# Pure versus Impure Code

## Pure Code Rules (Axioms)

Pure code consists of functions that conform to these two rules:

1. Takes inputs only from its parameters.
2. Outputs only via its return value.

## Example of a Pure Function

This function, , is pure:



The function takes input only from its parameters (rule #1). The function outputs only via its return value (rule #2).

## Example of an Impure Function

Suppose  is an external (global) variable. This function  is impure:



The function violates rule #1 – it takes input from other than its parameters; it takes input from.

## Another Example of an Impure Function

Again, suppose  is an external (global) variable. This function  is also impure:



The function violates rule #2 – it sends output to other than its return value; it changes the value of.

## I/O is Impure

Functions that perform input or output are impure. The following two examples show why.

**Example 1**: Suppose that  reads (inputs) a number, increments it, and that result is its return value:



The function violates rule #1 – it takes input from a source other than its parameters.

**Example 2**: Suppose that  writes (outputs) the value of its parameter:



The function violates rule #2 – it outputs to a destination other than its return value.

## No Input or Output?

All code that does I/O is impure. Assuming pure code is preferred over impure code (see below, *Advantages of Pure Code*), should code never perform I/O?

No. It means that code should separate those functions that perform I/O from those that are pure code.

Advantages of Pure Code

Pure functions can combine to make larger pure functions. Impure functions combined with pure functions make larger impure functions.

A consequence of pure code is that since it is dependent only on its inputs (rule #1) you can make statements like

is always is equal to  at any time

without knowing what the function  is, or how it’s implemented. That's quite powerful, and can be useful in optimization.

A pure piece of code causes no side-effects, and it is not affected by other code's side-effects. This property is very useful in parallel computations.

Consider a pure function that is called by some code. You can safely ignore the function when analyzing the calling code; you know the function doesn't change any state.

Pure code is easier to reason about, compose, and change than impure code.

## Programming Design Principles

**Principle 1**: Isolate/separate impure code from pure code.

**Principle 2**: Make most of your program pure, especially important logic.

These principles can be implemented in any programming language.

## More Information

Here are web pages that discuss pure code:

<http://en.wikipedia.org/wiki/Pure_function>

[http://en.wikipedia.org/wiki/Referential\_transparency\_(computer\_science)](http://en.wikipedia.org/wiki/Referential_transparency_%28computer_science%29)

<http://en.wikipedia.org/wiki/Idempotence#In_computing>

## Acknowledgements

Thanks to the following individuals for enlightening me on this fascinating subject:

* Thomas Davie
* Christopher Done
* Mike Meyer
* Sean Perry
* Paul Sargent