**Testing Example: Connect Four**

**Something to Consider About Naming**

[The Case Against Descriptive Variable Names](https://medium.com/%40Jernfrost/the-case-against-descriptive-variable-names-3c09dee0bac5)

**A Higher construct: “subroutines”**

My focus has been on individual “lines of code” so far. That is, naming, commenting and “style”.

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), a **subroutine** is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular [task](https://en.wikipedia.org/wiki/Task_%28computing%29) should be performed.

Subroutines may be defined within programs, or separately in [libraries](https://en.wikipedia.org/wiki/Library_%28computer_science%29) that can be used by many programs. In different programming languages, a subroutine may be called a **procedure**, a **function**, a **routine**, a [**method**](https://en.wikipedia.org/wiki/Method_%28computing%29), or a **subprogram**. The generic term **callable unit** is sometimes used.[[1]](https://en.wikipedia.org/wiki/Subroutine#cite_note-1)

Subroutines - <https://en.wikipedia.org/wiki/Subroutine>

**Functions vs Methods -**

A **function** is a piece of code that is called by name. It can be passed data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed.

A **method** is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:

1. A method is implicitly passed the object on which it was called.
2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

From: <https://stackoverflow.com/questions/155609/whats-the-difference-between-a-method-and-a-function>

**High Quality Routines** -

# Well Defined

* has only one purpose that is clearly defined
* does it well
* only creates things as its primary purpose
* only destroys things as its primary purpose
* [Loosely coupled](https://searchnetworking.techtarget.com/definition/loose-coupling)
* has [high cohesion](https://stackoverflow.com/questions/10830135/what-is-high-cohesion-and-how-to-use-it-make-it)

# Function Length

What is the right length for a routine?

## 25 Lines

This is an artificial and capricious number of lines. It has NO BASIS IN REALITY. As I have mentioned before, a line or statement is not a uniform unit of measure. The purpose for choosing a specific line count is:

1. It is concrete so students and graders know if it has been exceeded
2. It induces repetition on the part of the student in enduring the "psychological overhead" of defining and writing functions (methods, routines,..). You are forced to create more routines. This gives you practice in defining smaller pieces which hopefully will make you more comfortable in creating newer routines.
3. force a finer resolution in design. It should help focus your design toward smaller pieces. These smaller pieces will usually be private “helper” methods.

## One page

The meaning of "page" is not clear.

## Other Restrictions

## “Can be explained in a single breath.” Like the line count, this is arbitrary for similar reasons.

## My recommendation

This is more than a readability decision. This can help you in determining how fine your decomposition becomes.

The **largest** piece of code that is **small enough** to:

* describe/specify
* write (code)
* comment
* test

"off the top of your head".

### **Advantages**

* this manages complexity to your comfort level
* simplifies testing
* simplifies commenting
* grows along with your skill/experience

### **Potential Disadvantages**

your level of decomposition may not match the skill of your team

Let’s look at the refactored Douglas-Peucker code:

From my wiki: <https://wiki.illinois.edu/wiki/pages/viewpage.action?pageId=543401617>

**Testing Exercise - Connect Four**

<https://www.mathsisfun.com/games/connect4.html>

Plays like TicTacToe with gravity

* Board: 7 columns, 6 rows
* Insert a disc at the top of a column. Disc drops to the lowest slot in the column that is not already occupied.
* Win by creating a row, column or diagonal of length 4.
* No column may contain a disc that does not have a disc in the row below (except the bottom row).

Input: a string of N characters representing each slot in the 6 by 7 grid where the first character is the lower left corner of the grid the following characters appear on the next rows, moving left to right.

* Any character other than X or O is considered an empty slot.
* Any input string **longer** than 42 characters will be **truncated**. (long strings are OK).
* Any input string **shorter** than 42 characters will be assumed to contain all spaces after the last character.

Examples: (“M” is used because it occupies a similar size on line. This letter could be any letter other than X or O.)

“XMMMMMM” - a single “X” tile in the lower left corner

“XMXOMMM” - X, space, X, O, 3 spaces

================================================================

**public enum** Evaluation {

 ***InvalidInput***, */\* The string provided does not describe a valid tic-tac-toe board \*/*

***NoWinner***, */\* valid board, but the game hasn't been won by either player yet \*/*

***Xwins***, */\* valid board, and the game was won by player X \*/*

***Owins***, */\* valid board, and the game was won by player Y \*/*

***UnreachableState*** */\* string describes a board, but not a board state that is reachable from a valid game \*/*

================================================================

How do we deal with the input?

What are some types of

* Invalid boards?
* Unreachable states?

How do we detect a win?

Bonus: can you generalize your design to any size board?

#### [You Still Don’t Know How to Do Unit Testing (and Your Secret is Safe with Me)](https://stackify.com/unit-testing-basics-best-practices/)

[Create Tests in IntelliJ](https://www.jetbrains.com/help/idea/create-tests.html)

**Generically speaking: Java Assert();**

<https://www.geeksforgeeks.org/assertions-in-java/>

**Testing Example code:**

<https://www.java2novice.com/junit-examples/>

<https://www.vogella.com/tutorials/JUnit/article.html>

<https://www.java2novice.com/junit-examples/test-annotation/>

[TicTacToe SP2019 Assignment Doc](https://courses.grainger.illinois.edu/cs126/sp2019/assignments/TicTacToe.pdf)