# OOP Pillars

The OOP paradigm is prevalent in most modern programming languages. JavaScript is currently the most popular programming language, and it’s pretty much impossible to be a software developer without knowing JS. However, before it’s recent ES6 (JS 6) overhaul introducing syntactic sugar for class definition and OOP building blocks such as inheritance, JS was mostly a big clunky mess when it came to syntax for OOP implementation. You could either define a “class” through a function or create an object literal (singleton), also using a function.

Using a function:

function Apple (type) {

this.type = type;

this.color = "red";

this.display = function() {

return this.color + ' ' + this.type + ' apple';

};

}

var apple = new Apple('macintosh');

apple.color = "green";

console.log(apple.display());

Creating a singleton object:

var apple = new function() {

// data member and function definition

};

}

The syntax for inheritance was even more clunky, with the use of function prototypes to act as constructors for inherited classes.

Polymorphism as a concept mainly involves making type-definitions loose and forgiving by offering multiple implementations for a single type (object). However, since JS is a weak-typed, dynamic programming language, variables can change their type any number of times. Therefore, the notion of “type polymorphism” isn’t valid for JS. Overloading, however, is possible through arrow functions.

Abstraction also doesn’t have a native solution in JS. We have to implement our own fake abstract solution by throwing an error if the constructor attempts to make an object of the abstract class’ type.

While JavaScript has objects and JS functions are primitives which make OOP applications possible, it is not a language designed with native support for encapsulation or polymorphism.

# Environment

As previously discussed, JavaScript is a dynamic loose-typed language, meaning variables types are inferred upon assignment rather than at declaration. Variables can hence change their types, which isn’t possible in a strictly-typed language such as C++. This offers JS programmers more flexibility but also comes with debugging issues as run-time errors with mistyped conversions, assignments, etc. are extremely common in JS. JS offers great control over code operation and interaction between objects, but not much control over the memory handling.

On the other hand, the C++ environment offers a lot of control over the underlying memory structure of the application and an efficient C++ program requires deep knowledge on manipulating the memory to achieve optimal results. It offers native support for encapsulation and data hiding, as well as polymorphism.

The two languages promote vastly different styles of programming – JS is flexible and forgiving with it’s free-flowing syntax and loose-typing, but is harder to optimize and debug, and is insecure. C++ offers more security and fine-tuned controls to optimize a program but requires greater understanding of memory manipulation.

# Advantages & Drawbacks

C++ was intended for system and enterprise programming, as well as low-level optimization. C++ is compiled, so it is mostly used for back-end, especially in web-based applications. C++ is the go-to choice when making high-performance programs

C++ is not useful when portability is paramount, as it needs to be compiled differently for different OSes and different chip architectures (32 and 64-bit). It is generally not a good idea to build your front-end or any client-side application with C++.

JavaScript, on the other hand, is a great choice for client-side applications and web-based applications. In fact, all modern web browsers support only HTML, CSS, and JS for websites. Since JS uses an interpreter, every browser comes bundled with it’s own JS interpreter. Although today everything from servers to desktop applications are being built with JS, it’s main application remains in programs which interact with the client.

JavaScript shouldn’t be your go-to choice for building back-end frameworks because it’s insecure and hard to debug. Especially if you need to perform complex calculations, you’re better off using a low-level language that can be optimized like C++.