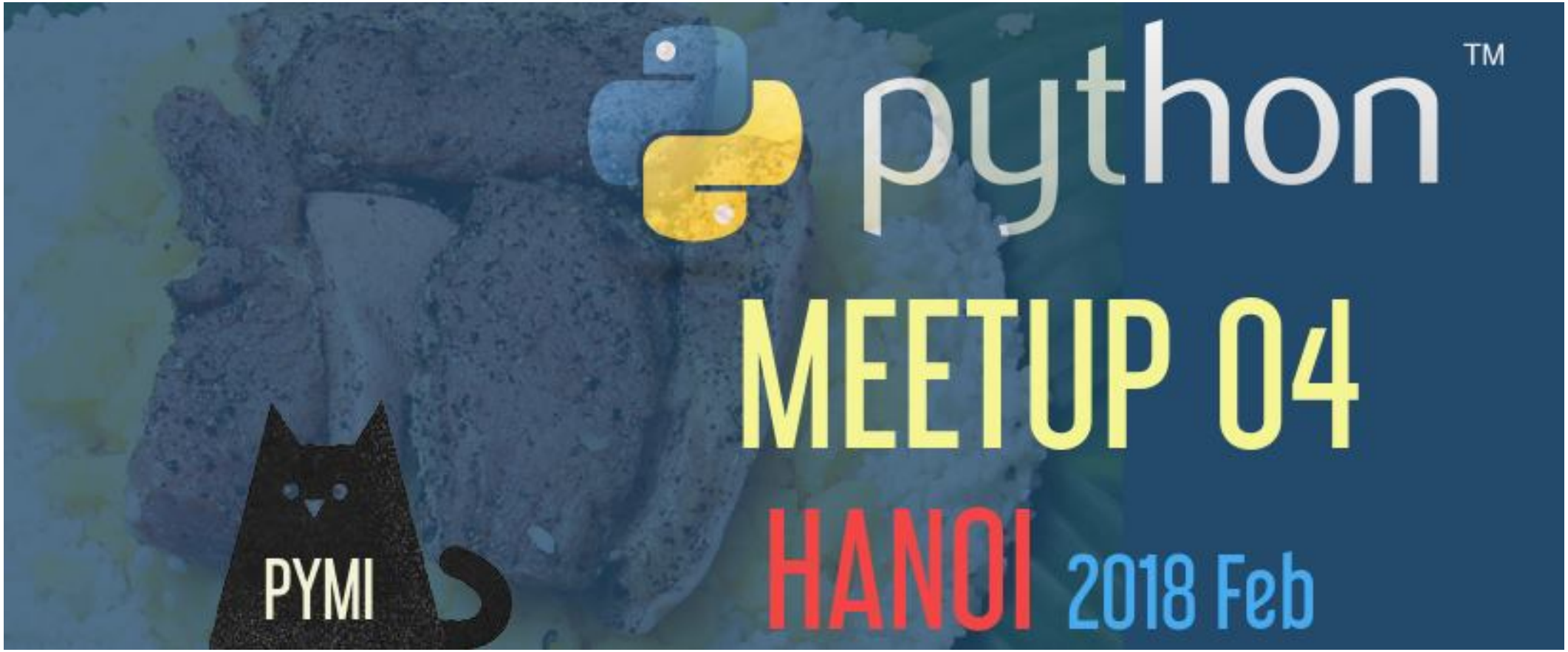
“CLEAN CODE & Python”



