Python Clean Code

Codando no nível Wakanda

Daniel Nascimento

Dev Consultant - **Thoughtworks**


O que é
Clean Code?

Filosofia cuja o principal objetivo é **aplicar técnicas** para **facilitar a escrita e leitura** de um código

Aprender a criar **códigos limpos** é uma **tarefa árdua** e requer mais do que conhecimento dos princípios e padrões. Você deve **suar a camisa; praticar sozinho** e ver que **cometeu erros,** assistir os outros praticarem e errarem, vê-los **tropeçar** e **refazer seus passos**, vê-los agonizar para tomar **decisões** e o preço que pagarão por terem tomado da maneira errada.

Robert C. Martin

Clean Code

O código faz

- Uma coisa bem
- O que é esperado

A busca por

- Qualidade de código
- Readability - fácil de ler
- Desenvolvimento ágil

DRY Principle

Don't Repeat Yourself

Evite códigos duplicados

- Dificéis de manter
- Propenso a erros

Obfuscated code

```
1 import time as t
2 while 1:print'\n'*99+'\n'.join([' '.join([['#'*4,"#  #","#   ","   #"]\
3 [int("01110333330302003030110330203002010033330101001030"[int(z)*5+y])]\
4 for z in t.strftime("%H%M%S")])for y in range(5)])+'\n',t.sleep(1)
```

O que é ofuscado?

```
def elapse(year):
  days = 365
  if year % 4 == 0 or (year % 100 == 0 and year % 400 == 0):
    days += 1

  for day in range(1, days + 1):
    print("Day {} of {}".format(day, year))
```

Desofuscado

```
def elapse(year):
  days = 365
  if year % 4 == 0 or (year % 100 == 0 and year % 400 == 0):
    days += 1

  for day in range(1, days + 1):
    print("Day {} of {}".format(day, year))
```

```
def elapse(year):
 days = 365
 if is_leap(year):
   days += 1
```

```
def is_leap(year):
 ...
```

Pros e cons

- Dificil de ler
- Não há separação de responsabilidade (função)

- Processo de compilação/build
- Outras linguagens fazem
- Ocorre no processo de minificação

Não é Clean Code

- Complexo, código ofuscado **(não legível)**
- Código duplicado
- Código que não revela sua **principal intenção**

Boas práticas

Nomes significativos

- Revelem **propósito**
- Passíveis de busca

```
ymd = datetime.now()              #Ruim

currentDate = datetime.now()      #Bom
```

Boas práticas

Evite números mágicos (aleatórios)

```
if my_var === 100:
   do_something()                #Ruim

TOTAL_PERCENT = 100
if my_var === TOTAL_PERCENT:   #Bom
    do_something()
```

Boas práticas

Condicionais - Utilize booleanos de **forma implícita**:

```
nome = 'joao'
lista_nomes = ['maria', 'joao', 'daniel']

if nome in lista_nomes:
  return true
else:
  return false                    # Ruim

return nome in lista_nomes        # Bom
```

Boas práticas

Condicionais - **evitar** condicionais negativas:

```
if should_not_process():
if not should_not_process():        # Ruim


if should_process():
if not should_process():            # Bom
```

Padrões de Design

Decorators

@decorators - PEP-318

Modificação de objetos existentes

- Novas funcionalidades
- Não modificação de estrutura

Decorators

```
def upper_decor(function):
  def wrapper():
    func = function()
    make_uppercase = func.upper()

    return make_uppercase

  return wrapper
```

```
def say_hello():
 return 'hello world'

decor = upper_decor(say_hello)
print(decorate())

---------------------------------

@upper_decor
def say_goodbye():
  return 'goodbye people'
```

Context Managers

Padrão de design

- Condições de entrada `__enter__():`
- Condições de saída `__exit__():`

Separação de responsabilidades

Gerenciamento de recursos

- Arquivos
- DB

Context Managers

Comando with - PEP-343

```
with open(filename) as fd:
    do_something(fd)
```

Context Managers

```
def stop_db():
 run("stop database")

def start_db():
 run("start database")

class DBHandler():
 def __enter__(self):
   stop_db()
   return self

 def __exit(self, *args):
   start_db()
```

```
def main():
 with DBHandler():
   run("dump database")
   ...
```

