Just Python

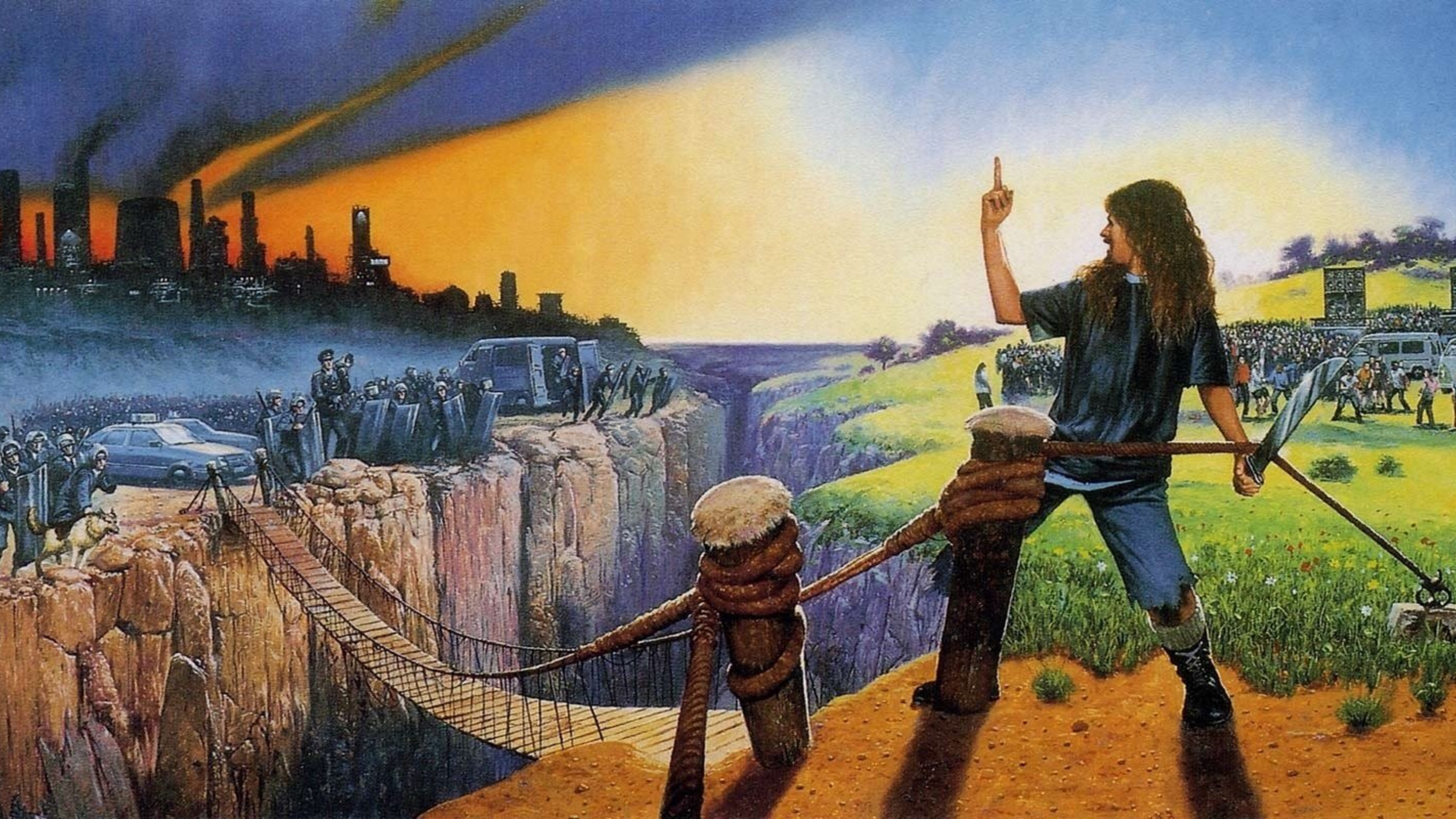
Subversive Edition

Manifesto Just Python

Vamos ver essas regras...

Você não pode:

- Usar bibliotecas externas
- Usar frameworks
- Falar sobre comunidade, carreira, empreendedorismo e outros temas que não sejam diretamente sobre código



Toy

O Framework Web de Brinquedo

Esperem! Não me expulsem!

Vamos colocar uma lupa nessas regras!