Trình bày: Nguyễn Quang Vinh

Nguyễn Quang Vinh

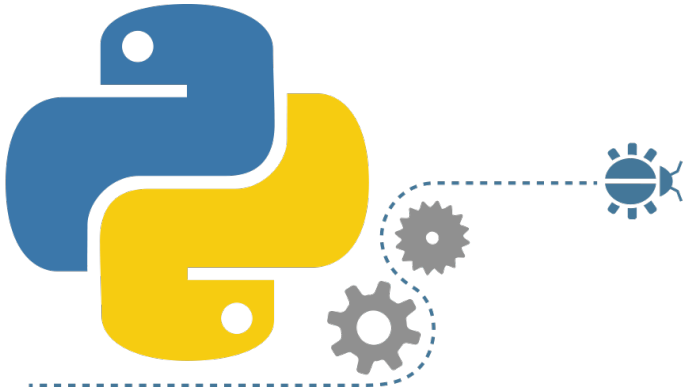
Python .NET

Developer & Teacher

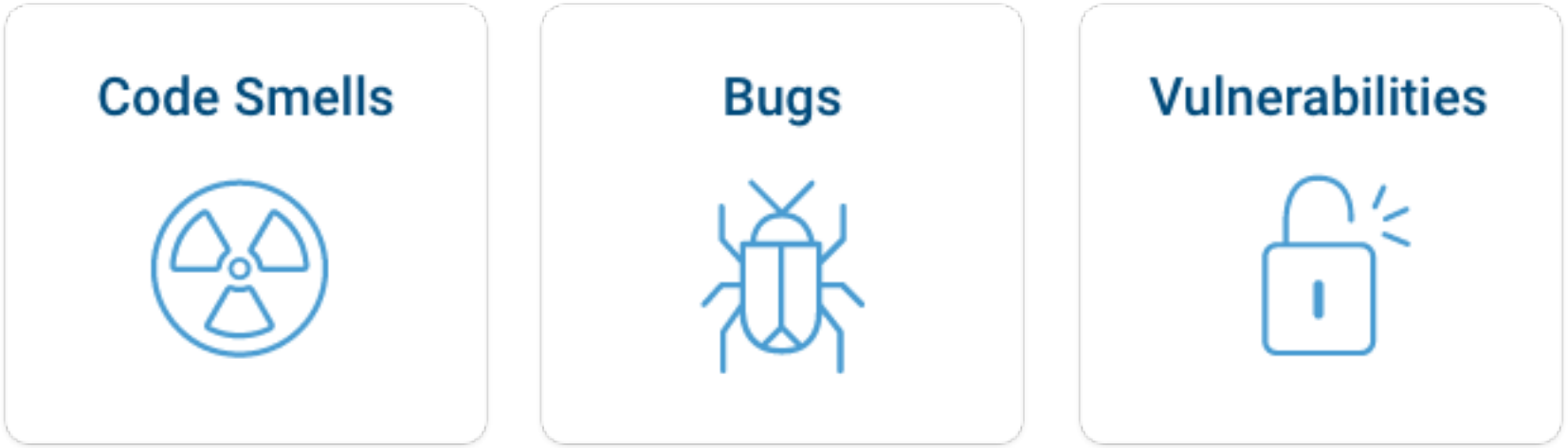
Contact me: quangvinh19862003@gmail.com

“CLEAN CODE & Python”

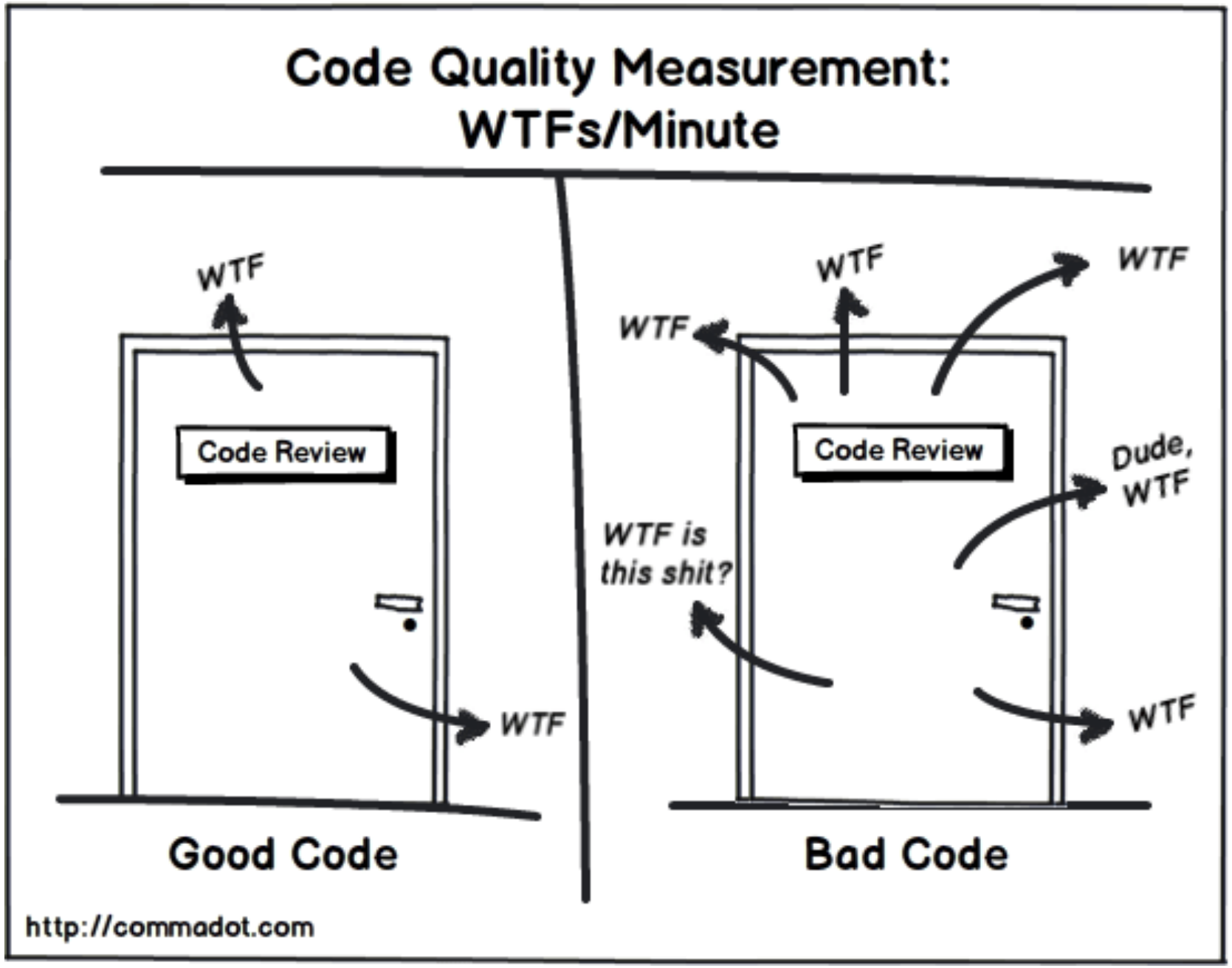
“Clean code"



“Clean code” in Python



“Clean code” tools for devs


Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude,
WTF
WTF is
this shit?
WTF
Bad Code
http://commadot.com

a “BAD CODE” story

Ta cùng đến với một câu truyện ngắn :

