Common code smells

# Application-level smells

* **Mysterious Name:** functions, modules, variables or classes that are named in a way that does not communicate what they do or how to use them.
* **Duplicated code:** identical or very similar code that exists in more than one location.
* **Contrived complexity:** forced usage of overcomplicated design patterns where simpler design patterns would suffice.
* **Shotgun surgery:** a single change that needs to be applied to multiple classes at the same time.
* **Uncontrolled side effects:** side effects of coding that commonly cause runtime exceptions, with unit tests unable to capture the exact cause of the problem
* **Variable mutations:** mutations that vary widely enough that refactoring the code becomes increasingly difficult, due to the actual value's status as unpredictable and hard to reason about.
* **Boolean blindness:** easy to assert on the opposite value and still type checks.

# Class-level smells

* **Large class:** a class that has grown too large. See God object.
* **Feature envy:** a class that uses methods of another class excessively.
* **Inappropriate intimacy:** a class that has dependencies on implementation details of another class. See Object orgy.
* **Refused bequest:** a class that overrides a method of a base class in such a way that the contract of the base class is not honored by the derived class. See Liskov substitution principle.
* **Lazy class/freeloader:** a class that does too little.
* **Excessive use of literals:** these should be coded as named constants, to improve readability and to avoid programming errors. Additionally, literals can and should be externalized into resource files/scripts, or other data stores such as databases where possible, to facilitate localization of software if it is intended to be deployed in different regions.
* **Cyclomatic complexity:** too many branches or loops; this may indicate a function needs to be broken up into smaller functions, or that it has potential for simplification/refactoring.
* **Downcasting:** a type cast which breaks the abstraction model; the abstraction may have to be refactored or eliminated.
* **Orphan variable or constant class:** a class that typically has a collection of constants which belong elsewhere where those constants should be owned by one of the other member classes.
* **Data clump:** Occurs when a group of variables are passed around together in various parts of the program. In general, this suggests that it would be more appropriate to formally group the different variables together into a single object, and pass around only the new object instead.

# Method-level smells

* **Too many parameters:** a long list of parameters is hard to read, and makes calling and testing the function complicated. It may indicate that the purpose of the function is ill-conceived and that the code should be refactored so responsibility is assigned in a more clean-cut way.
* **Long method:** a method, function, or procedure that has grown too large.
* **Excessively long identifiers:** in particular, the use of naming conventions to provide disambiguation that should be implicit in the software architecture.
* **Excessively short identifiers:** the name of a variable should reflect its function unless the function is obvious.
* **Excessive return of data:** a function or method that returns more than what each of its callers needs.
* **Excessive comments:** a class, function or method has irrelevant or trivial comments. A comment on an attribute setter/getter is a good example.
* **Excessively long line of code (or God Line):** A line of code which is too long, making the code difficult to read, understand, debug, refactor, or even identify possibilities of software reuse.