# A Tour of Babel: Introduction

Justin Kenworthy
darknexus87@gmail.com

January 8, 2010

## Contents



## 1 Overview

This series of articles will cover various popular programming languages. Its purpose is to give a tour of each of the covered languages so that developers may better understand why they might choose those languages for development. As the title of the series implies, this is a tour of a world which offers many different languages for an equally diverse variety of application areas. It should be understood that there are countless numbers of programming languages (courtesy of academia) so there is no way (or reason) to cover them all. The definition of "popular" for this context should be accepted as those languages which are main-stream so to speak. Although there are plenty of obscure (and possibly exciting) languages to tour, I leave that to the reader to accomplish on their own.

I would like to offer a list of the languages to be covered in the series, however I really cannot provide it. At this point I have a few languages I would like to dig into but it will mostly be based on my availability of time and interest. I

hope for this to be a continuing project so I am guessing I will end up covering quite a few languages after a while (if that helps you gauge my intended bredth).

These articles assume your background knowledge of programming is at least modreately experienced. I assume you are comfortable with at least one or two programming langauges already and that you have, at least, some experience in developing small to medium sized applications. For those topics which are more advanced I may devote more time to explaining some fundamental concepts before delving into the pertaining features of a language. If you have absolutely no programming experience then, at the very least, you can get a feel for the languages covered.

These tours are by no means exhaustive and serve to give an overview of the important features of each language as well as those features which set them apart from other languages. Of course, the reader must understand that that these tours are bound to be founded on a subjective experience with each language. I will try to keep my analysis as objective as possible but this is very difficult to completley achieve. It is up to the reader to make the final decision as to whether they view a language to be useful, for whatever purpose.

# 2 Language Features

Before we start touring languages let us first discuss what major components of languages are important to consider. There are many different ways to compare programming languages but perhaps the most common (and important) are the history and influence; performance and stability; platform, license, and support; syntax; power and usage; typing systems; paradigms; and data structures, and standard libraries. Of course, more will be covered depending on each language and the feature set each has to offer. For each article, however, these topics will comprise the major framework for the analysis of each language so to offer some degree of consistency throughout the series.

Some readers may be wondering why the chosen language components are important. I will devote a little time to talking about why, as a developer, you should be concerned with each of the selected components.

## 2.1 History and Influence

Although the history of a programming language may not seem immediately important, it is. Not only are the histories of most languages semi-interesting, we can also learn a lot by understanding what influenced the design decisions of those languages. The most obvious impact of a language's influences is in the area of syntax. Languages (like Python, Ruby, Java, etc.) borrow their syntactic flavor from influential predecessors (like C, Perl, Lisp, etc.). Understanding a language's influential predecessors can also greatly expedite the learning process which often involves memorizing syntactic structures. By better understading the origins of a language we can better understand its ultimate goal.

## 2.2 Platform, License, and Support

The topic of platforms is fairly straightfoward but still very important. The operating system and architecture of a target machine seriously impact the languages that we can develop with. Most of the main stream languages are multi-platform so there is no worry there but sometimes we can run into trouble.

Licensing and support are incredibly important when it comes to deciding how and when to use a language. If the license is free-software-like then we can guess the language will be fairly solid in both design and stability. Because these languages are under the scruitiny of public and peer review, they progress rapidly and for the better (hopefully). The community or company which supports a given language is also very important. Developers are less likely to use a language if little support is available. We are all bound to get stuck at some point and without proper support in place we might just stay stuck.

## 2.3 Performance and Stability

The maturity of a language is generally directly linked to its popularity, stability, security, and effeciency. The more mature a language becomes, the more it is used and therefore the more mature the compilers and interpreters become as well. The maturity of a language is also directly related to the design of the language. Take for instance Python 3.0 which is fixing many of the design flaws from previous versions of the language. A language's age also says a lot about the community which supports it. By discussing the maturity of a language we can better decide how seriously we should treat it. Is it a scripting language or a fully fledged programming language? Can we produce production quality software with it or is it better used for less critical development?

