

Clean Code Fundamentals

Form

Pre-work

- Video: <https://cleancoders.com/episode/clean-code-episode-5>
- Exam: <https://cleancoders.com/episode/clean-code-episode-5/exam>

Chapters

Chapter	Time
Overview	00:00:51
Sol's Pyre	00:03:57
Comments	00:11:12
Coding Standards	00:11:32
Comments should be Rare	00:13:00
Comments are Failures	00:15:23
Comments are Lies	00:18:28
Good Comments	00:20:08
Bad Comments	00:22:53
Explanatory Structures	00:31:00

Chapter	Time
Formatting	00:33:23
File Size	00:34:23
Vertical Formatting	00:37:35
Horizontal Formatting	00:41:01
Indentation	00:43:28
Classes	00:45:53
Data Structures	00:53:33
Boundaries	00:59:46
The Impedance Mismatch	01:03:52
Conclusion	01:09:20

Timetable

Activity	Time
Warmup	5 min
Exercise 1	20 min
Exercise 2	20 min
Exercise 3	20 min
Wrap up	5 min

Warmup

- What are your formatting preferences? (e.g. tabs vs spaces, 2 spaces vs 4 spaces, snake_case vs camelCase, etc.)
 - Type in the meeting chat

Exercise 1

- Prompt
 - Compare classes and data structures
- Time limit: 10 minutes

Possible answer

- Classes
 - Cohesive data and functions that operate on that data
 - Private data, public functions
 - Tell, don't ask
 - Polymorphism
 - Protects from new types, exposes to new methods
- Data Structures
 - Cohesive data, simple functions like getters and setters
 - Public data
 - Ask, don't tell
 - Switch statements
 - Protects from new methods, exposes to new types

Exercise 2

- Code-review practice (part 1)
- Use the worksheet
- Go through scenarios 1-3
- Suggest code improvements, and explain why
 - Use the worksheet to record your suggestions
 - Use active voice (e.g. “introduce parameter object” instead of “parameter object should be introduced”)
- Time limit: 15 minutes
- Scenarios adopted from [Eder Diaz blog](#)

Scenario 1

```
1 def can_buy_beer(age, money):
2     if age >= 21 and money >= 20:
3         return True
4
5     return False
```

Scenario 1 solution

- Introduce constants to clarify the *magic numbers*
- Collapse if-statement when evaluating boolean expressions

```
1 LEGAL_DRINKING_AGE = 21
2 BEER_PRICE = 20
3
4
5 def can_buy_beer(age, money):
6     return age >= LEGAL_DRINKING_AGE and money >= BEER_PRICE
```

Scenario 2

```
1 def should_show_image(item_index, article, show_all_images):
2     return (
3         bool(article.image_url)
4         if item_index in (0, 1, 2)
5         else bool(article.image_url)
6         if show_all_images
7         else False
8     )
```

Ternary operators note:

- C++: condition ? a : b
- Python: a if condition else b

Scenario 2 solution

- Use explanatory variables to clarify sub-expressions
- Remove code duplication

```
1 def should_show_image(item_index, article, show_all_images):
2     is_first_three_items = item_index in (0, 1, 2)
3     has_image = article.image_url is not None
4
5     return (is_first_three_items or show_all_images) and has_image
```

Scenario 3

```
1 import math
2
3
4 def get_area(shape, width, height, radius):
5     if shape == "circle":
6         return math.pi * radius * radius
7     elif shape == "square":
8         return width * width
9     elif shape == "rectangle":
10        return width * height
```

Scenario 3 solution

- Introduce polymorphism to protect from new shapes
- Replace shape names with classes to make it less error-prone

```
1 import abc
2 import math
3
4
5 class Shape(abc.ABC):
6     @abc.abstractmethod
7     def get_area(self):
8         raise NotImplementedError
9
10
11 class Circle(Shape):
12     def __init__(self, radius):
13         self.radius = radius
14
15     def get_area(self):
16         return math.pi * self.radius * self.radius
```

Exercise 3

- Code-review practice (part 2)
- Use the worksheet
- Go through scenarios 4-6
- Suggest code improvements, and explain why
 - Use the worksheet to record your suggestions
 - Use active voice (e.g. “introduce parameter object” instead of “parameter object should be introduced”)
- Time limit: 15 minutes

Scenario 4

```
1 chemical_symbols = {
2     "Sodium": "Na",
3     "Hydrogen": "H",
4     "Helium": "He",
5     "Oxygen": "O",
6 }
7
8
9 def get_symbol(name):
10     symbol = chemical_symbols.get(name)
11     if symbol:
12         return symbol
13
14     print("symbol not found")
15     return "not found"
```

Scenario 4 solution

- Introduce error handling
- Use PEP8 naming conventions

```
1 class SymbolNotFoundError(Exception):
2     pass
3
4
5 CHEMICAL_SYMBOLS = {
6     "Sodium": "Na",
7     "Hydrogen": "H",
8     "Helium": "He",
9     "Oxygen": "O",
10 }
11
12
13 def get_symbol(name):
14     try:
15         return CHEMICAL_SYMBOLS[name]
16     except KeyError:
17         raise SymbolNotFoundError("symbol not found")
```

Scenario 5

```
1 def get_display_image(article, watermark=None):
2     image = None
3
4     if article["image"] and article["displayImage"]:
5         if watermark:
6             image = apply_watermark(article["image"], watermark)
7         else:
8             image = article["image"]
9
10    return image
```

Scenario 5 solution 1

- Use early return to separate main logic from corner cases
- Use separate function for watermark=None argument

```
1 def has_image_and_display_image(article):
2     return article["image"] and article["displayImage"]
3
4
5 def get_display_image_with_watermark(article, watermark):
6     if not has_image_and_display_image(article):
7         return None
8     return apply_watermark(article["image"], watermark)
9
10
11 def get_display_image(article):
12     if not has_image_and_display_image(article):
13         return None
14     return article["image"]
```

Scenario 5 solution 2

- Introduce a class

```
1 class Article:
2     def __init__(self, image, display_image):
3         self.image = image
4         self.display_image = display_image
5
6     def has_image_to_display(self):
7         return self.image and self.display_image
8
9     def get_display_image(self):
10        if not self.has_image_to_display():
11            return None
12        return self.image
13
14    def get_display_image_with_watermark(self, watermark):
15        if not self.has_image_to_display():
16            return None
17        return apply_watermark(self.image, watermark)
```

Scenario 6

```
1  # The code checks if auth is enabled, if it's enforced,
2  # and if user already registered (has email field).
3  if (
4      type(AUTH_ENABLED) == str and
5      AUTH_ENABLED == "true" and # enabled!
6      not skip_authentication and # enforced!!
7      user.email # already registered!!!
8  ):
9      print(f"Welcome back, {user.name}")
```

Scenario 6 solution

- Replace comments with explanatory variables

```
1 is_auth_enabled = AUTH_ENABLED == "true"
2 is_auth_enforced = is_auth_enabled and not skip_authentication
3 is_registered_user = user.email is not None
4
5 if is_auth_enforced and is_registered_user:
6     print(f"Welcome back, {user.name}")
```

Summary

- Comments
- Formatting
- Classes vs data structures
- Abstraction boundaries

What is next?

- Expect an e-mail with instructions for upcoming coding dojo

Final words

Always leave the code better than you found it.
– *The Software Craftsmanship Rule*