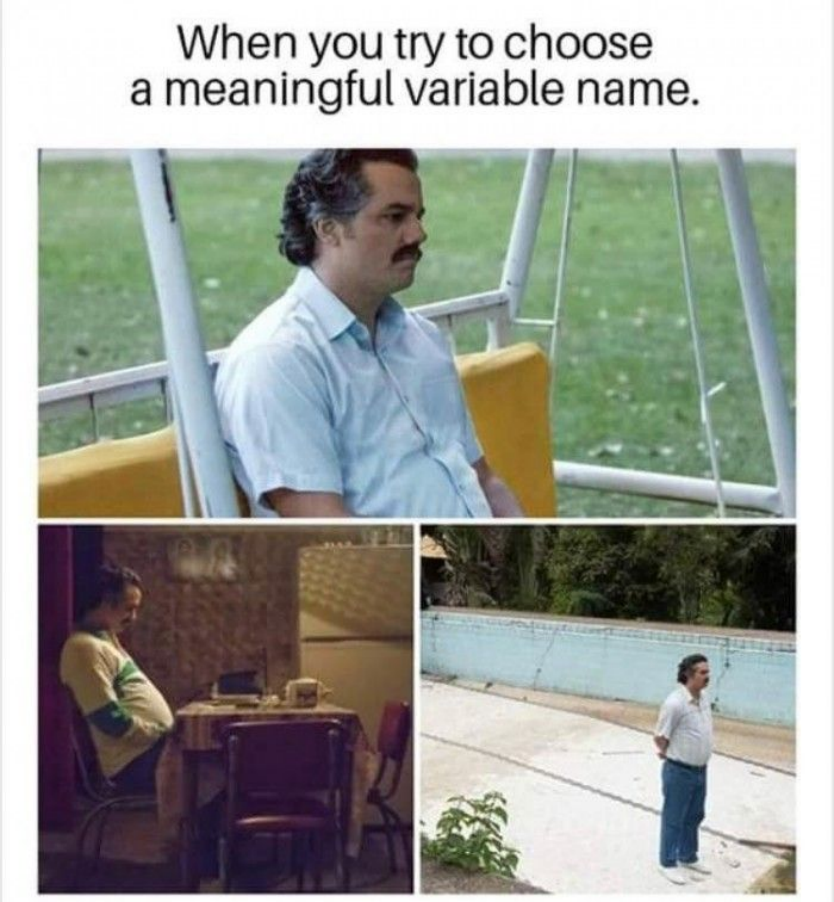
MAKE SCHOOL

Refactoring

Composing functions (1)

SPD 2.31


When you try to choose
a meaningful variable name.

Learning Outcomes

MAKE SCHOOL

By the end of today, you should be able to…

1. Describe refactoring and code smells
2. Compare and contrast different refactoring techniques for composing/extracting functions.
3. Identify code smells and apply refactoring techniques to improve code quality.

"Any fool can write code that a computer can understand.
Good programmers write code that humans can understand."

Martin Fowler

MAKE SCHOOL

What is Refactoring?

What are Code Smells?

Code Smells

- How would you improve this snippet?
  - Name at least 3 improvements.
- What's good about it?

10 min

```python
import math

def printStat():
  gradeList = []
  # Get the inputs from the user
  nStudent = 5
  for _ in range(0, nStudent):
      gradeList.append(int(input('Enter a number: ')))

  # Calculate the mean and standard deviation of the grades
  sum = 0
  for i in range(0, len(gradeList)):
      sum = sum + gradeList[i]
  mean = sum / len(gradeList)
  sd = 0 # standard deviation
  sum_Of_Sqrs = 0
  for i in range(0,len(gradeList)):
      sum_Of_Sqrs += (gradeList[i] - mean) ** 2
  sd = math.sqrt(sum_Of_Sqrs / len(gradeList))

  # print out the mean and standard deviation in a nice format.
  print("The mean is:", mean)
  print('The standard deviation is: ', round(sd, 3))

printStat()
```

Refactoring Definition

Refactoring is the process of restructuring existing computer code without changing its external behavior.

Refactoring usually makes the code:

- Easier to understand
- More maintainable
- More extendable

The Two Hats

MAKE SCHOOL

When you code, you divide your time between two distinct activities:

1. Adding features
2. Refactoring.

When you wear your **feature hat,** you only add features. You shouldn't be refactoring .

When you wear your **refactoring hat**, you only refactor. You shouldn’t be adding features.