Performance and stability are topics which must be taken into consideration. Depending on the deployment environment and usage, a piece of software may need to be lightning fast and rock solid stable. On the other end of the spectrum we have sluggish and possibly unreliable. Either way a decision has to be made concerning the performance of a language and how it will affect the software and the user.

This section in language tours will most likely contain benchmarks to some extent and some research on stability. Because these traits are often more emperical than the others I will try my best to present reliable statistics.

## 2.4 Syntax

Syntax is fairly straightforward in some regards, but more complex than you might think in others. Once you learn the syntax of one language, all languages in the same paradigm only differ by syntax (for the most part). For this component we will discuss common syntactic structures such as flow control, functions, classes, mathematical expressions, and so on. Is the syntax tedious and difficult to read or does it flow because it is full of syntactic sugar? Is the syntax concise and powerfull or verbose and weak? Does the syntax compliment the applica-

tion area for the language? All of these questions, and more, are important to consider when analyzing syntax. Although syntax may seem straightforward it is often times what makes or breaks a language in main stream usage. For instance, many developers cannot stand the whitespace aspect of Python, while others cannot live without the many parenthesis of Lisp (and vice versa).

## 2.5 Power and Usage

This section will address the unique and "powerful" features of languages, whatever that may be. This is where the interesting and truly useful features of languages will be discussed. Things like clever introspection, lambda expression trickery, safetey mechanisms, etc., will be covered. Some languages are far more powerful than others (for instance high level in comparison to low level languages). This will probably be the most subjective portion of each article for obvious reasons. Expect to see lots of code in these sections as my goal will be to showcase the usefulness and semantic power of the languages in question.

## 2.6 Typing System

The typing system of a language defines how data and objects (not strictly in the OO sense) interact with each other. This is a crucial aspect of any language as it directly correlates to the saftey, stability, and power of the language. Not all typing systems are created equal. Each language's take on type theory changes the way we model problems and solutions. Dynamic or static? Weak or strong? nominative or structural? Types are something most developers take for granted but are the area where most languages have an opportunity to truly shine. If we do not fully understand a language's typing system then we do not fully understand the language, or at the very least we cannot recognize its full potential. This section will be one of the most technical and theoretical of the feature sections as it directly relates to the principles of programming languages.

## 2.7 Paradigm

A language's paradigm is important in many of the same ways as typing systems, except in a much larger way. A language's paradigm completely controls how we can model problems and solutions. The two most popular in modern times are declarative and imperative. These are both very broad terms but what we really need to know for know is that they are opposed. Declarative paradigms give developers a way to describe what needs to achieved rather than how. Imperative paradigms are just the opposite: they allow developers to describe how rather than what. Most languages are not strictly pure in their paradigms and in fact offer a variety of paradigms to choose from. For example, Python is a muli-paradigm language which offers flavors of both declarative and imperative programming as well as some nifty meta-programming.

This section will discuss which paradigms a language offers as well as how those paradigms co-exist (or don't) to achieve their intended goals. This section, like typing systems, will also be fairly technical. Paradigms, also like typing systems, are pretty intresting so the technicality will not be without reward. The better developers can grasp the functionality (no pun intended) of their chosen language's paradigm, the better they can rapidly develop elegant solutions. There's nothing worse than fighting against a paradigm so this is one of the most important aspects when it comes to programming languages.

## 2.8 Standard Library and Data Structures

Standard libraries and data structures make languages easier to apply. If languages come packaged with a rich toolset then developers are going to be far more likely to use those "batteries included" languages over languages with less impressive collections. If we are able to determine how rich a language's included toolset is we can determine the sort of applications the language is best suited for. Some languages, like Python and Ada, have collosal libraries backing them. Other languages, like C, offer fewer and less useful standard libraries.

Data structures are incredibly important (more so than libraries) so most of this section will be devoted to that aspect of a language toolsets rather than libraries. The ability to create new data structures and use existing built-in types is where most developers start when deciding which language is best for the task at hand. This section should give you a good idea of what you will have to work with in the sense of standard libraries and included utilities.