Regras Just Python

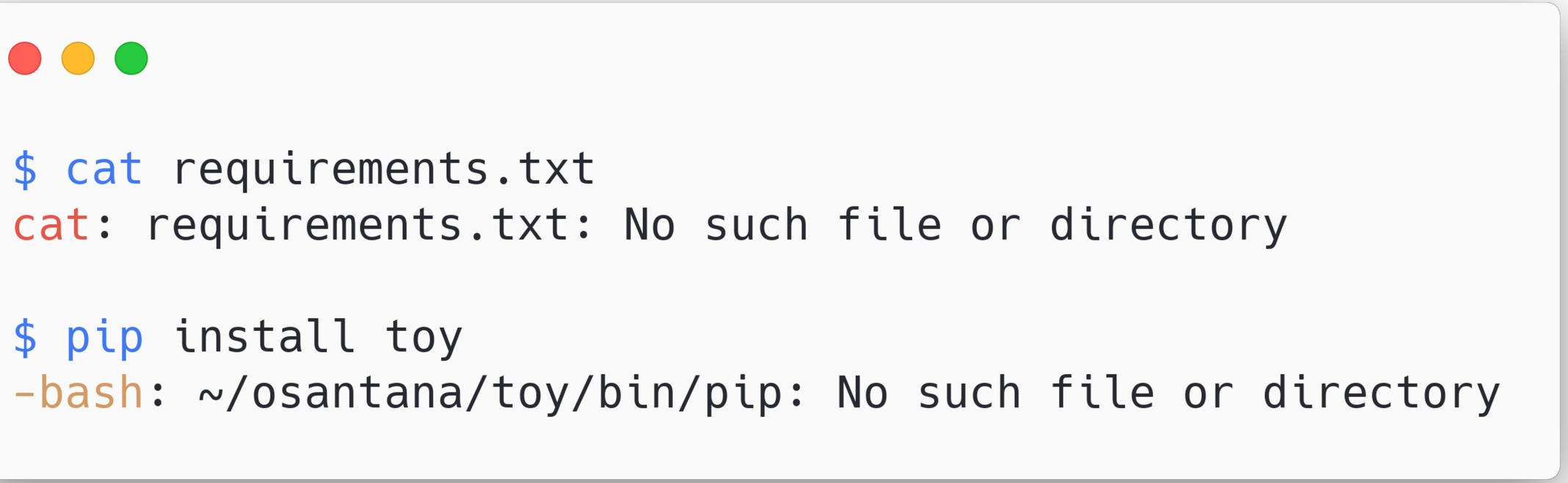
Você não pode:

- Usar bibliotecas externas
- Usar frameworks
- Falar sobre comunidade, carreira, empreendedorismo e outros temas que não sejam diretamente sobre código

Regras Just Python

Você não pode:

- Usar bibliotecas externas
- Usar frameworks
- Falar sobre comunidade, carreira, empreendedorismo e outros temas que não sejam diretamente sobre código


$ cat requirements.txt
cat: requirements.txt: No such file or directory
$ pip install toy
-bash: ~/osantana/toy/bin/pip: No such file or directory

Regras Just Python

Você não pode:

- Usar bibliotecas externas
- Usar frameworks
- Falar sobre comunidade, carreira, empreendedorismo e outros temas que não sejam diretamente sobre código

Regras Just Python

Você não pode:

- [redacted]
- Usar frameworks
- Falar sobre comunidade, carreira, empreendedorismo e outros temas que não sejam diretamente sobre código

Não falaria sobre carreira...

Estamos contratando!
http://99jobs.com/olist

Regras Just Python

Você não pode:

- [redacted]
- Usar frameworks
- Falar sobre comunidade, carreira, empreendedorismo e outros temas que não sejam diretamente sobre código

Regras Just Python

Você não pode:

- Usar frameworks

Regras Just Python

Você não pode:

- Usar frameworks

Regras Just Python

Você não pode:

- Usar frameworks ?