- Một công ty viết một ứng dụng dạng “hot trend”
- Ứng dụng nhanh chóng phổ biến, tiếp cân nhiều người dùng
- Nhiều lỗi và yêu cầu nâng cấp chức năng phát sinh
- Công ty không khắc phục được lỗi
- Số người dùng tăng lên, ứng dụng có hiện tượng quá tải
- Sản phẩm “lỗi”, người dùng bỏ rơi
- Ứng dụng chết – Công ty Phá Sản

Lý do được nêu ra : Ứng dụng chết do công ty không thể xử lý được xự lộn xộn trong mã mà công ty đã tạo ra, càng nhiều chức năng thì càng không quản lý và xử lý được.

a “BAD CODE” story

Mã xấu được sinh ra ngay trong quá trình code:

- Làm vội vàng, áp lực thời gian
- Cứ có sản phẩm chạy được đã, chấp nhận lộn xộn trong code
- Nhìn thấy vấn đề nhưng suy nghĩ để lại cuối cùng, giải quyết sau
- Từ một mã xấu nhỏ, kéo theo các mã xấu tiếp theo, chồng chất lên nhau rồi không thể giải quyết nổi

“GOOD CODE-WHAT?”

- Cấu trúc mã nguồn dễ đọc, dễ hiểu

- Luồng logic được hiển thị rõ ràng qua mã nguồn

- Các thành phần quan trọng đều được mô tả bằng tài liệu đi kèm (docs)

“GOOD CODE - CLEAN CODE”

- Sẵn sàng đáp ứng khi có thay đổi yêu cầu đề bài

- Tương đối dễ dàng để đưa các “nguồn lực” mới vào dự án

- Kiểm soát tiến độ dự án thông qua các kết quả Unit test từng thành phần

- Phát hiện và giải quyết vấn đề hiệu quả hơn

“CLEAN CODE” in Python

1. NAMING
2. Function
3. Logic structure
4. Comments

PEP 8 -- Style Guide for Python Code

NAMING - Name

Name (variable – biến):

- Là danh từ
- Có thể phát âm dễ dàng
- Có ý nghĩa cụ thể, gợi nhớ đến thông tin cần lưu trữ
- Nên: Sử dụng chữ viết thường, giữa các từ là dấu _

PEP 8 -- Style Guide for Python Code--Naming Conventions

NAMING

Bad code

```
ymdstr = datetime.date.today().strftime("%y-%m-%d")
```

Good code

```
current_date = datetime.date.today().strftime("%y-%m-%d")
```

NAMING - Name

Hạn chế sử dụng “magic_number”:

- Thuận tiện cho việc tìm kiếm
- Tạo logic dễ hiểu cho code

NAMING

Bad code

```
time.sleep(86400)
```

Good code

```
SECONDS_IN_A_DAY = 60 * 60 * 24
time.sleep(SECONDS_IN_A_DAY)
```

NAMING - Name

Tường minh tốt hơn là ẩn:

- Đừng tra tấn “não” đồng nghiệp và “trí nhớ” của chúng ta
- Tạo logic dễ hiểu cho code

Explicit is better than implicit

Bad code

```
seq = ('Austin', 'New York', 'San Francisco')
for item in seq:
    do_stuff()
    do_some_other_stuff()
    # ...
    # Wait, what's `item` for again?
    dispatch(item)
```

Good code

```
locations = ('Austin', 'New York', 'San Francisco')
for location in locations:
do_stuff()
do_some_other_stuff()
# ...
dispatch(location)
```

NAMING-Name

- Đặt tên đầy đủ, đừng thêm các ngữ cảnh không cần thiết

Bad code

```
car = {
        car_make: 'Honda',
        car_model: 'Accord',
        car_color: 'Blue’
        };
print(car['car_color'])
```

Good code

```
car = {
        make: 'Honda',
        model: 'Accord',
        color: 'Blue'
        };
print(my_car['color'])
```

NAMING-Name – Tip & trick

Có thể bạn đã biết _ là “temporary name” được gán giá trị gần nhất đưa vào bộ nhớ.

Underscore _ is considered as "I don't Care" or "Throwaway" variable in Python

Có thể sử dụng _ thay vì thực hiện định nghĩa một tên mới.

NAMING-Name – Tip & trick

Good code

```
# Ignore a value when unpacking
x, _, y = (1, 2, 3)

# Ignore the index
for _ in range(100):
print(".")
```

“NAMING” - Function

FUNCTION:

- Bắt đầu là các cụm động từ theo sau là các danh từ
- Mô tả rõ ràng hành động muốn thực hiện theo đúng logic nghiệp vụ
- Nên: Sử dụng chữ viết thường, giữa các từ sử dụng dấu _

Bad code

```
def token(token):
    """Check token is valid
    """
    to_do()

def admin(user_id):
    """Check user_id is admin
    """
    to_do()
```

Good code

```
def is_valid_token(token):
to_do()

def is_admin(user_id):
to_do()
```

“NAMING” - Function

FUNCTION & ALL:

- Hạn chế viết tắt
- Nên dùng tiếng Anh, hạn chế sử dụng tiếng việt không dấu

Bad code

```
def get_artice_sjid():
    '''Get subject id of a artice
    '''
    to_do()

def them_ly_do_uong(bia_ha_noi):
    to_do()
```

