Clean code always looks like it was written by someone who cares.

– Michael Feathers [1]

Before talking about clean code we should ask an important question.

What is clean code?

This quote from Bjarne Stroustrup, inventor of the C++ programming language clearly explains what clean code means:

"I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well."

From the quote we can pick some of the qualities of clean code:

1. Clean code is focused.

Each function, class, or module should do one thing and do it well.

2. Clean code is easy to read.

According to Grady Booch, author of Object-Oriented Analysis and Design with Applications: clean code reads like well-written prose.

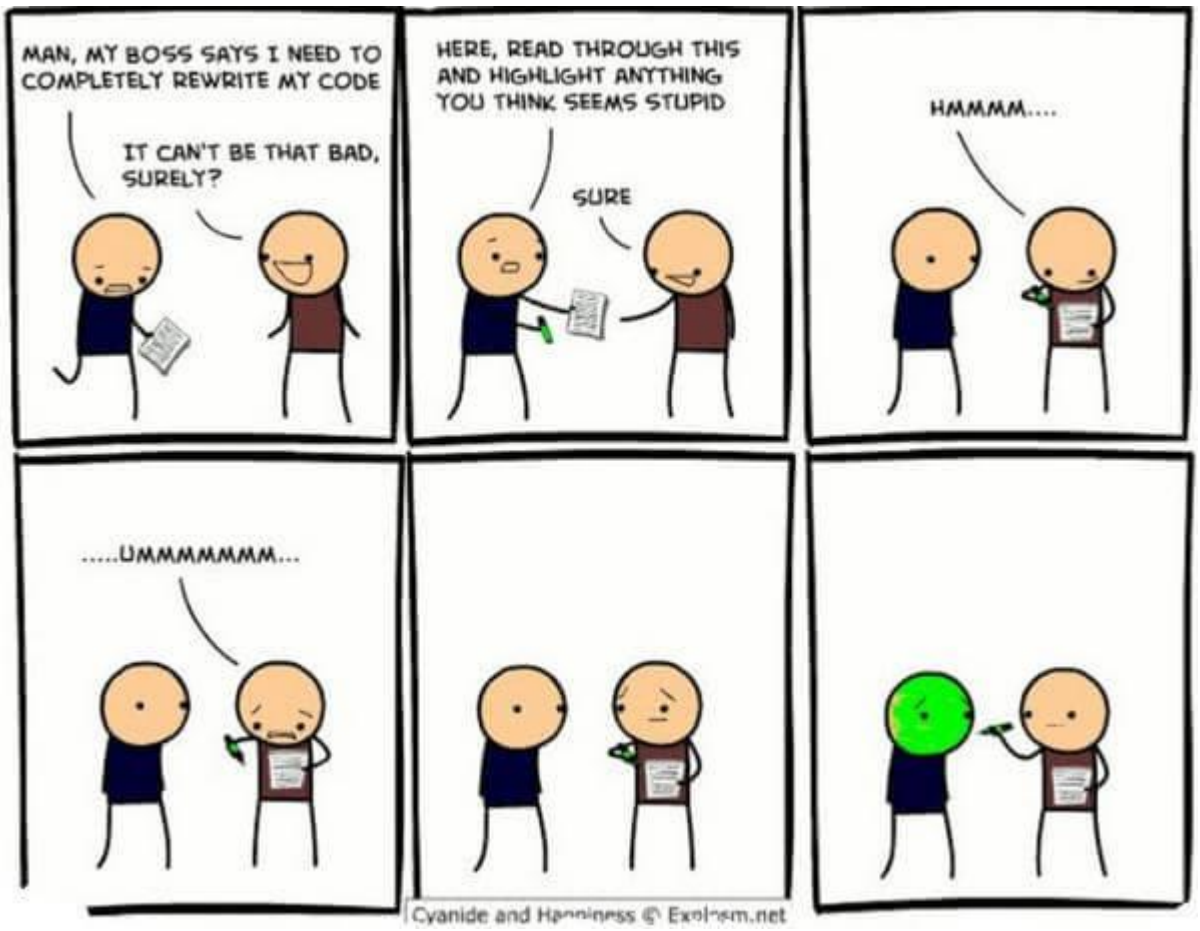
3. Clean code is easy to debug.

4. Clean code is easy to maintain.

That is it can easily be read and enhanced by other developers.

5. Clean code is highly performant.

Well, a developer is free to write their code however they please because there is no fixed or binding rule to compel him/her to write clean code. However, bad code can lead to technical debt which can have severe consequences on the company. And this, therefore, is the caveat for writing clean code.



Patterns for writing clean code in Python:

Naming convention:

Naming conventions is one of the most useful and important aspects of writing clean code. When naming variables, functions, classes, etc, use meaningful names that are intention-revealing. And this means we would favor long descriptive names over short ambiguous names.