Regras Just Python

Você não pode:

- Usar frameworks

Regras Just Python

Você não pode:

- Usar frameworks

Usar != Fazer

Toy

Toy

Criando um Framework de Brinquedo

Toy

Criando um Framework de Brinquedo

(que não é uma piada de 1º de abril 🤪)

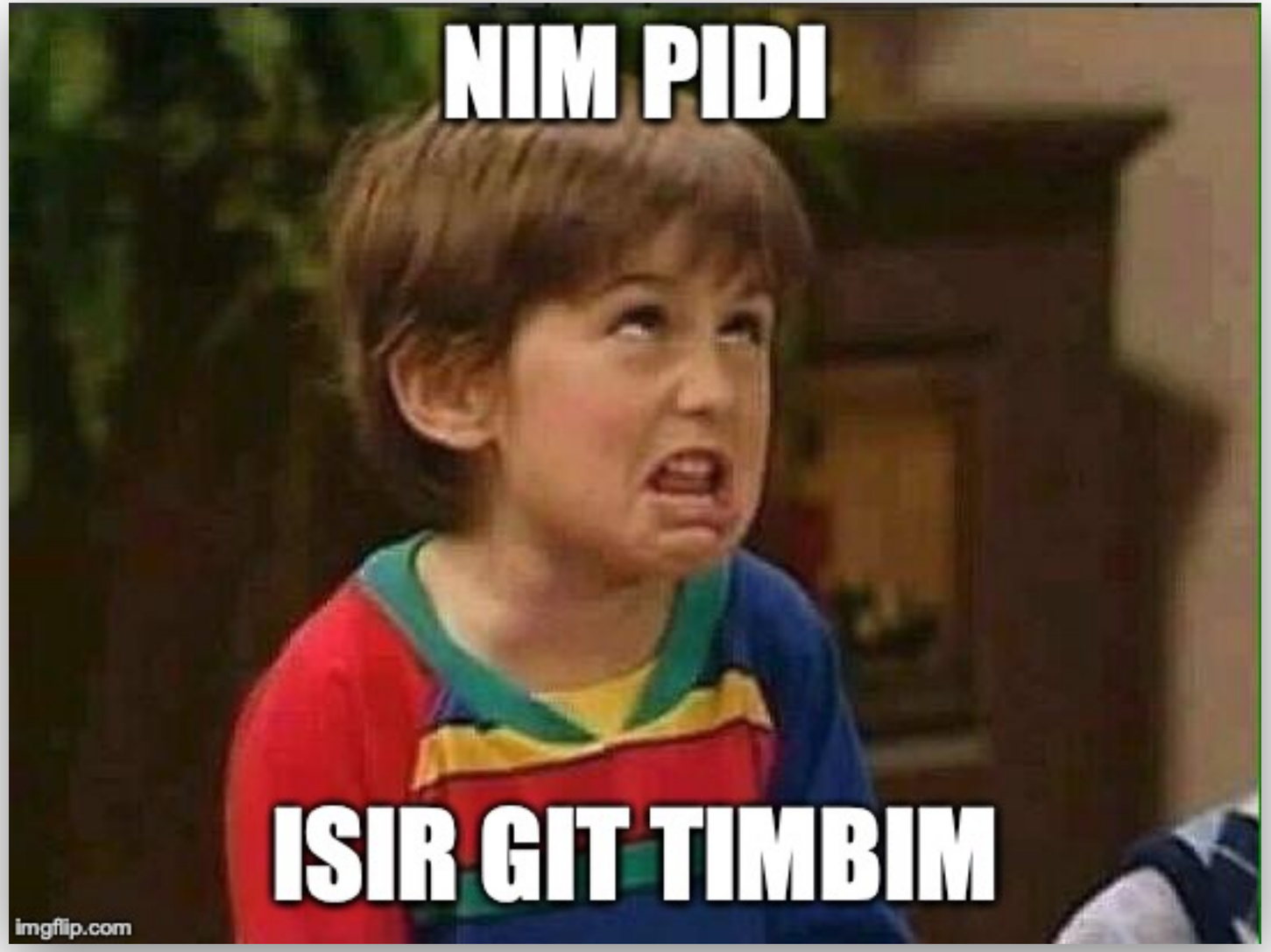
Sem dependência externa

Sem dependência externa

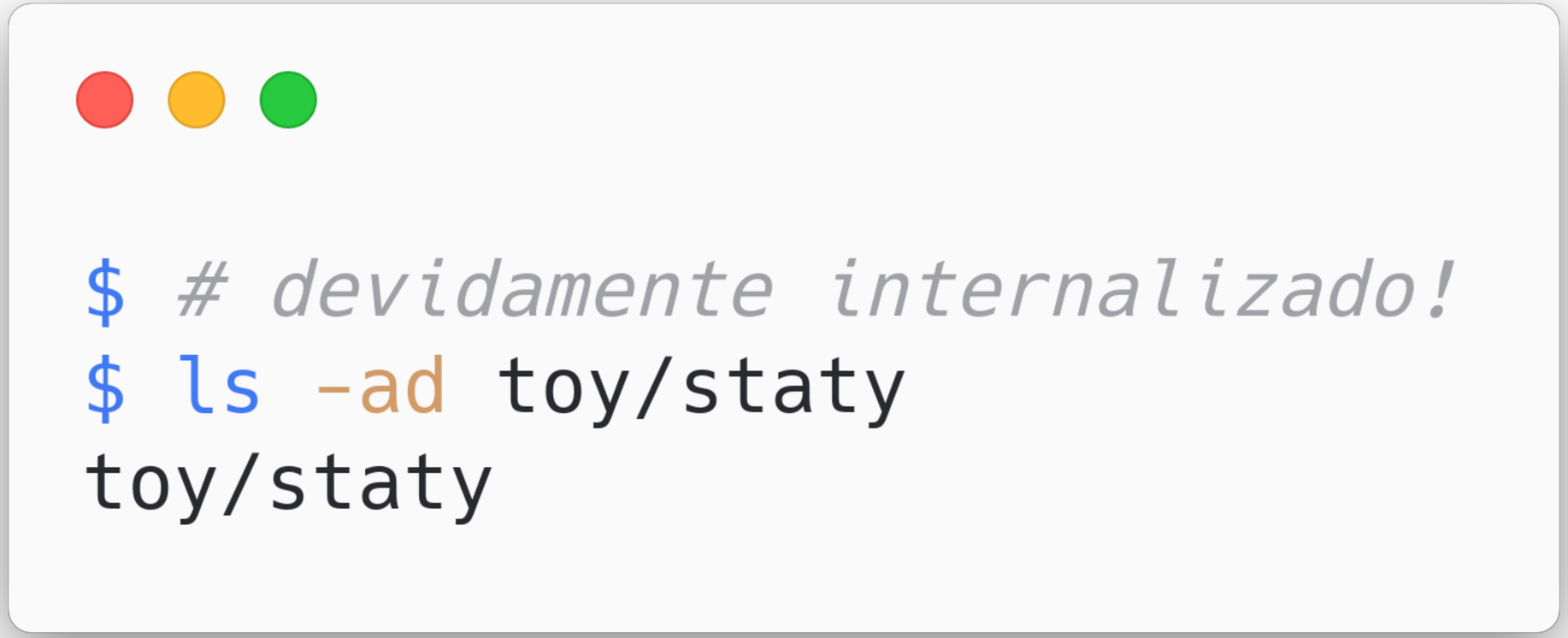
- A única dependência externa era o projeto **staty**. Mas o projeto também é Pure Python e eu que desenvolvi. Então ficou naquela 'gray area'.
  - https://github.com/**osantana**/staty ← Eu que fiz, tá?

Sem dependência externa

- A única dependência externa era o projeto **staty**. Mas o projeto também é Pure Python e eu que desenvolvi. Então ficou naquela 'gray area'.
  - https://github.com/**osantana**/staty ← Eu que fiz, tá?
- **Já que não pode pip, pode git?**


NIM PIDI
ISIR GIT TIMBIM
imgflip.com

ok...



Como fazer um Framework Pure Python?

Vou mostrar como faz mas não posso mostrar como **usa**...
estão ligados na regra, né?

Como fazer?

Como fazer?

1. Você não testa. Ninguém merece usar o **unittest** *builtin* do Python

Como fazer?

1. Você não testa. Ninguém merece usar o **unittest** *builtin* do Python

    - Só programadores Brainf*ck podem desenvolver sem testes.

Como fazer?

1. Você não testa. Ninguém merece usar o **unittest** *builtin* do Python

   - Só programadores Brainf*ck podem desenvolver sem testes.

   - Brincadeira... eu fiz teste com pytest mas não vou mostrar porque... enfim... (as regras...)

Como fazer?

1. Você não testa. Ninguém merece usar o **unittest** *builtin* do Python

   - Só programadores Brainf*ck podem desenvolver sem testes.