Good code

```
def get_artice_subject_id():
to_do()

def get_more_drink(ha_noi_beer):
to_do()
```

“NAMING” - CLASS

CLASS:

- Là danh từ
- Đưa thông tin liên quan nhất đến thực thể muốn mô tả
- Nên: Sử dụng kiểu CamelCase or CapWords

Bad code

```
class animal:
pass

class cleancode:
pass
```

Good code

```
class Animal:
pass

class CleanCode:
pass
```

“NAMING” – Module & Package

Module & Package:

- Là danh từ
- Càng ngắn càng tốt
- Viết bằng chữ thường

Module: Sử dụng _ nếu có nhiều từ
Package: Nếu có nhiều từ, viết các từ liền nhau

Package and Module Names

Function

Một chương trình nên chia nhỏ thành nhiều function, mỗi function thực hiện chức năng riêng

→ Tốt cho việc đọc/hiểu logic bài toán

→ Cài đặt unit test

Function

Lý tưởng nhất với function:

- Mỗi function chỉ nên giải quyết một vấn đề

Bad code

```
def send_email_to_clients(client_emails):
    for client_email in client_emails:
        client = db.get_client_by_email(client_email)
        if client.is_actiave():
            send_email(client_email)
```

Not bad code

```
def is_active_client(client_email):
    return db.get_client_by_email(client_email).is_actiave()


def send_email_to_clients(client_emails):
    for client_email in client_emails:
        if is_active_client(client_email):
            send_email(client_email)
```

Function

Lý tưởng nhất với function:

- Số lượng đối số truyền vào (arguments) là từ 0 - 3

Bad code

```
def create_post(title, subject, body, image_url):
    do_some_thing()
```

Good code

```
post_data = {
            "title": title,
            "subject": subject,
            "body": body,
            "image_url": image_url
        }

def create_post(post_data):
    do_some_thing()
```

Function

Lý tưởng nhất với function:

- Số dòng trong một function ~ 30 dòng

Note: Nếu nhiều hơn 30 dòng → Xem lại logic xử lý để tách nhỏ function ra nữa.

Good code

```
def is_active_client(client_email):
    return db.get_client_by_email(client_email).is_actiave()


def get_active_client_emails(client_emails):
    return [client_email for client_email in client_emails
            if is_active_client(client_email)]


def send_email_to_clients(client_emails):
    for client_email in get_active_client_emails(client_emails):
        send_email(client_email)
```

Dữ liệu trả về đồng nhất về kiểu dữ liệu

Bad code

```
def get_something():
    try:
        do_some_thing()
        return 100
    except Exception as ex:
        return ex.__str__()
```

Good code

```
def get_something():
    try:
        do_some_thing()
        return {"status":"OK",
                "return": 100}
    except Exception as ex:
        return {"status":"NOK",
                "return": ex.__str__()}
```

Trả về một dictionary thay vì một tuple ?

Bad code

```
def sell():
    return ("beer", "330ml", 24)
```

Good code

```
def sell():
    return {"name": "beer", "type": "330ml", "number": number}
```

Tránh sử dụng [] hoặc {} là tham số mặc định

Bad code

```
def append_item(item, list_item=[]):
    list_item.append(item)
    return list_item
```

Good code

```
def append_item(item, list_item=None):
    if not list_item:
        list_item = []
    list_item.append(item)
    return list_item
```

Gọi function, thêm tên cho các tham số truyền vào

Bad code

```
def buy_something(name, type, number):
    pass

buy_something("beer", "330ml", 24)
```

Good code

```
def buy_something(name, type, number):
    pass

buy_something(name="beer", type="330ml", number=24)
```

Logic structure

Giảm thiểu việc “chồng tầng” trong code

Task
Write a program that prints the integers from **1** to **100** (inclusive).

But:

- for multiples of three, print **Fizz** (instead of the number)
- for multiples of five, print **Buzz** (instead of the number)
- for multiples of both three and five, print **FizzBuzz** (instead of the number)

The *FizzBuzz* problem was presented as the lowest level of comprehension required to illustrate adequacy.

Bad code

```
def print_fizz_buzz():
    for number in range(1, 101):
        if not number % 3:
            if not number % 5:
                print("FizzBuzz")
            else:
                print("Fizz")
        elif not number % 5:
            print("Buzz")
        else:
            print(number)
```

1

2

3

Not bad code

```
def print_fizz_buzz():
    for number in range(1, 101):
        if not number % 3 and not number % 5:
            print("FizzBuzz")
        elif not number % 3:
            print("Fizz")
        elif not number % 5:
            print("Buzz")
        else:
            print(number)
```

Good code

```
print("\n".join("Fizz"*(not number % 3) + "Buzz"*(not number % 5)
                or str(number)
                for number in range(1,101)))
```

Logic structure

Sử dụng iterable trong câu lệnh if

Bad code

```
if language == 'Python' or language == 'Go' or language == 'Ruby':
    pass
```

Good code

```
if language in ('Python', 'Go', 'Ruby'):
    pass
```

Logic structure

Nên tách code thành các function để logic trong sáng

Bad code

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

Good code

```
def is_leap_year(year):
    if year % 400 == 0 or (year % 4 == 0 and year % 100):
        return True
    return False

def get_day_of_year(year):
    days = 365
    if is_leap_year(year):
        days += 1
    return days
```

Logic structure

DRY principle: Don’t Repeat Yourself!