Propriedades

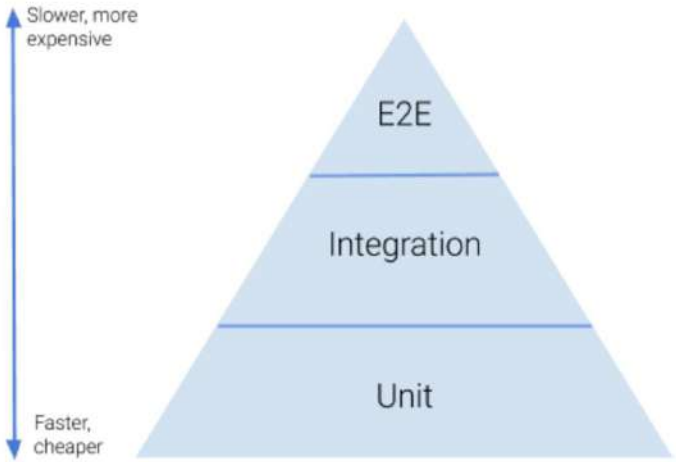
```
class Connector:
    def __init__(self, source):
    self.source = source
    self._timeout = 60


>>> conn = Connector("postgresql://localhost")
>>> conn.source
'postgresql://localhost'
>>> conn._timeout
60
>>> conn.__dict__
{'source': 'postgresql://localhost', '_timeout': 60}
```