   - Brincadeira... eu fiz teste com pytest mas não vou mostrar porque... enfim... (as regras...)

2. Prestar atenção ao módulo **wsgiref** que implementa uma referência do padrão WSGI 1.0

Como fazer?

1. Você não testa. Ninguém merece usar o **unittest** *builtin* do Python

   - Só programadores Brainf*ck podem desenvolver sem testes.

   - Brincadeira... eu fiz teste com pytest mas não vou mostrar porque... enfim... (as regras...)

2. Prestar atenção ao módulo **wsgiref** que implementa uma referência do padrão WSGI 1.0

   - PEP-333 apenas (não tem tudo o que tem na PEP-3333, WSGI 1.0.1)

```python
from wsgiref.simple_server import WSGIRequestHandler, WSGIServer
from .app import my_wsgi_app

class HTTPServer:
    def __init__(self, app, wsgi_server=WSGIServer, **kwargs):
        self.app = app

    def run(self):
        server = WSGIServer(('localhost', 8080), WSGIRequestHandler)
        server.set_app(self.app)

        print(f"Serving on localhost:8080 (press ctrl-c to stop)...")

        try:
            server.serve_forever()
        except KeyboardInterrupt:
            self._print("\nStopping...")

server = HTTPServer(my_wsgi_app)
server.run()
```

Já temos um servidor.
Precisamos de uma **App**.

```python
def application(environ, start_response):
    status = '200 OK'
    output = 'Hello World!\n'
    response_headers = [('Content-type', 'text/plain'),
                        ('Content-Length', str(len(output)))]
    start_response(status, response_headers)
    return [output]
```

```python
from .http import Request, Response, WSGIResponse  # já vemos isso!


class Application:
    def __init__(self, **kwargs):
        self.initialize()

    def initialize(self):
        pass

    def call_handler(self, request: Request) -> Response:
      # ... where magic happens...

    def __call__(self, environ, start_response):
        request = Request(environ)
        response = self.call_handler(request)
        wsgi_response = WSGIResponse(response)

        start_response(wsgi_response.status, wsgi_response.headers)
        return wsgi_response.body
```

```python
from .http import Request, Response, WSGIResponse  # já vemos isso!


class Application:
    def __init__(self, **kwargs):
        self.initialize()

    def initialize(self):
        pass

    def call_handler(self, request: Request) -> Response:
      # ... where magic happens...

    def __call__(self, environ, start_response):
        request = Request(environ)
        response = self.call_handler(request)
        wsgi_response = WSGIResponse(response)

        start_response(wsgi_response.status, wsgi_response.headers)
        return wsgi_response.body
```

WSGI environ

```
from io import BytesIO

def environ_builder(method, path, input_stream=None):
    input_stream.seek(0, 2)
    end_pos = input_stream.tell()
    input_stream.seek(0)
    start_pos = input_stream.tell()
    content_length = end_pos - start_pos

    result = {
        'REQUEST_METHOD': method.upper(),
        'PATH_INFO': path,
        'QUERY_STRING': '',
        'ACCEPT': 'application/json',
        'ACCEPT_CHARSET': 'utf-8',
        'CONTENT_TYPE': 'application/json'),
        'CONTENT_LENGTH': str(content_length),
        'wsgi.input': BytesIO(input_stream),
    }

    result['HTTP_ACCEPT'] = result['ACCEPT']
    result['HTTP_ACCEPT_CHARSET'] = result['ACCEPT_CHARSET']
    result['HTTP_CONTENT_TYPE'] = result['CONTENT_TYPE']
    result['HTTP_CONTENT_LENGTH'] = result['CONTENT_LENGTH']

    return result
```

```python
from io import BytesIO

def environ_builder(method, path, input_stream=None):
    input_stream.seek(0, 2)
    end_pos = input_stream.tell()
    input_stream.seek(0)
    start_pos = input_stream.tell()
    content_length = end_pos - start_pos

    result = {
        'REQUEST_METHOD': method.upper(),
        'PATH_INFO': path,
        'QUERY_STRING': '',
        'ACCEPT': 'application/json',
        'ACCEPT_CHARSET': 'utf-8',
        'CONTENT_TYPE': 'application/json'),
        'CONTENT_LENGTH': str(content_length),
        'wsgi.input': BytesIO(input_stream),
    }

    result['HTTP_ACCEPT'] = result['ACCEPT']
    result['HTTP_ACCEPT_CHARSET'] = result['ACCEPT_CHARSET']
    result['HTTP_CONTENT_TYPE'] = result['CONTENT_TYPE']
    result['HTTP_CONTENT_LENGTH'] = result['CONTENT_LENGTH']

    return result
```