Code Smells

MAKE SCHOOL

"**If it stinks, change it.**" Kent Beck

Some Bad Code Smells:

- Duplicated code
- Long function
- Large class
- Long function parameter list
- Too many nested if/elif/else statement
- Confusing names for variables or functions

MAKE SCHOOL

Refactoring Technique: functions of Composing

Extract function

How can we make the following code more modular?

```
# by Martin Fowler.
# Code snippet. Not runnable.
def print_owing():
    outstanding = 0.0
    # print banner
    print("********************")
    print("***Customer owes***")
    print("********************")
    # calculate outstanding
    for order in orders:
        outstanding += order.get_amount()
    # print details
    print("name: " + name)
    print("amount: " + outstanding)
```

Extract function - Refactored

MAKE SCHOOL

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.
# Code snippet. Not runnable.
# Refactored.
def print_banner():
    # print banner
    print("********************")
    print("***Customer owes***")
    print("********************")


def get_outstanding_order_amount():
    outstanding = 0.0
    for order in orders:
        outstanding += order.get_amount()
    return outstanding


def print_owing():
    print_banner()
    outstanding = get_outstanding_order_amount()
    print("amount: " + outstanding)
```

Extract function

Extract function: ***You have a code fragment(s) that can be grouped together. Turn the fragment into a function whose name explains the purpose of the function.***

- ‘Extract function’ refactoring technique is so common and useful that most IDEs have the feature to do that automatically for you.

What are situations in which you benefit from extracting a function?

- Long functions
- A piece of code that is repeated several times.

The Rule of Three:

- The first time you do something, you just do it.
- The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway.
- The third time you do something similar, you refactor.

Extract function

15 mins

Solve Exercise 1: 'Extract function' Technique

MAKE SCHOOL

Break

Get up, stretch, get some water, and relax your mind. 

Inline functions

How can we make the following code easier to understand?

```
# by Martin Fowler.
# Code snippet. Not runnable.
number_of_late_deliveries = 8


def get_rating():
    if is_more_than_five_late_deliveries():
        return "bad"
    else:
        return "good"

def is_more_than_five_late_deliveries():
    return number_of_late_deliveries > 5


print(get_rating())
```

this function does not carry its own weight. Let's get rid of it!

Inline functions - Refactored

MAKE SCHOOL

```
# by Martin Fowler.
# Code snippet. Not runnable.
# Refactored.

number_of_late_deliveries = 8


def get_rating():
    if number_of_late_deliveries > 5:
        return "bad"
    else:
        return "good"


print(get_rating())
```

Inline function

Inline function: **A function's body is just as clear as its name. Put the function's body into the body of its caller and remove the function.**

- 'Inline function' refactoring is the opposite of the 'extracting function' technique.
- *Indirection* is helpful but needless indirection is irritating.

Inline functions

7 mins

MAKE SCHOOL

Solve Exercise 2: 'Inline function' Technique

When extracting functions is hard ...

MAKE SCHOOL

- We learned how to extract a function. Sometimes we have code that makes it harder to extract a function (see the next slide).

Replace Temp with query

MAKE SCHOOL

Try to extract ‘area’ and ‘perimeter’ logic. How would you deal with the ‘height’ and ‘width’ variables?

```
# By Kami Bigdely Nov. 2020
# We have two points of a rectangle. We want to calculate the area and perimeter of the rectangle.
import numpy
class Rectangle:
    def __init__(self, left_bottom_point, right_top_point):
        self.left_bottom_point = left_bottom_point
        self.right_top_point = right_top_point
    def get_rect_properties(self):
        """Return area and perimeter of the rectangle."""
        height = self.right_top_point[1] - self.left_bottom_point[1]
        width = self.right_top_point[0] - self.left_bottom_point[0]
        area = height * width
        perimeter = 2 * (height + width)
        return area, perimeter

# define p1 and p2 points [x,y]
point1 = numpy.array([1,1]) # x = 1, y = 1
point2 = numpy.array([3,3]) # x = 3, y = 3
rect = Rectangle(point1, point2)
area, perimeter = rect.get_rect_properties()
print("Rectangle's area is:", area)
print("Rectangle's perimeter is:", perimeter)
```

When trying to extract ‘area’ and ‘perimeter’, the temporary variables ‘height’ and ‘width’ are getting in the way. You are forced to pass them as arguments to those extracted functions:
get_area(height, width)
get_perimeter(height, width)

If we replace them with a queries (function calls), we can refactor the rest of the code without passing them around.

Replace Temp with query (interim)

MAKE SCHOOL

