NEW! STRONGER THAN DIRT

ENTER.
tech with a heartbeat

{clean code}

CAUTION: EVEN BAD CODE CAN FUNCTION
40 FL OZ (1.25 QT) 1.18L

INTRODUCTION

- Functions are the first line of organization of any program.
- Provide modularity and reusability of code.
- "The first rule is that they should be small. *The second rule is that they should be smaller than that!*"
- Functions should be *transparently obvious* and *should tell a story*.

❗ REMEMBER ❗

Functions should be no bigger than4 or 5 lines.

ENTER.
tech with a heartbeat

KEEPING YOUR FUNCTIONS SMALL

- The code inside `if`, `else` and `while` statements should be a function call;
  - It makes the code inside those statements one line long.
  - It keeps the inclosing code small and serves as documentation because the function called within the block have a nicely descriptive name.

DO ONE THING

- *Functions should do one thing. They should do it well; and they should do it only!*

- Your function is doing one thing if you can’t extract a function out of it.

- Extract until you get a function which name can only be a description of its implementation.

DO ONE THING

- In order to make sure that a function does one thing, ensure that the statements within the function are all the same level of abstraction.
    - When levels of abstraction are mixed within a function body, readers may not be able to tell whether a particular expression is an essential concept or a detail.
- *We want a code that can be read as a top down narrative; Every function should be followed by those at the next level of abstraction.*

SWITCH STATEMENTS

- By definition they do N things.
- Cannot always be replaced, but sometimes it can be replaced by polymorphic classes.
  - Solution: Implement an abstract factory to hide the switch statement.
- Avoid using `switch` statements when possible.

ENTER. tech with a heartbeat

FUNCTION ARGUMENTS

- *The fewer the better*. Ideally **zero**, but it is not always possible so use at most **two**.
  - Functions with three arguments are considerably harder to understand.
- They can mess with the readability of the code by including lower abstraction level information to a function call.
- Should not be used as the output of a function.
  - Functions should give they returns values via return not via their arguments.

FUNCTION ARGUMENTS

- *Flag arguments are ugly!* And they immediately indicate that a function does **more than one thing**.

- Multiple arguments make sense when the arguments are ordered components of a single value (a 2D point for example requires X and Y).

- If your function receives 3 or more arguments wrap them in a class of their own.

❗ REMEMBER ❗

Argument names should form a verb/noum pair.

```
Ex: Write(String name)
```

FUNCTION CALLS SHOULD HAVE NO SIDE EFFECTS

- Functions should never do *hidden* things.
  - Do not make unexpected changes in globals or member variables.

- Avoid temporal coupling (when calling a function can have different effects depending on when it is called).

- If it cannot be avoided, make it clear in its name.

COMMAND AND QUERY SEPARATION

- *Functions should* **do** something or **answer** something, **never both**!

```
If(set(“username”, “uncle bob”))
```

- *What does the return of set means?*

COMMAND AND QUERY SEPARATION

- *Functions should* **do** smething or **answer** something, **never both**!

```
If(set(“username”, “uncle bob”))
```

  - *What does the return of set means?*

- It can be improved look like this

```
 If(AttributeExists(“username”)
    setAttribute(“username”, “uncle bob”)
```

USE EXCEPTIONS

- Prefer exceptions to error codes;
  - Returning error codes are a violation to the “command-query” principle;
  - Error codes a usually comes with enums which can be a dependency magnet.
- The use of Exceptions makes it easy to organize the code between happy and unhappy flows.
- Separate the bodies of try-catch blocks in their own functions as well;