HTTP Request

```python
from urllib.parse import parse_qs

class Request:
    def __init__(self, environ):
        self.method = environ.get('REQUEST_METHOD', 'GET').upper()
        self.path = environ.get('PATH_INFO', '/')
        self.query_string = parse_qs(environ['QUERY_STRING'])
        self.headers = {}  # parse headers started with 'HTTP_'
        self.content_type = environ.get('CONTENT_TYPE', 'application/octet-stream')
        self.accept = parse_accept(self.headers.get('Accept', self.content_type))
        self.path_arguments = {}

        try:
            self.content_length = int(environ.get('CONTENT_LENGTH', 0))
        except (TypeError, ValueError):
            self.content_stream = 0

        self.content_stream = environ['wsgi.input']

    @property
    def data(self):
         content = self.content_stream.read(self.content_length)
        return content.decode('utf-8')
```

```python
from urllib.parse import parse_qs

class Request:
    def __init__(self, environ):
        self.method = environ.get('REQUEST_METHOD', 'GET').upper()
        self.path = environ.get('PATH_INFO', '/')
        self.query_string = parse_qs(environ['QUERY_STRING'])
        self.headers = {}  # parse headers started with 'HTTP_'
        self.content_type = environ.get('CONTENT_TYPE', 'application/octet-stream')
        self.accept = parse_accept(self.headers.get('Accept', self.content_type))
        self.path_arguments = {}

        try:
            self.content_length = int(environ.get('CONTENT_LENGTH', 0))
        except (TypeError, ValueError):
            self.content_stream = 0

        self.content_stream = environ['wsgi.input']

    @property
    def data(self):
         content = self.content_stream.read(self.content_length)
        return content.decode('utf-8')
```

```python
from urllib.parse import parse_qs

class Request:
    def __init__(self, environ):
        self.method = environ.get('REQUEST_METHOD', 'GET').upper()
        self.path = environ.get('PATH_INFO', '/')
        self.query_string = parse_qs(environ['QUERY_STRING'])
        self.headers = {}  # parse headers started with 'HTTP_'
        self.content_type = environ.get('CONTENT_TYPE', 'application/octet-stream')
        self.accept = parse_accept(self.headers.get('Accept', self.content_type))
        self.path_arguments = {}

        try:
            self.content_length = int(environ.get('CONTENT_LENGTH', 0))
        except (TypeError, ValueError):
            self.content_stream = 0

        self.content_stream = environ['wsgi.input']

    @property
    def data(self):
         content = self.content_stream.read(self.content_length)
        return content.decode('utf-8')
```

```python
from urllib.parse import parse_qs

class Request:
    def __init__(self, environ):
        self.method = environ.get('REQUEST_METHOD', 'GET').upper()
        self.path = environ.get('PATH_INFO', '/')
        self.query_string = parse_qs(environ['QUERY_STRING'])
        self.headers = {}  # parse headers started with 'HTTP_'
        self.content_type = environ.get('CONTENT_TYPE', 'application/octet-stream')
        self.accept = parse_accept(self.headers.get('Accept', self.content_type))
        self.path_arguments = {}

        try:
            self.content_length = int(environ.get('CONTENT_LENGTH', 0))
        except (TypeError, ValueError):
            self.content_stream = 0

        self.content_stream = environ['wsgi.input']

    @property
    def data(self):
         content = self.content_stream.read(self.content_length)
        return content.decode('utf-8')
```

HTTP Response

```
from io import BytesIO

from .staty import Ok

class Response:
    def __init__(self, data, status=None, headers, content_type):
        if status is None:
            status = Ok()

        self.status = status
        self.data = data
        self.content_type = content_type

        self.headers = headers
        self.headers['Content-Type'] = f'{content_type}'

    @property
    def content_stream(self):
        if self.data is None or self.charset is None:
            return BytesIO()

        return BytesIO(self.data.encode(self.charset))
```

