Programming Languages

There are many different programming languages including Python, Java, C, C++, C#, JavaScript, PHP, Lua, F#, Ruby, Go, Assembly, Fortran, COBOL, and many more. There are also variations of languages such as NodeJS and Ruby on Rails. Some languages are also higher levels than others. The higher the level the further away it is from the hardware and typically the easier it is for humans to read and write. For example, C# is a very high-level language and Assembly is a low-level language. Between the two resides C/C++.

Different Uses:

Different languages are used for different tasks and work differently. JavaScript is used in web applications, C/C++ is used for software, and Python is used for scripting. Of course, you can use these languages for different things as well. For example, you can use Python to build websites (what can't it be used for?).

Execution Differences:

There are two main types of languages: **compiled** and **interpreted**.

- **Compiled languages** are compiled/translated directly into native machine code. This means that any computer that it was compiled for can directly run the program. The nice thing about compiled programs is that they, usually, don't need an extra program to run them. For instance, if you want to run a Java program you need to install the **Java Runtime Environment (JRE)** first. However, to run a C++ program you don't need anything like the JRE to do so.
  I want to reiterate that for the program to run, the code must be compiled into a language that machine understands or else it won't be able to execute. This is a good time to add that Linux is not "bit compatible" like Windows is. You can't run a 32-bit program on 64-bit Linux like you can on Windows.
- **Interpreted languages** are languages that aren't compiled directly into machine code but are instead interpreted from their source *indirectly* into machine code. This interpretation can be done by compiling parts or even individual lines of code then executing that part. For example, when you run a Python program each line is essentially compiled then executed individually (this isn't exactly correct but we'll get to that soon). The advantage of this is that the program doesn't need to be re-written or compiled for specific systems. The big downside to this is that they tend to be *significantly* slower than compiled languages.
  Another form of interpretation is using some sort of medium such as bytecode (bytecode is talked about in "Intermediate Languages").

Intermediate languages

I feel like calling languages interpreted undersells them, especially when talking about Java and .NET. I consider an interpreted language as any language that isn't compiled directly into machine code. So I categorize Java as an interpreted language, however, I refer to it as an intermediate language if I'm trying to be more specific. It's extremely important to talk about intermediate languages. Let's use Java as an example. A program written in Java is compiled into what's called **Java bytecode**, this is also the binary/executable that will be distributed. You can think of this bytecode as a semi-compiled version. Note that this is *not* machine code. This bytecode cannot be understood by the architecture so it needs something to translate that bytecode. To run a Java program you have to "install Java" (specifically the **Java Runtime Environment (JRE)**) which then provides a translator for the bytecode. This translator will translate the bytecode into machine code. When the program is executed, the bytecode is interpreted by the **Java Virtual Machine (JVM)** which is part of the JRE. When the JVM interprets the bytecode what it's doing is translating the bytecode to machine code which can then be executed natively. A compiler that does this sort of compilation and execution is considered a **Just-In-Time (JIT)** compiler. These sort of languages have the advantage of being portable because as long as the user has the interpreter installed they can run it on any architecture. Of course, the interpreter itself *will* be architecture specific.

.NET is similar to Java. Quick note, .NET consists of many languages, but C# is "*the*" .NET language. .NET is compiled into **Microsoft Intermediate Language (MSIL)** which is just bytecode. This bytecode is then run by a part of the .NET Framework. The part of the .NET Framework I'm referring to is the **Common Language Runtime (CLR)**. The CLR includes garbage collection, security things, and the JIT compiler. That JIT compiler will interpret the bytecode into machine code just as Java does. As I said, .NET is very similar to Java in terms of how it compiles and executes.
Another huge advantage of these kinds of languages is security. Everything runs through some sort of VM which can do all kinds of security checks and management. Garbage collection is a huge advantage that deserves to be reiterated. Garbage collection makes a developers job easier while also dealing with memory management automatically which is good for security.

Back to Python. Python is similar to Java and .NET because Python is actually compiled into bytecode. This bytecode is then interpreted line by line by a VM just like Java and .NET.