Duplicated code: Các đoạn mã thực hiện cùng một chức năng, nhưng xuất hiện lặp đi lặp lại nhiều lần.

Nguyên nhân**:**

- Team work
- Thói quen copy & paste

Hậu quả:

- Khó khăn trong việc bảo trì code
- Thay đổi nghiệp vụ dẫn đến thay đổi nhiều chỗ
- Khó thích nghi khi có nhân sự mới

Bad code

```
def get_resource_by_name(resource_name):
    url = "{}/resource?name={}".format(Settings.api_url, resource_name)
    header = {"Authorization": "Bearer " + self.token["access_token"]}
    session = requests.session()
    session.headers.update(header)
    response = session.get(url)
    return response

def get_resource_by_id(resource_id):
    url = "{}/resource?id={}".format(Settings.api_url, resource_id)
    header = {"Authorization": "Bearer " + self.token["access_token"]}
    session = requests.session()
    session.headers.update(header)
    response = session.get(url)
    return response
```

Good code

```
def get_response(url):
    header = {"Authorization": "Bearer " + self.token["access_token"]}
    session = requests.session()
    session.headers.update(header)
    return session.get(url)

def get_resource_by_name(resource_name):
    url = "{}/resource?name={}".format(Settings.api_url, resource_name)
    return get_response(url)

def get_resource_by_id(resource_id):
    url = "{}/resource?id={}".format(Settings.api_url, resource_id)
    return get_response(url)
```

Logic structure

DRY principle: Don’t Repeat Yourself!

Cách giải quyết:

- Thường xuyên review code & thực hiện refactor code

- Tập thói quen … thấy “ngứa mắt” là phải làm luôn, việc hôm nay không để ngày mai

TIP & TRICK

Tránh viết code trên một dòng với dấu :

Bad code

```
if is_active(user_name): print("{} is actived".format(user_name))
```

Good code

```
if is_active(user_name):
    print("{} is actived".format(user_name))
```

TIP & TRICK

Trong một dự án, dùng đồng nhất “ hoặc ‘ cho biểu diễn string

Bad code

```
my_data = {
           "double": "use double quote",
           'single': 'use single quote'
          }
```

Good code

```
my_data = {
           "double": "use double quote",
           "single": "use single quote"
           }
```

TIP & TRICK

Sử dụng các thể hiện của kiểu boolean

Bad code

```
if is_valid == True:
    pass

if value is None:
    pass

if len(items) == 0:
    is_sold_out = True
else:
    is_sold_out = False

if not is_not_valid():
    pass
```

Good code

```
# Compare implicitly
if is_valid:
    pass

if not value:
    pass

# Assign implicitly
is_sold_out = len(items) == 0

# Avoid negative value
if is_valid():
    pass
```

TIP & TRICK

Dùng else trong vòng lặp for

Bad code

```
def find_index(target, numbers):
    found = False
    for i, value in enumerate(numbers):
        if value == target:
            found = True
            break
    if not found:
        return -1
    return i
```

Good code

```
def find_index(target, numbers):
    for index, value in enumerate(numbers):
        if value == target:
            break
    else:
        return -1
    return index
```

COMMENTS

- Comment tốt nhất chính là tên của function, class, variable

- Chỉ nên sử dụng comment cho để giải thích cho các giả định không rõ ràng bằng cách đọc mã nguồn. (chưa chắc chắn có hoạt động tốt hay không).

- Sử dụng document để mô tả cho các ý tưởng được cài đặt trong function, module

- Sử dụng prefix # TODO:, # FIXME:, # WARN:,… để lưu lại “nhật ký” trong code

- Tránh sử dụng kiểu comment liên tục trong code (zombie comments)

COMMENTS

```python
def find_index(target, numbers):
    # Duyệt từng phần tử trong list để lấy ra index và giá trị
    for index, value in enumerate(numbers):
        # Kiểm tra giá trị có bằng số đầu vào không
        if value == target:
            break  # Nếu có dừng vòng lặp
    else: # Nếu vòng lặp không bị dừng
        return -1 # Trả về -1 tương ứng với không tìm thấy index
    return index
```

zombie comments

COMMENTS

```
def find_index(target, numbers):
    """Tìm vị trí của một giá trị trong một list dữ liệu
    :param tartget: Giá trị cần tìm
    :param number: list các dữ liệu
    :return: - Nếu tìm thấy trả về vị trí của giá trị trong list
             - Trả về -1 nếu không tìm thấy
    :rtype: integer
    """
    for index, value in enumerate(numbers):
        if value == target:
            break
    else:
        return -1
    return index
```