WSGI Response

```
class WSGIResponse:
    def __init__(self, response) -> None:
        self.response = response

    @property
    def status(self):
        return str(self.response.status)

    @property
    def headers(self) -> list:
        headers = []
        for key, value in self.response.headers.items():
            headers.append((key, value))
        return headers

    @property
    def body(self):
        return [self.response.content_stream.read()]
```

```python
from .http import Request, Response, WSGIResponse  # já vemos isso!


class Application:
    def __init__(self, **kwargs):
        self.initialize()

    def initialize(self):
        pass

    def call_handler(self, request: Request) -> Response:
      # ... where magic happens...

    def __call__(self, environ, start_response):
        request = Request(environ)
        response = self.call_handler(request)
        wsgi_response = WSGIResponse(response)

        start_response(wsgi_response.status, wsgi_response.headers)
        return wsgi_response.body
```

```python
from .http import Request, Response, WSGIResponse  # já vemos isso!


class Application:
    def __init__(self, **kwargs):
        self.initialize()

    def initialize(self):
        pass

    def call_handler(self, request: Request) -> Response:
      # ... where magic happens...

    def __call__(self, environ, start_response):
        request = Request(environ)
        response = self.call_handler(request)
        wsgi_response = WSGIResponse(response)

        start_response(wsgi_response.status, wsgi_response.headers)
        return wsgi_response.body
```

```python
from .http import Request, Response, WSGIResponse  # já vemos isso!


class Application:
    def __init__(self, **kwargs):
        self.initialize()

    def initialize(self):
        pass

    def call_handler(self, request: Request) -> Response:
      # ... where magic happens...

    def __call__(self, environ, start_response):
        request = Request(environ)
        response = self.call_handler(request)
        wsgi_response = WSGIResponse(response)

        start_response(wsgi_response.status, wsgi_response.headers)
        return wsgi_response.body
```

Rotas e Handlers

O básico do básico...

```
import re

class Route:
    def __init__(self, path, handler):
        self.path = path
        self.handler = handler
        self.pattern = re.compile(path)
        self.path_arguments = {}

    def match(self, path):
        match = self.pattern.search(path)
        if not match:
            return

        self.path_arguments.update(match.groupdict())
        return self.path_arguments

class Routes:
    def __init__(self):
        self._routes = []

    def add(self, route: Route):
        self._routes.append(route)

    def add_route(self, path, handler):
        self.add(Route(path, handler))

    def match(self, path):
        return [route for route in self._routes if route.match(path) is not None]
```

```python
from .routing import Routes

class Application:
    def __init__(self, **kwargs):
        self.routes = Routes()
        self.initialize()  # implemented in subclass

    def add_route(self, path, handler):
        self.routes.add_route(path, handler)

    def call_handler(self, request) -> Response:
        path = request.path
        routes = self.routes.match(path) # 404 if empty

        for route in routes:
            request.path_arguments.update(route.path_arguments)
            try:
                return route.handler(request)
            except MethodNotAllowedException:
                continue
            except NotFoundException:
                return # 404
            except Exception:
                return # error 500

        return # http method not allowed
```

```python
from .routing import Routes

class Application:
    def __init__(self, **kwargs):
        self.routes = Routes()
        self.initialize()  # implemented in subclass

    def add_route(self, path, handler):
        self.routes.add_route(path, handler)

    def call_handler(self, request) -> Response:
        path = request.path
        routes = self.routes.match(path) # 404 if empty

        for route in routes:
            request.path_arguments.update(route.path_arguments)
            try:
                return route.handler(request)
            except MethodNotAllowedException:
                continue
            except NotFoundException:
                return # 404
            except Exception:
                return # error 500

        return # http method not allowed
```

```python
from .routing import Routes

class Application:
    def __init__(self, **kwargs):
        self.routes = Routes()
        self.initialize()  # implemented in subclass

    def add_route(self, path, handler):
        self.routes.add_route(path, handler)

    def call_handler(self, request) -> Response:
        path = request.path
        routes = self.routes.match(path) # 404 if empty

        for route in routes:
            request.path_arguments.update(route.path_arguments)
            try:
                return route.handler(request)
            except MethodNotAllowedException:
                continue
            except NotFoundException:
                return # 404
            except Exception:
                return # error 500

        return # http method not allowed
```

Usando Rotas

```
from toy.application import Application
from toy.server import HTTPServer
from toy.http import Response

def hello_world(request):
    return Response('Hello World!',content_type='text/plain')

class MyApp(Application):
    def initialize(self):
        self.add_route(r'/', hello_world)
```

Usando Rotas

```
from toy.application import Application
from toy.server import HTTPServer
from toy.http import Response

def hello_world(request):
    return Response('Hello World!',content_type='text/plain')

class MyApp(Application):
    def initialize(self):
        self.add_route(r'/', hello_world)
```

E tem mais...

