Python Anti-Patterns

What we should **NOT** do in our code

July, 2021

Vinicius Gubiani Ferreira
 QA / Edge Services SWE
A little bit about myself

Vinícius Gubiani Ferreira
Summary

- Motivation
- Generic anti-patterns
- The Little Book of Python Anti-Patterns
1

Motivation
Help you reach the **next level**
2

Generic anti-patterns
(apply to any language)
What exactly is a (design) pattern?

- Common solution to recurring problem;
- Happens at least 3 times, with different teams, without contact among them;
- Ends up being widely adopted;
- Convergence methodology;
- Reliable and effective;
The AntiPattern on the other hand

- Looks great when we start ...
- until it’s not anymore!
- Often causes more damage than the original problem itself;

...but further on leads you into a maze filled with monsters.

A anti-pattern it’s a solution that initially look like a attractive road lined with flowers...
Usually belong to 1 of 3 large categories

- Development;
- Architecture;
- Project Management;
A few AntiPatterns

Too many patterns to discuss in a 30 minute presentation, so we'll only discuss some of them, such as

- Boat anchor;
- Spaghetti code;
- God object;
- Vendor Lock-in;
- Cargo cult programming;
- Premature optimization;
- Magic numbers;
- Gold plating;
The Little Book of Python Anti-Patterns
Categories of the book

- Correctness
- Maintainability
- Performance
- Security
- Readability
- Migration
No exception type specified

**Bad Code sample**

```
1  try:
2    do_something()
3  except:
4    pass
```

**Good Code sample**

```
1  try:
2    do_something()
3  except ValueError:
4    logging.exception('Caught error')
5    pass
```
Ignore context managers to handle files

**Bad Code sample**

```python
1  f = open("file.txt", "r")
2  content = f.read()
3  1 / 0
4  f.close()
```

**Good Code sample**

```python
1  with open("file.txt", "r") as f:
2     content = f.read()
3  1 / 0
```
Return more than one variable type in function calls

**Bad Code sample**

```python
1 def get_secret_code(password):
2     if password != "bicycle":
3         return None
4     return "42"
```

**Good Code sample**

```python
1 def get_secret_code(password):
2     if password != "bicycle":
3         raise ValueError
4     return "42"
```
Accessing a protected member from outside the class

**Bad Code sample**

```python
    class Rectangle(object):
        def __init__(self, width, height):
            self._width = width
            self._height = heigh

        tr = Rectangle(5, 6)
        memberprint(
        "Width:{:d}".format(r._width)
    )
```

**Good Code sample**

```
    Make attributes part of the public interface of the class (getters and setters).
```
Assigning to built-in function

- **Very Bad** Code sample
  
  ```python
  1  list = [1, 2, 3]
  2  cars = list()
  ```

- **Good Code sample**
  
  ```python
  1  numbers = [1, 2, 3]
  2  cars = list()
  ```
Using tabs or mixing tabs with spaces

<table>
<thead>
<tr>
<th>Bad sample</th>
<th>Good Code sample</th>
</tr>
</thead>
<tbody>
<tr>
<td># Indentation with Tabs</td>
<td># Indentation with 4 spaces</td>
</tr>
</tbody>
</table>
Developers who use spaces make more money!
Python anti-patterns

Source: Stackoverflow

Use
- Spaces
- Tabs
- Both

Median annual salary (US Dollars)

Numbers of years coded as part of their job

Source: Stackoverflow
Not using else where appropriate in a loop

**Bad Code sample**

```python
my_list = [1, 2, 3]
magic_number = 4
found = False

for number in my_list:
    if number == magic_number:
        found = True
        print("Magic number found")
        break

if not found:
    print("Magic number not found")
```

**Good Code sample**

```python
my_list = [1, 2, 3]
magic_number = 4

for number in my_list:
    if number == magic_number:
        found = True
        print("Magic number found")
        break
    else:
        print("Magic number not found")
```
Not using `get()` to return a default value from a dict

**Bad Code sample**

```python
1 dictionary = {"message": "Hello!"}
2 data = ""
3 if "message" in dictionary:
4     data = dictionary["message"]
5 print(data)
```

**Good Code sample**

```python
1 dictionary = {"message": "Hello!"}
2 data = dictionary.get("message", "")
3 print(data)
```
Using wildcard imports

**Bad sample**

```
1 from math import *
```

**Good Code sample**

```
1 from math import ceil
```
Using the global statement

**Bad Code sample**

```python
1 WIDTH = 0
2 HEIGHT = 0
3
4 def area(w, h):
5     global WIDTH
6     global HEIGHT
7     WIDTH = w
8     HEIGHT = h
9     return WIDTH * HEIGHT
```

**Good Code sample**

```python
1 class Rectangle:
2     def __init__(self, width, height):
3         self.width = width
4         self.height = height
5     def area(self):
6         return self.width * self.height
```
Using single letter to name your variables

**Very Bad** sample

```python
1 l = [1, 2, 3, 4, 5]
```

**Good sample**

```python
1 car_ids = [1, 2, 3, 4, 5];
```
Comparing things to True the wrong way

**Bad Code sample**

```python
flag = True
if flag == True:
    print("This works!")
```

**Good Code sample**

```python
flag = True
if flag:
    print("This works!")
flag = True
if flag is True:
    print("This works!")
```
>>> 1 is True
False
>>> id(1)
4495956272

>>> id(True)

>>> 1 == True
True
>>> id(1.0)
4497188208

>>> 1.0 == True
True
>>> id(-1)

>>> -1 == True
False
>>> id(-1)
4495956208

>>> True is True
True
Using type() to compare types

**Bad Code sample**

```python
1  c = Circle(2)
2  r = Rectangle(3, 4)
3  if type(r) is not type(c):
4      print("object types do not match")
```

**Good Code sample**

```python
1  r = Rectangle(3, 4)
2  if isinstance(r, types.ListType):
3      print("object r is a list")
```
Not using named tuples in function return

**Bad Code sample**

```python
1  def get_name():
2      return "Richard", "Jones"
3  name = get_name()
4  # no idea what these indexes map to!
5  print(name[0], name[1])
```

**Good Code sample**

```python
1  from collections import namedtuple
2  def get_name():
3      name = namedtuple(  
4          "name", ["first", "last"]
5      )
6      return name("Richard", "Jones")
7  name = get_name()
8  print(name.first, name.last)
```
References

- AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis
- https://martinfowler.com/bliki/AntiPattern.html
- https://sourcemaking.com/antipatterns
- https://en.wikipedia.org/wiki/Anti-pattern
- https://stackoverflow.blog/2017/06/15/developers-use-spaces-make-money-use-tabs/
- https://realpython.com/the-most-diabolical-python-antipattern/
- https://deepsouce.io/blog/8-new-python-antipatterns/
Thank you

Obrigado
Gracias
Vielen Dank
Спасибо
谢谢啦
ありがとう
Have a question?

Please contact me

овая губерния

vinigfer

vinigfer

vinicius-gubiani-ferreira

vini.g.fer@gmail.com