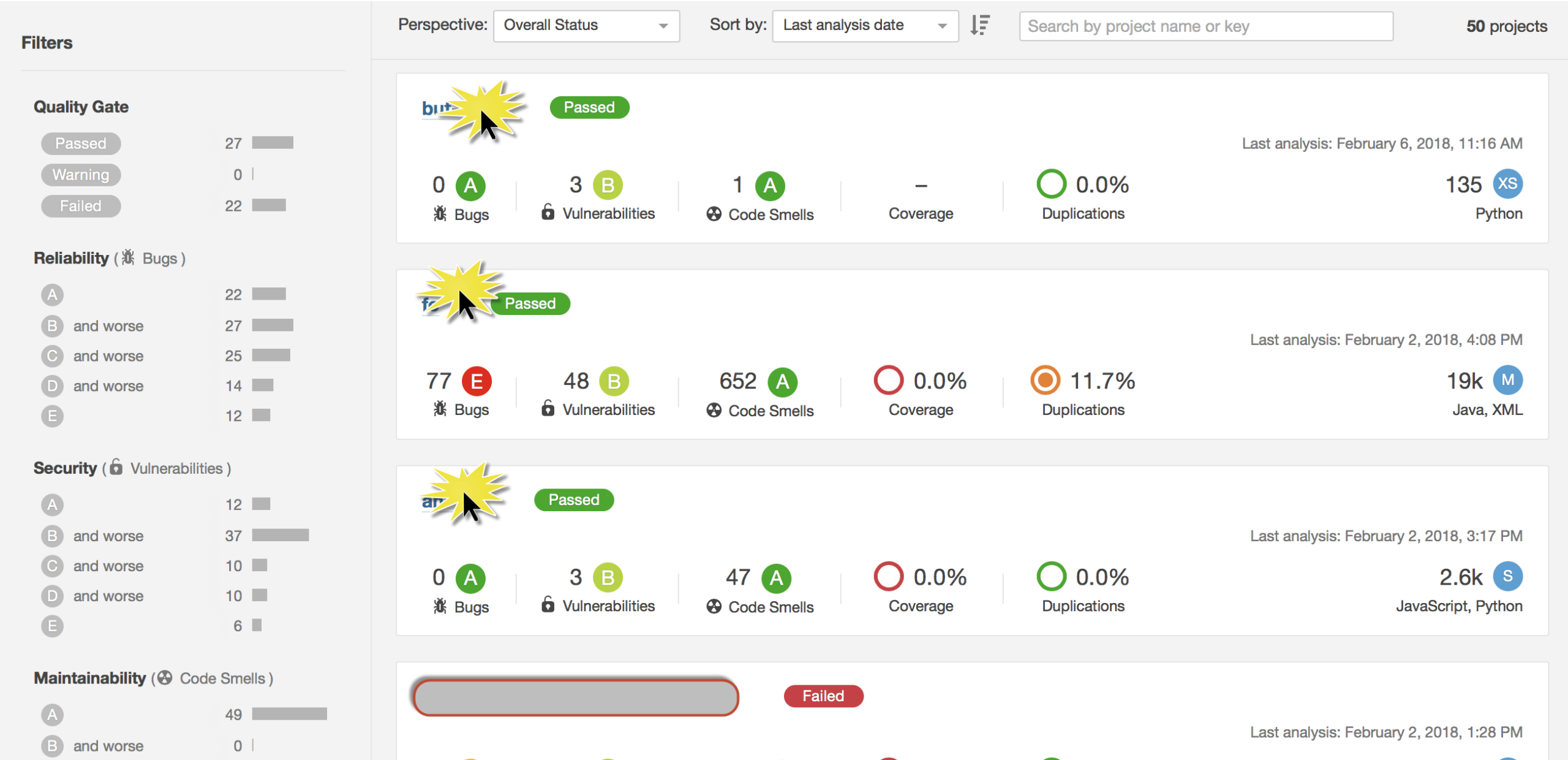
“Clean code” tools for devs

Sonarqube:

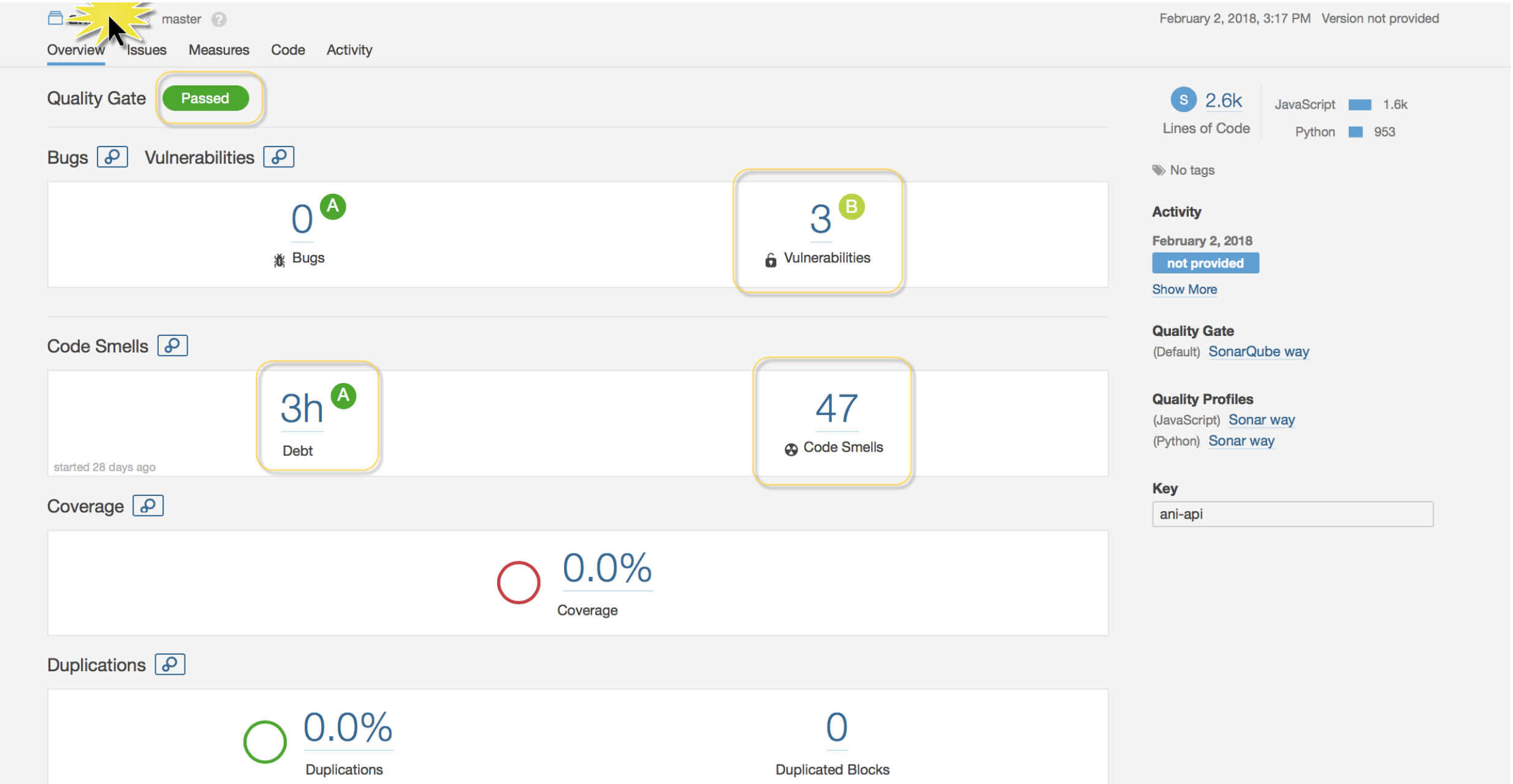
The leading product for CONTINUOUS **CODE QUALITY**