Class Handlers

```
from .http import HTTP_METHODS
from .staty import exceptions as error_status

class Handler:
    allowed_methods = []

    def __init__(self, methods):
        self._methods = set(m.lower() for m in self.allowed_methods)
        self._methods.update(m.lower() for m in methods)
        self._methods = self._methods.intersection(m.lower() for m in HTTP_METHODS)

    def _find_handler(self, request):
        if request.method.lower() not in self._methods:
            raise error_status.MethodNotAllowedException()

        try:
            return getattr(self, request.method.lower())
        except AttributeError:
            raise error_status.MethodNotAllowedException()

    def __call__(self, request):
        handler = self._find_handler(request)
        return handler(request)
```

Class Handlers

```python
from toy.application import Application
from toy.http import Response
from toy.handlers import Handler

class MyHandler(Handler):
    allowed_methods = ['get']
    def get(self, request):
        return Response('Hello World!', content_type='text/plain')

class MyApp(Application):
    def initialize(self):
        self.add_route(r'/', MyHandler())
```

Resources

```python
from toy.fields import UUIDField, CharField
from toy.resources import Resource


class MyResource(BaseResource):
    fields = [
        UUIDField(name='id', required=True, lazy=True),
        CharField(name='name', max_length=255, required=True),
    ]

    @classmethod
    def do_get(cls, request=None, application_args=None):
        # ... get object (eg. from database) and return a Resource instance...
        return cls(request, application_args)

    def do_create(self, parent_resource=None):
        # ... create object (POST)

    def do_remove(self):
        # ... remove object (DELETE)

    def do_replace(self):
        # ... replace object (PUT)

    def do_change(self, **kwargs):
        # ... change object (PATCH)
```

Usando o Framework

```
from toy.server import HTTPServer
from toy import handlers, resources, fields, http, application

class MyResource(resources.Resource):
    fields = [fields.CharField('name', max_length=255)]

    @classmethod
    def do_get(cls, request, app_args):
        return  cls(name='World')

class MyHandler(handlers.ResourceHandler):
    allowed_methods = ['get']
    resource_type = MyResource

class MyApp(application.Application):
    def initialize(self):
        self.add_route(r'/', MyHandler())

server = HTTPServer(MyApp())
server.run()
```

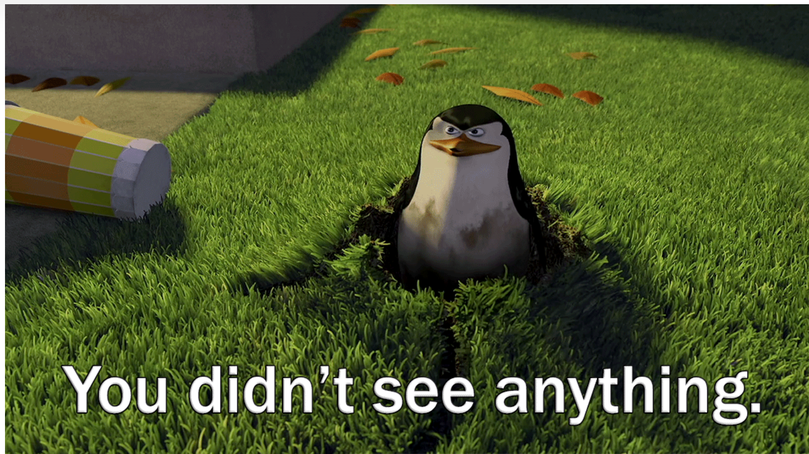
Usando o Framework

```
$ curl -H 'Accept: application/json' http://localhost:8080
{"name": "World"}
```

~~Usando o~~ Framework

Ops!

```
$ curl -H 'Accept: application/json' http://localhost:8080
{"name": "World"}
```



http://github.com/osantana/toy