1. Use long descriptive names that are easy to read. And this will remove the need for writing unnecessary comments as seen below:

```
# Not recommended
# The au variable is the number of active users
au = 105

# Recommended
total_active_users = 105
```

2. Use descriptive intention revealing names. Other developers should be able to figure out what your variable stores from the name. In a nutshell, your code should be easy to read and reason about.

```
# Not recommended

c = [“UK”, “USA”, “UAE”]
        for x in c:
            print(x)

# Recommended

countries = [“UK”, “USA”, “UAE”]
            for country in countries:
                print(country)
```

3. Avoid using ambiguous shorthand. A variable should have a long descriptive name rather than a short confusing name.

```
# Not recommended

fn = 'John'
Ln = ‘Doe’
cre_tmstp = 1621535852

# Recommended

first_name = ‘JOhn’
Las_name = ‘Doe’
creation_timestamp = 1621535852
```

4. Always use the same vocabulary. Be consistent with your naming convention. Maintaining a consistent naming convention is important to eliminate confusion when other developers work on your code. And this applies to naming variables, files, functions, and even directory structures.

```
# Not recommended
client_first_name = ‘John’
customer_last_name = ‘Doe;
```

```
# Recommended
client_first_name = ‘John’
client_last_name = ‘Doe’

Also, consider this example:
#bad code
def fetch_clients(response, variable):
    # do something
    pass

def fetch_posts(res, var):
    # do something
    pass

# Recommended
def fetch_clients(response, variable):
    # do something
    pass

def fetch_posts(response, variable):
    # do something
    pass
```

5. Start tracking codebase issues in your editor.
A major component of keeping your python codebase clean is making it easy for engineers to track and see issues in the code itself.

6. Don’t use magic numbers. Magic numbers are numbers with special, hardcoded semantics that appear in code but do not have any meaning or explanation. Usually, these numbers appear as literals in more than one location in our code.

```
import random

# Not recommended
def roll_dice():
    return random.randint(0, 4)  # What is 4 supposed to represent?
```

```
# Recommended
DICE_SIDES = 4

def roll_dice():
    return random.randint(0, DICE_SIDES)
```

Functions

7. Be consistent with your function naming convention.

As seen with variables above, stick to a naming convention when naming functions. Using different naming conventions would confuse other developers.

```
# Not recommended

def get_users():
    # do something
    Pass

def fetch_user(id):
    # do something
    Pass

def get_posts():
    # do something
    Pass

def fetch_post(id):
    # do something
    pass


# Recommended

def fetch_users():
    # do something
    Pass

def fetch_user(id):
    # do something
    Pass
```

```
def fetch_posts():
   # do something
   Pass

def fetch_post(id):
   # do something
   pass
```

8. Don’t Repeat Yourself (DRY)

This principle suggests that code should not have unnecessary duplication. Instead, it should be organized in a way that avoids redundancy and makes it easy to maintain. For example, instead of writing the same calculation in multiple places in the code, create a function that performs the calculation and call that function from the different places where the calculation is needed.

```
total = 0
for i in len(prices):
   total += prices[i]

print(total)
for i in len(prices):
   total += prices[i]

print(total)

// good example
def calculateTotal(prices) {
   total = 0
     for i in len(prices):
        total += prices[i]
    return total
}
print(calculateTotal(prices))
print(calculateTotal(prices))
```

9. Functions should do one thing and do it well. Write short and simple functions that perform a single task. A good rule of thumb to note is that if your function name contains “and” you may need to split it into two functions.

```
# Not recommended
def fetch_and_display_users():
    users = [] # result from some api call
        for user in users:
            print(user)

# Recommended
def fetch_usersl():
   users = [] # result from some api call
       return users

def display_users(users):
     for user in users:
         print(user)
```

10. Do not use flags or Boolean flags. Boolean flags are variables that hold a boolean value — true or false. These flags are passed to a function and are used by the function to determine its behavior.

```
text = "Python is a simple and elegant programming language."

# Not recommended
def transform_text(text, uppercase):
   if uppercase:
       return text.upper()
   else:
       return text.lower()

uppercase_text = transform_text(text, True)
lowercase_text = transform_text(text, False)


# Recommended
def transform_to_uppercase(text):
   return text.upper()

def transform_to_lowercase(text):
   return text.lower()

uppercase_text = transform_to_uppercase(text)
lowercase_text = transform_to_lowercase(text
```

Classes:

11. Do not add redundant context. This can occur by adding unnecessary variables to variable names when working with classes.

```
# Not recommended

class Person:
    def __init__(self, person_username, person_email, person_phone, person_address):
        self.person_username = person_username
        self.person_email = person_email
        self.person_phone = person_phone
        self.person_address = person_address

# Recommended
class Person:
    def __init__(self, username, email, phone, address):
        self.username = username
        self.email = email
        self.phone = phone
        self.address = address
```

In the example above, since we are already inside the Person class, there's no need to add the person_ prefix to every class variable.

Bonus: Modularize your code:

To keep your code organized and maintainable, split your logic into different files or classes called modules. A module in Python is simply a file that ends with the .py extension. And each module should be focused on doing one thing and doing it well.
You can follow object-oriented — OOP principles such as follow basic OOP principles like encapsulation, abstraction, inheritance, and polymorphism.

Writing clean code comes with a lot of advantages, improving your software quality, code maintainability, and eliminating technical debt.