**Define a programming language**

A programming language is a means of instructing computers to perform a specific task. A programming language is a formal constructed language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behaviour of a machine or to express algorithms.

**Different Programming Languages**

To create a program for a computer, an app for a phone or even a website, you need to know one of a large variety of programming languages. Programming languages tell the computers what to do and allow you to interact with the computer to perform functions such as making a document, controlling game characters and photo editing. Like spoken language, each has its own rules and common areas of use. Why are there so many? Why is a webpage written in a different language than a Windows program or a game?

The primary reason for many different programming languages, is that there are many different types of programs and computers. Some programming languages are designed to be used by many different computer configurations and operating systems . Some languages just control elements inside of a program, (these are usually referred to as scripting languages) while others define and create the entire program themselves (these are referred to as system programming languages).

HTML, the language used to write webpages, is an example of a scripting language. Since HTML runs inside of a program, (a web browser) it needs to only define how to control aspects of the browser such as its ability to display text and images or get input from users. If a web browser complies with the HTML standard, it will be able to understand the code and display the page correctly.

In comparison, a programming language like C++, a system programming language, is used when a programmer needs maximum flexibility to design the program the way they choose. The downside of programming languages like C++, is that everything from the basic interface to the mathematical equations that run the program, must be defined and included by the programmer.

There are over 20 programming languages currently used today.  What are some of the most popular?

**Windows Applications:** C#, Visual C++, Visual Basic.Net, DirectX API’s, HTML 5, Java

**Mac OS Applications:** Objective C, X Code with Cocoa Framework, Java

**iPhone Apps:** Objective C with Cocoa Framework

**Android Apps:** Java and some C#

**Web Pages:** HTML, CSS, Flash, JavaScript, Java, PHP, Perl, ASP.net

**TV’s & Electronics:** Assembly and C#

Want to see an example of the differences the programing language can make when writing a simple application? All the code below would produce the same thing; the text “Hello World”.



**Explain the historical development and evolution of programming languages**

**Follow the link for brief description of each language and its uses.**

<http://www.howtogeek.com/94917/the-evolution-of-computer-programming-languages-infographic/>

|  |  |
| --- | --- |
| **Year** | **Programming Language** |
| 1957 | **Fortran -** Formula Translation |
| 1959 | **COBOL** – **Co**mmon **B**usiness **O**riented **L**anguage |
| 1964 | **BASIC**  |
| 1969 | **C**  |
| 1970 | **PASCAL after Blaise Pascal** |
| 1983 | **C++** |
| 1987  | **Pearl**  |
| 1991 | **Python** |
| 1993 | **Ruby** |
| 1995 | **PHP** |
| 1995 | **JAVA** |
| 1995 | **Javascript** |
| 2005 | **Ruby on Rails** |



**Procedural programming** – languages such as Pascal, Fortran, Cobol, etc

**Object Oriented Programming** – OOP programming languages such as Java, Visual Basic, Python, etc

**Scripting languages** are programming languages that don't require an explicit compilation step.

For example, in the normal case, you have to compile a C program before you can run it. But in the normal case, you don't have to compile a JavaScript program before you run it. So JavaScript is sometimes called a "scripting" language. Javascript on a webpage runs within a Browser for example.

**Syntax of a programming language**

Syntax refers to the spelling and grammar of a programming language. Computers are inflexible machines that understand what you type only if you type it in the exact form that the computer expects. The expected form is called the syntax.

**Programming languages - what they have in common**

* e.g. strict syntax rules, data storage, input statements, output statements, branching, looping.
* Planning Phase – Design Phase – Testing Phase

**Programming languages - their differences**

* e.g. different syntax, different structures, different focus
* procedural versus object oriented (OOP)

**Video** <https://www.youtube.com/watch?v=qmksVfulV0o>

Machine code is a computer programming language consisting of binary or hexadecimal instructions which a computer can respond to directly.

**Low-level programming language**

(See Machine Language <https://chortle.ccsu.edu/java5/Notes/chap04/ch04_4.html>)

In computer science, a low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. Generally this refers to either machine code or assembly language.

Low-level languages are useful because written in them can be crafted to run very fast and with a very small memory footprint. However, they are considered more difficult to utilize because they require a deeper knowledge of machine language.

Languages such as C and C++ are considered "lower-level" — they provide a minimal amount of abstraction at the smallest possible cost to performance and efficiency. These abstractions, such as classes, lambda functions and macros, allow programmers to use complex functionality without writing overly complex code. For this reason, lower-level languages are used in projects where abstractions are necessary to keep code highly readable and maintainable, but where maximum performance is still paramount. Many operating systems and high-frame rate computer games are a good example of this.

**High level programming languages**

A high-level language is a computer programming language that isn't limited by the computer, designed for a specific job, and is easier to understand. It is more like human language and less like machine language. However, for a computer to understand and run a program created with a high-level language, it must be compiled into machine language.

The first high-level languages were introduced in the 1950's. Today, there are many high-level languages in use, including BASIC, C, C++, Cobol, FORTRAN, Java, Pascal, Perl, PHP, Python, Ruby, and Visual Basic.

**Difference between Low-Level & High-Level Language**

**High-level Language**

1. Learning - High-level languages are easy to learn.
2. Understanding – High level languages are near to human languages.
3. Execution - Programs in high-level languages are slow in execution.
4. Modification - Programs in high-level languages are easy to modify.
5. Facility at hardware level - High-level languages do not provide much facility at hardware level.
6. Knowledge of hardware Deep - Knowledge of hardware is not required to write programs.
7. Uses - These languages are normally used to write application programs.

**Low-level languages**

1. Learning - Low-level languages are difficult to learn.
2 Understanding - Low-level languages are far from human languages.
3. Execution - Programs in low-level languages are fast in execution.
4. Modification - Programs in low-level languages are difficult to modify.
5. Facility at hardware level - Low-level languages provide facility to write programs at hardware level.
6. Knowledge of hardware Deep - Deep knowledge of hardware is required to write programs.
7. Uses - These languages are normally used to write hardware programs.

**OOP – Object-Oriented Programming Concepts** *(this will be covered during OOP classes)*

If you've never used an object-oriented programming language before, you'll need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language.

**What Is an Object?**

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner.

**What Is a Class?**

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behaviour.

**What Is Inheritance?**

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language.

**What Is an Interface?**

An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it.

**What Is a Package?**

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. This section explains why this is useful, and introduces you to the Application Programming Interface (API) provided by the Java platform.