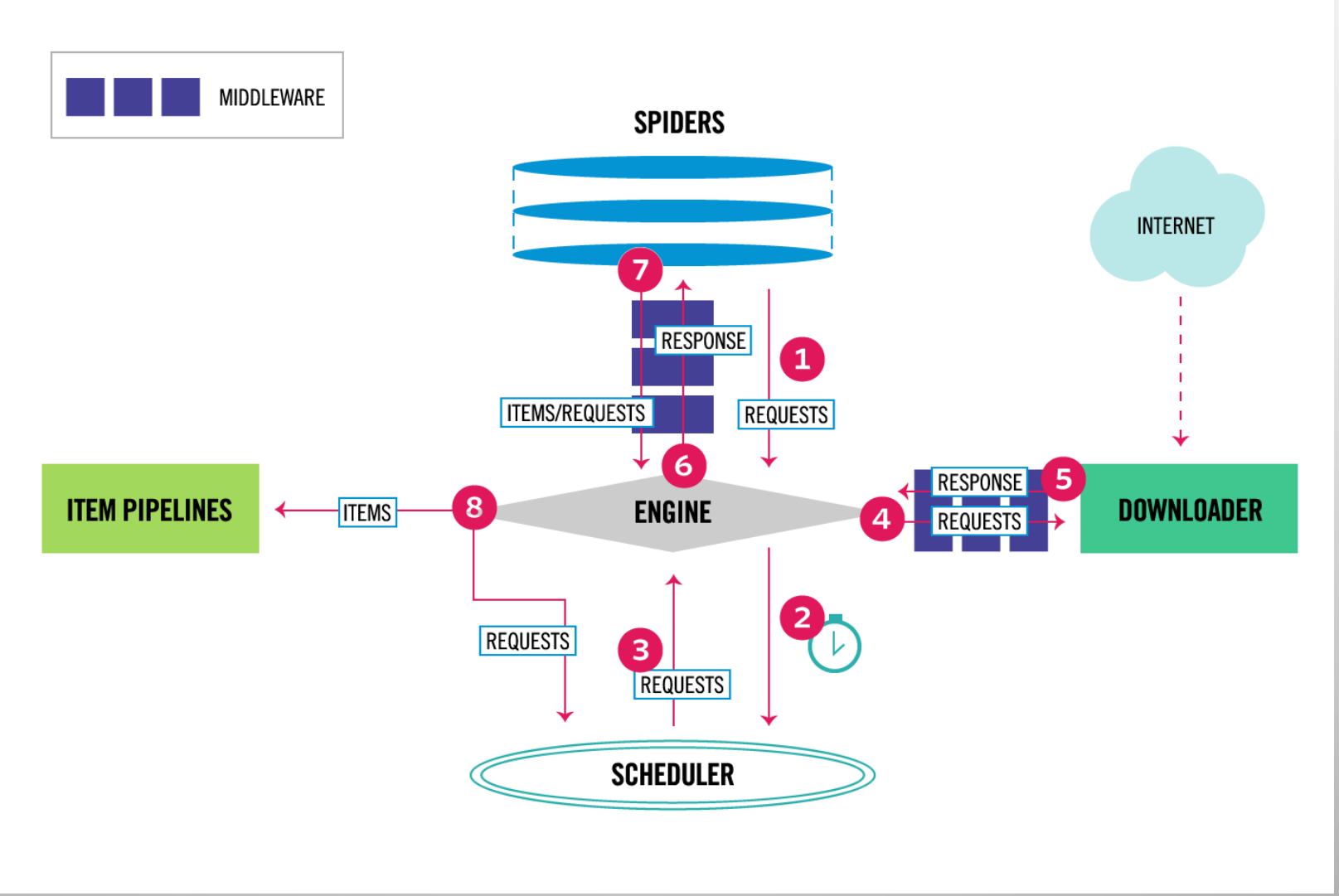
The reason is: aiohttp with disabled access log shows about 16,000 RPS on sanic's own benchmark. It's pretty much faster than 3,000 RPS from the table.

```
-+-----------+----------------------+----------------+---------------+
-| Server    | Implementation       | Requests/sec   | Avg Latency   |
-+===========+======================+================+===============+
-| Sanic     | Python 3.5 + uvloop  | 33,342         | 2.96ms        |
-+-----------+----------------------+----------------+---------------+
-| Wheezy    | gunicorn + meinheld  | 20,244         | 4.94ms        |
-+-----------+----------------------+----------------+---------------+
-| Falcon    | gunicorn + meinheld  | 18,972         | 5.27ms        |
-+-----------+----------------------+----------------+---------------+
-| Bottle    | gunicorn + meinheld  | 13,596         | 7.36ms        |
-+-----------+----------------------+----------------+---------------+
-| Flask     | gunicorn + meinheld  | 4,988          | 20.08ms       |
-+-----------+----------------------+----------------+---------------+
-| Kyoukai   | Python 3.5 + uvloop  | 3,889          | 27.44ms       |
-+-----------+----------------------+----------------+---------------+
-| Aiohttp   | Python 3.5 + uvloop  | 2,979          | 33.42ms       |
-+-----------+----------------------+----------------+---------------+
-| Tornado   | Python 3.5           | 2,138          | 46.66ms       |
-+-----------+----------------------+----------------+---------------+
```

Scrapy

- http://scrapy.org/
- Framework for building web crawlers
- Extracts structured data from unstructured web pages
- Inspired by Django
- Enables developers to focus on the rules to extract the data they want
- Does the hard parts of crawling
- Fast event-driven code

Scrapy architecture



Why Scrapy?

- Scrapy handles your requests asynchronously and it is really fast.
- Scrapy follows redirections and also avoids getting caught in <noscript> redirection traps.
- Scrapy retries failed requests and you can customize the retrying policy.
- Scrapy filters duplicate requests and you can customize the filter behaviour.
- Scrapy exports the scraped data in the most common formats, such as JSON, XML and CSV.
- Etc.

A command-line utility that creates projects from **cookiecutters** (project templates), e.g. creating a Python package project from a Python package project template.

Choose a template and copy its git URL. There are a number of available Python templates. Run the cookiecutter command with the git URL.

```
$ cookiecutter https://github.com/ionelmc/cookiecutter-pylibrary
```

Answer the prompts that appear, and you're done!

Github Marketplace

Codacy
Automated code reviews to help developers ship better software, faster

Codecov
Code coverage done right.® Group, merge, archive and compare coverage reports

Sentry
Real-time, cross-platform crash reporting and error logging

Travis CI
Test and deploy with confidence

Thank you!
Questions?