https://www.sonarqube.org/

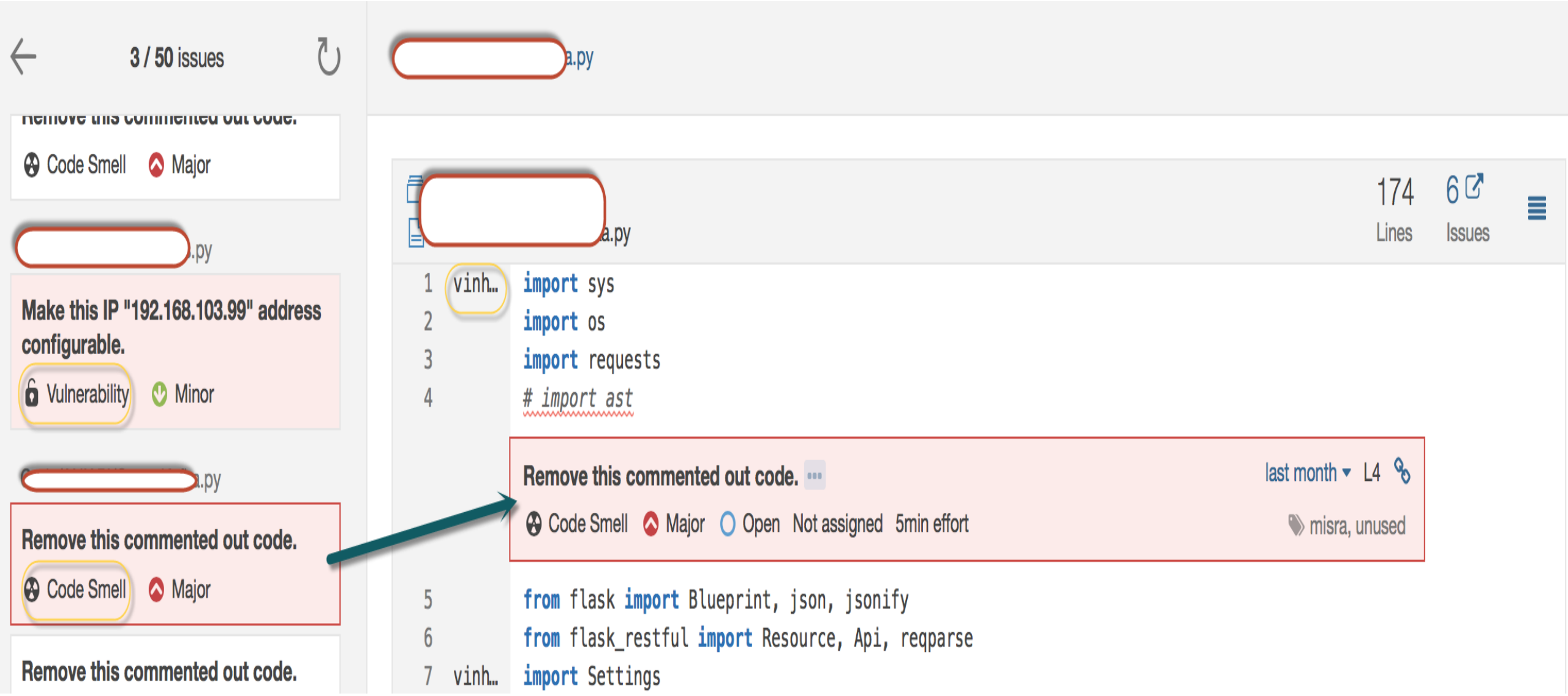
“Clean code” tools for devs



“Clean code” tools for devs



“Clean code” tools for devs



“Clean code” tools for devs

[redacted].py

Rename this local variable "remittanceVSD" to match the regular expression ^[_a-z][a-z0-9_]*$. 28 days ago L44