Qualidade de código

Guia de estilo do Python: PEP-8

Pirâmide de testes

- Testes unitários

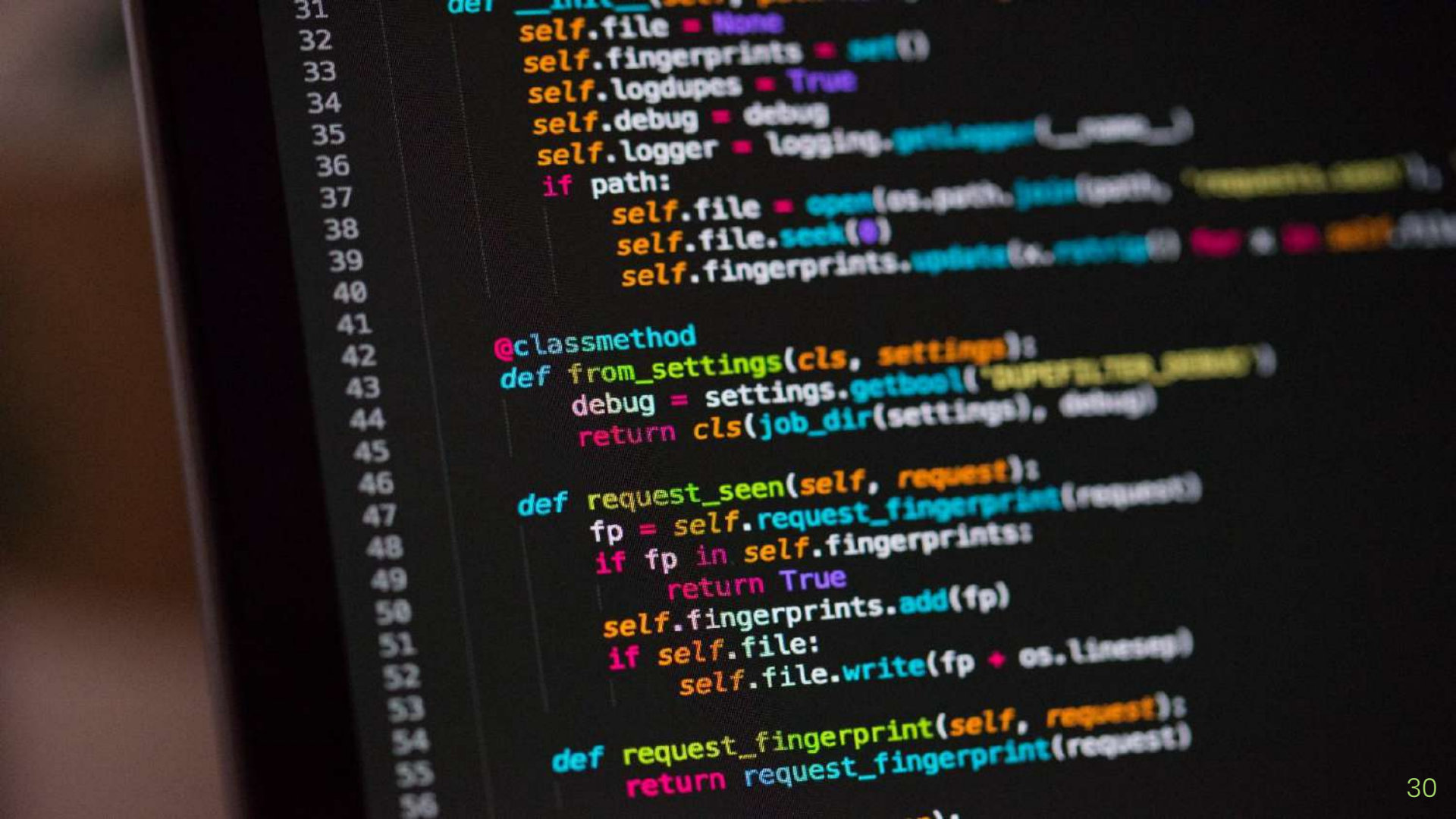


Ferramentas

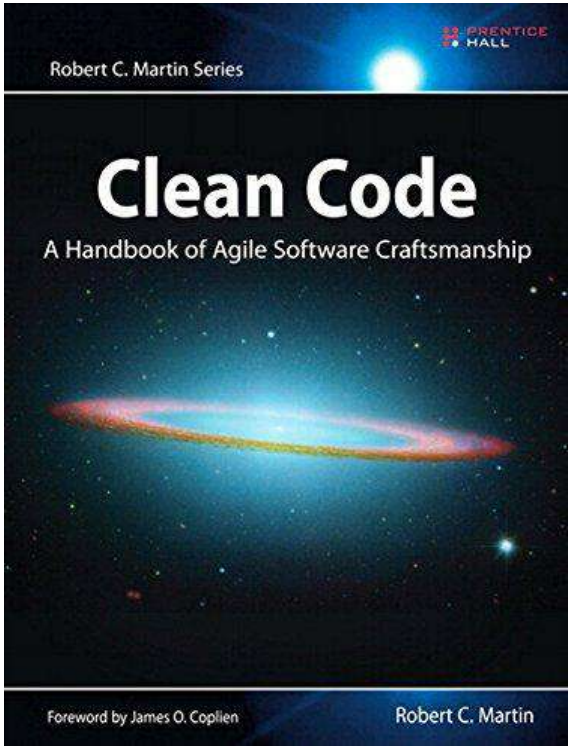
- Pylint
- Pycodestyle
- Flake8
- Pyflakes

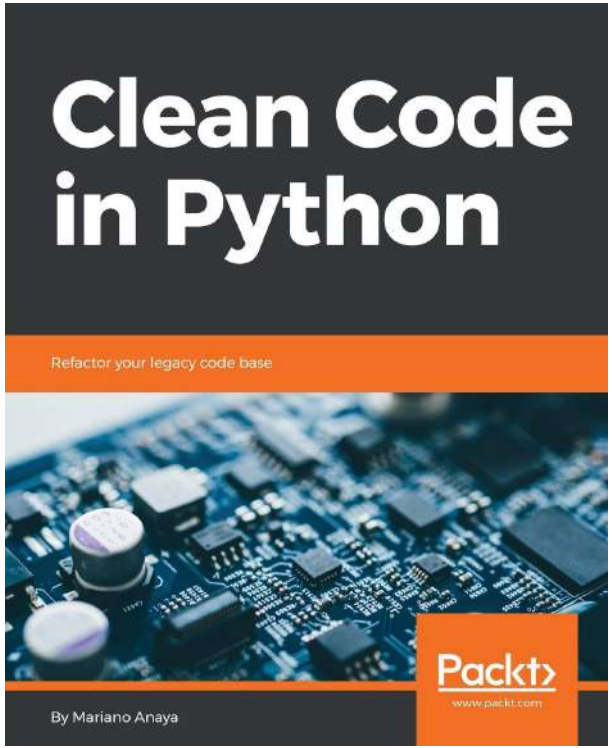
- Pychecker
- Mypy
- Pep8
- PyLama

- Pyminifier
- Pymor


self.file = None
self.fingerprints = set()
self.logdupes = True
self.debug = debug
self.logger = logging.
if path:
self.file = open(os.
self.file.seek(0)
self.fingerprints.
@classmethod
def from_settings(cls, settings):
debug = settings.getbool(
return cls(job_dir(settings), debug)
def request_seen(self, request):
fp = self.request_fingerprint(
if fp in self.fingerprints:
return True
self.fingerprints.add(fp)
if self.file:
self.file.write(fp +
def request_fingerprint(self, request):
return request_fingerprint(request)

Referências – livros





Referências – links

Python Code Quality Authority:

- https://github.com/PyCQA

Python Enhancement Proposals:

- https://www.python.org/dev/peps

Obrigaduu!

Duvidas?

- linkedin.com/in/daniel-nascimento/
- deo.daniel@gmail.com