```
# By Kami Bigdely
# We have two points of a rectangle. We want to calculate the area and perimeter of the rectangle.
# Refactored 1.
import numpy

class Rectangle:
    def __init__(self, left_bottom_point, right_top_point):
        self.left_bottom_point = left_bottom_point
        self.right_top_point = right_top_point

    def get_rect_properties(self):
        """Return area and perimeter of the rectangle."""
        area = self.get_height() * self.get_width()
        perimeter = 2 * (self.get_height() + self.get_width())
        return area, perimeter

    def get_width(self):
        return self.right_top_point[0] - self.left_bottom_point[0]

    def get_height(self):
        return self.right_top_point[1] - self.left_bottom_point[1]

# define p1 and p2 points [x,y]
point1 = numpy.array([1,1]) # x = 1, y = 1
point2 = numpy.array([3,3]) # x = 3, y = 3
rect = Rectangle(point1, point2)
area, perimeter = rect.get_rect_properties()
print("Rectangle's area is:", area)
print("Rectangle's perimeter is:", perimeter)
```

Now that ‘area’ and ‘perimeter’ are not based on temp variables, we can extract functions for them.

These two functions are queries.

Replace Temp with query (Refactored)

MAKE SCHOOL

```
# By Kami Bigdely
# We have two points of a rectangle. We want to calculate the area and perimeter of the rectangle.
# Refactored 2.
import numpy

class Rectangle:
    def __init__(self, left_bottom_point, right_top_point):
        self.left_bottom_point = left_bottom_point
        self.right_top_point = right_top_point

    def get_rect_properties(self):
        """Return area and perimeter of the rectangle."""
        return self.get_area(), self.get_perimeter()

    def get_area(self):
        return self.get_height() * self.get_width()

    def get_perimeter(self):
        return 2 * (self.get_height() + self.get_width())

    def get_width(self):
        return self.right_top_point[0] - self.left_bottom_point[0]

    def get_height(self):
        return self.right_top_point[1] - self.left_bottom_point[1]

# define p1 and p2 points [x,y]
point1 = numpy.array([1,1]) # x = 1, y = 1
point2 = numpy.array([3,3]) # x = 3, y = 3
rect = Rectangle(point1, point2)
area, perimeter = rect.get_rect_properties()
print("Rectangle's area is:", area)
print("Rectangle's perimeter is:", perimeter)
```

Replace Temp with Query

MAKE SCHOOL

You are using a temporary variable to hold the result of an expression.

Extract the expression into a function. *Replace all references to the temp with the new function.* The new function can then be used in other functions.

- Temps tend to encourage longer functions. Replace them with queries. Then shorten the long function by extracting functions out of it.
- From the performance point of view, what's wrong with this refactoring technique? How would you address that?
  - memoization

Replace Temp with query

7 mins

Solve Exercise 3: ' Replace Temp with Query' Technique

Introduce Explaining Variable (aka extract variable)

MAKE SCHOOL

How can we make the following code easier to understand?

```
# by Martin Fowler.
# Introduce Explaining variable (aka extract variable)
platform = "mac pro 2018"
browser = 'safari 8.0'
resize = True
def is_initialized():
   return True
def render_banner():
   if ('MAC' in platform.upper() and
   "SAFARI" in browser.upper() and
   is_initialized()):
       print("Yes!")
       # do something
render_banner()
```

Introduce Explaining Variable (Refactored)

MAKE SCHOOL

```
# by Martin Fowler.
# Refactored.
platform = "mac pro 2018"
browser = 'safari 8.0'
resize = True
def is_initialized():
   return True
def render_banner():
   is_mac_os = 'MAC' in platform.upper()
   is_safari = "SAFARI" in browser.upper()
   if is_mac_os and is_safari and is_initialized():
       print("Yes!")
   # do something
render_banner()
```

Introduce Explaining Variable

MAKE SCHOOL

You have a complicated expression.

Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.

What are situations in which you benefit using 'introduce explaining variable' technique**?**

- With long conditional logic
- Long algorithms in which each step in the computation can be explained with a temp.

Introduce Explaining variable

10 min

MAKE SCHOOL

Solve Exercise 4: Introduce Explaining Variable' Technique

Summary

1. Extract function
2. Inline function
3. Replace temporary variables with query
4. Introduce explaining variable

References and Further Study

1. "Refactoring: Improving the Design of Existing Code" (1st edition) by Martin Fowler
2. https://en.wikipedia.org/wiki/Code_refactoring