Code Smell Minor Open Not assigned 2min effort convention

Rename this local variable "remittanceVSD_status" to match the regular expression ^[_a-z][a-z0-9_]*$. 28 days ago L45

Code Smell Minor Open Not assigned 2min effort convention

Rename this local variable "remittanceVSD_id" to match the regular expression ^[_a-z][a-z0-9_]*$. 28 days ago L47

Code Smell Minor Open Not assigned 2min effort convention

Rename this local variable "remittanceVSD_code" to match the regular expression ^[_a-z][a-z0-9_]*$. 28 days ago L48

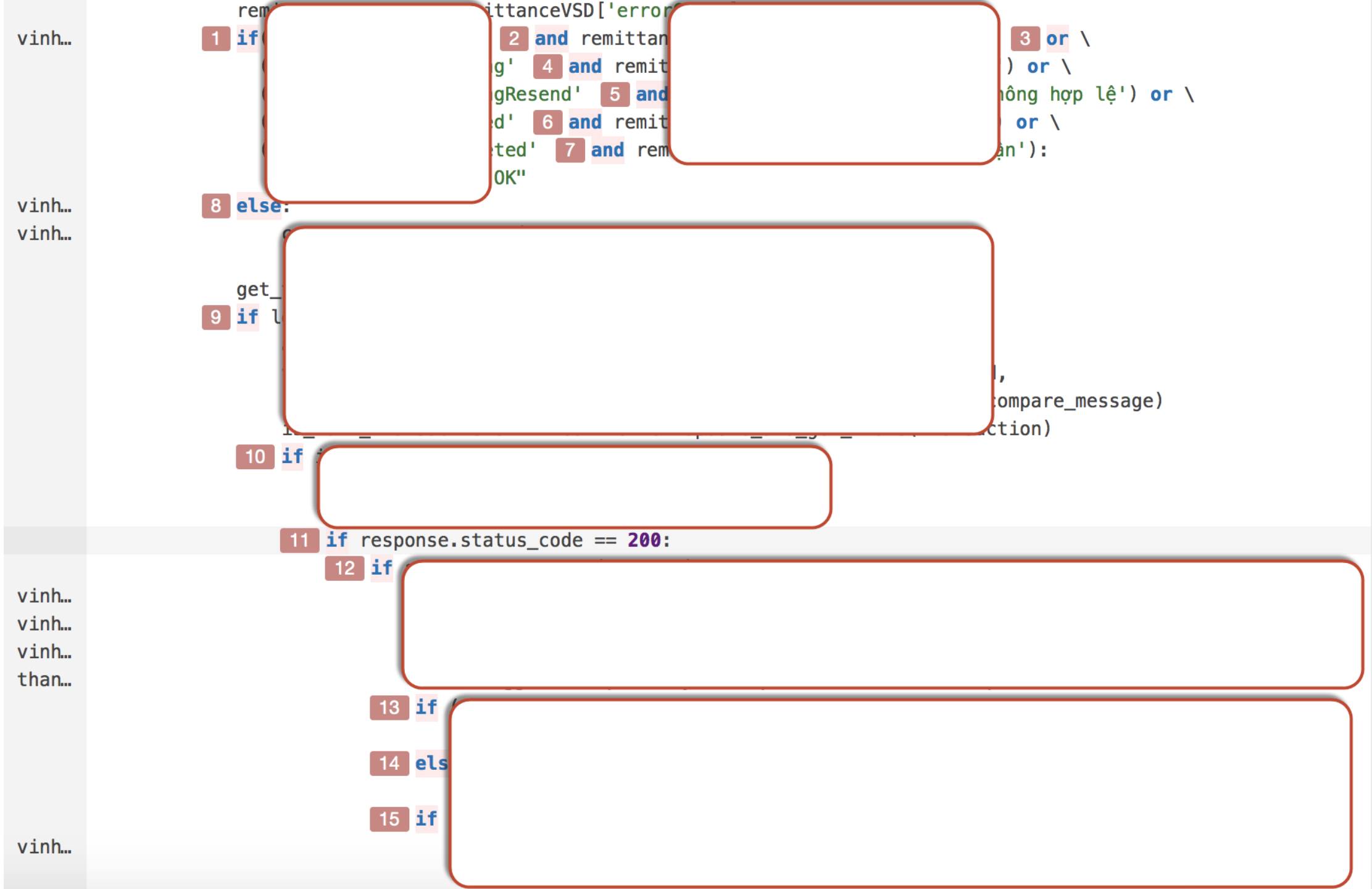
Code Smell Minor Open Not assigned 2min effort convention

“Clean code” tools for devs

Refactor this function to reduce its Cognitive Complexity from 32 to the 15 allowed.

Code Smell Critical +18

1	+1
2	+1
3	+1
4	+1
5	+1
6	+1
7	+1
8	+1
9	+1
10	+2 (incl 1 for nesting)
11	+3 (incl 2 for nesting)
12	+4 (incl 3 for nesting)
13	+5 (incl 4 for nesting)
14	+1
15	+5 (incl 4 for nesting)
16	+1
17	+1
18	+1



Tham khảo

Clean Code: A Handbook of Agile Software Craftsmanship

PEP 8 -- Style Guide for Python Code

Code Quality Assistance Tips and Tricks, or How to make your code look pretty?

END!

Cảm ơn các bạn đã lắng nghe