Refactoring Signs & Patterns

String Literals & Magic Numbers

```
import tkinter as tk

frame = tk.Frame()
canvas = tk.Canvas(frame, width=480, height=400,
                   sticky="nsew")
```

Use Named Constants

```
import tkinter as tk

CANVAS_HEIGHT = 400
CANVAS_WIDTH = 480

frame = tk.Frame()
canvas = tk.Canvas(frame,
         width=CANVAS_WIDTH, height=CANVAS_HEIGHT,
         sticky=tk.NSEW)
```

Nondescriptive & Vague Names

```
# is the ball on-screen?
check = 0 < ball.x < 480 and 0 < ball.y < 400

if not check:
    ball.remove()
```

Rename Symbol

```
# is the ball on-screen?
is_onscreen = 0 < ball.x < CANVAS_WIDTH and \
              0 < ball.y < CANVAS_HEIGHT

if not is_onscreen:
    ball.remove()
```

Rename Symbol

Assign a more descriptive, meaningful name to a variable, method, class, or package.

Motivation: make code easier to understand. If you find a more descriptive name for a variable, method, class, or package then change it.

Code evolves over time, The purpose of some piece of code may change so the original name isn't quite right.

Mechanics: use an IDE's Refactor -> Rename feature to consistently change the name. Don't use search and replace, which may change unintended matches.

Long Method or Doing More Than One Thing

```
@login_required
def vote(request, question_id: int):
    """Vote for a choice on a poll question."""
    try:
        question = Question.objects.get(id=question_id)
    except Question.DoesNotExist:
        ...
    selected_choice = ...
    # get user's vote or create a new vote
    try:
        vote = Vote.objects.get(user=user,
                                 choice__question=question)
    except Vote.DoesNotExist:
        vote = Vote(user=user)   # make a new Vote
    vote.choice = selected_choice
    vote.save()
```

Extract Method

```
@login_required
def vote(request, question_id: int):
    """Vote for a choice on a poll question."""
    try:
        question = Question.objects.get(id=question_id)
    except Question.DoesNotExist:
        ...
    selected_choice = ...
    # get user's vote or create a new vote
    vote = get_vote_for_user(question=question, user=user)
    vote.choice = selected_choice
    vote.save()
```

Extract Method

Extract a block of code as a separate method.

Motivation:

a) method is long and difficult to understand, or
b) method doing more than one thing, or
c) a block of code can be used by several methods

Mechanics: move the code to a new method.
Any values needed should be passed as parameters.

Example: extract logic for computing movie rental price from long "statement( )" method. (*Movie Rental*)

Local Var Used Only Once

```
class Person:
    def __init__(self, name, national_id):
        self.name = name
        self.national_id = national_id

    def __eq__(self, other):
        if not isinstance(other, Person):
            return False
        matches = (self.name == other.name and
                   self.national_id == other.national_id)
        return matches
```

Inline Temp

```
class Person:
    def __init__(self, name, national_id):
        self.name = name
        self.national_id = national_id

    def __eq__(self, other):
        if not isinstance(other, Person):
            return False
        return (self.name == other.name and
                self.national_id == other.national_id)
```

Inline Temp if it makes the code easier to read.

Inline Temp

You have a local variable that is assigned to and then used only once. The expression is not complicated.

Solution: Put the expression right where it is used, without assigning it to a temp var.

Motivation:
a) assignment to temps makes code harder to read,
b) the assignment to temp is getting in the way of other refactorings.

See Also: Introduce Explanatory Variable which is the opposite of this!

Complex Expression

```
from datetime import datetime

if (13 <= datetime.now().hour <= 16 and
    datetime.now().isoweekday == 2):
    print("Study refactoring")
```

Introduce Explaining Variable

```
from datetime import datetime

is_tuesday = (datetime.now().isoweekday == 2)
isp_lab_time = is_tuesday and (13 <= datetime.now().hour <= 16)

if isp_lab_time:
    print("Study refactoring")
```

Introduce Explaining Variable

A complicated expression makes it hard to understand the intent of the code.

Solution: Assign result of part of the expression to a local variable whose name describes the meaning.

Motivation: clarify the meaning of a complex expression.

Mechanics: let the IDE do it!
Select the code to extract and choose Refactor -> Assign to local variable or Refactor -> Extract local variable.

Move Method

A method uses more members of another class than members of it's own class.

Solution: Move it to the other class.

Motivation: reduces coupling and often makes the code simpler and classes more coherent.

Mechanics: see references.

Example: computing price of a movie rental depends on rental data, not customer info. So move it to the rental class.

Replace Constructor with Creation Method

Some classes have multiple constructors and their purpose is not clear.

Solution: Replace constructor with static method that create objects, use a name that describe intention of the method.

Motivation: makes creating objects easier to understand.

Mechanics: Define a static method (class method) that creates and returns a new object.

You may have several such methods for different cases.

Refactoring Signs

Sign or signal that you should consider refactoring.

Also called "code smells".*

The purpose of refactoring:

- *Make this code easier to read or maintain.*

** I don't like the term "code smells" -- it is subjective, and refactoring signs are objective. Code doesn't have a smell.*

Name some "symptoms" or "signs"

Name some signs that code may need refactoring.

1. Duplicate logic or duplicate code.

2.

3.

4.

5.

6.

List of Symptoms

A good online list is:

`https://blog.codinghorror.com/code-smells/`

Chapter 3 of *Refactoring* book has longer explanation.

https://refactoring.guru also has a good list.

Duplicate Code or Duplicate Logic

The #1 symptom

Solutions:

Extract Method

Pull up Method

Define a method that performs the duplicate code.

Other Symptoms

Long method

Large class - class with many methods and attributes

Incohesive class - class with many weakly related or unrelated responsibilities

Long parameter list - more than 3 parameters

Temporary field - a class has an attribute that is used only rarely, and can easily be recreated as needed.

Data Class

A class that is just a holder for data (like a 'struct' in C).

Doesn't have any responsibilities, just get/set methods.

Solution:

Look at how other classes are using the data class.

You may simplify the code by moving behavior <u>to</u> the data class. Use the Move Method or Extract Method.

Eclipse Show References: Right click on class name and choose References -> Project. Shows all places where this class is used.

Python dataclass

Python 3.7 `dataclass` provides automatic constructor and methods for classes that are intended to be data "containers".

A data class is used as a container for related data, or data + data specific methods.

```
from dataclasses import dataclass

@dataclass
class Coordinate:
    x: float
    y: float
```

Lazy Class

A step above Data Class.

Motivation: A lazy class doesn't do enough to justify its existence.

Solution:

Either give the class something to do (Move Method) or eliminate it.

Speculative Generality

"*I think we might need this in the future*".

Design for change is good.

But if it involves a lot of extra code or classes, be critical.

Symptoms: Abstract classes that don't do anything.
Interfaces with only 1 implementation.

Solution:

Collapse class hierarchy by moving behavior.

Exercise

Find the *refactoring symptoms* in this code.

Suggest refactorings.

https://vivekagarwal.wordpress.com/2008/06/21/code-smelling-exercise/

Resources

Refactoring, 2nd Edn by Martin Fowler (2018). The first 3 chapters cover the fundamentals.

https://refactoring.guru - refactoring symptoms, techniques, and examples

Refactoring Symptoms & Solutions

List of "code smells"

`https://blog.codinghorror.com/code-smells/`

Code Smells Cheat Sheet

`http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf`

and blog post "*Smells to Refactorings*"

`https://www.industriallogic.com/blog/smells-to-refactorings-cheatsheet/`