Introduction:

Programming has been around for a very long time (relatively speaking). As such, developers have discovered many unique paradigms for how we think about and write code. Programming in many ways has become a free market of ideas, each person advocating for one pattern or algorithm to rule them all. It is one thing to learn how to read and write code, but it is another thing entirely to be able to reason about and engage with these ideas. Every good developer, at a base level, needs to understand at least some of these core concepts – how they are applicable, and what their advantages and disadvantages are. It is this knowledge that I hope to be able to impart unto you, which will hopefully get you up to speed quicker than it took me.  
  
Paradigms:

If you read the word paradigm and weren’t entirely certain as to what I meant, then allow me to explain. A programming paradigm generally refers to a philosophy that is adhered to as we write software. Paradigms are not necessarlily tied to any particular language, though many languages are written with specific paradigms in mind. As an aside, this teeters into the realm of Domain Specific Lanuages (DSLs) compared to General Purpose Languages (GPLs). A domain specific language is a language which is written and used for a specific field or domain. An example of a GPL is R or Matlab, which are intended to be used for data analysis. A general purpose language, on the other hand, might be something like Python. Anyhow, a common language paradigm with which most people are familiar is Object Oritented Programming (OOP). OOP is the philosophy or paradigm which is adhered to as we program. As I stated, it is not necessarily tethered to the language, but the language typically pushes us towards the paradigm it was designed for. C++ is an example of a language which loosely pushes towards using OOP, whereas Java more strongly encourages using OOP. In neither case, however, are you forced to write OOP code. It is important to note that classes do not make a language OOP by any means. OOP entails a lot more than that, which we will cover explicitly at a later point. Paradigms don’t have to just apply to programming either. There are many database paradigms, for example, the most prominent being relational databases. We will be focusing specifically on programming paradigms for the purposes of this document though. Here is an heirarchical overview of the paradigms we will cover:

* **Imperative**: The code direcly controls execution flow and state change
  + **Procedural**: Organized as procedures which call each other
  + **OOP**: Organized as objects that contain both structure and associated behavior
* **Declarative**: Code declares properties of the desired result, but not how to compute it
  + **Functional**: A desired result is declared as the value of a series of function evaluations
  + **Logic**: A desired result is declared as the answer to a question about a system of facts and rules
  + **Reactive**: A desired result is declared with data streams and the propogation of change

These may seem like just a word salad as you read them, but as we cover them individually, the concepts will hopefully become more concrete.

Imperative:

You may think of the word imperative as meaning urgent or necessary, but another way that the word can be defined is command or instruction (which is, of course, the way it’s intended in this context). When we say imperative language, we’re being a little more abstract, as an imperative language can be referring to one of a few imperative paradigms. Imperative languages are typically one of two language paradigms: procedural, or OOP. There are perhaps others, but these are the most common. We’ll cover both of these separately, but for now let’s focus on what makes these paradigms imperative. The