# JSJS

*A general purpose, strongly typed programming language for the web*

**J**ain Bahul | Language Guru **S**rivastav Prakhar | Manager **J**ain Ayush, | Tester **S**adekar Gaurang | System Architect

---

## Description

JSJS is a strongly typed language for the web. Taking inspiration from languages such Scala, TypeScript and OCaml, the goal of JSJS is to enable programmers to write type-safe, correct and robust code that compiles down to Javascript. Our target users are programmers who like functional nature of Javascript but want the safety and guarantees that come with a statically typed language without sacrificing its ubiquity.

## Motivation

> *Any application that can be written in JavaScript, will eventually be written in JavaScript* - **Jeff Atwood**

Javascript is the *lingua franca* of the web. With its humble beginnings in engine of the browser, Javascript has slowly and steadily has gained wide adoption across the developer community at large. Famed to have gone from conception to a working implementation in just 10 days, Javascript now runs on everything - from massive servers to even the smallest raspberry pi. However, javascript's massive success has often been met with wide bewilderment (even disdain in some cases) from developers.

The members of this group, however, believe that Javascript is an incredibly powerful language. Having spent most of our time programming for the web - both frontend and backend, building apps ranging from trivial browser plugins to standalone cross-platform native ones, we marvel that at the fact that a language could have such a widespread use and applicability. The good news that we are not alone in this assessment. The biggest tech giants of today are spending numerous resources in making the language even better. This includes but is not limited to the monumental advances done by the Google folks on V8, the recent Chakra engine by Microsoft and the numerous tools open-sourced by Facebook.

As a language Javascript is quite interesting. With features such as closures, functions as first class objects, asynchronous programming via callbacks and a prototype based system, Javascript can definitely be touted as a modern language. On the other hand, JS is infamous for weird object rules, global name definitions and hard to grok prototype chain. The fact that one of the most popular books on JS is titled “Javascript: The Good Parts” indicates the warts in the language.

Our goal with **JSJS** is to create a functional language that has features such as type safety and immutability but at the same time can be used where Javascript runs. Despite Javascript treating functions as first class objects, Javascript is typically used as an imperative programming language. Its lack of structure has encouraged programmers to think of Javascript objects in terms of state, instead of attempting to transform data. Javascript is a product of rapid evolution, and thus for many people from the functional programming school of thought, it seems broken. Although its core library is small, Javascript’s weak typing and object construction system create a broken mix, where functions are first class, yet objects are used imperatively. These are some of the issues that we plan to address with JSJS.

In conclusion, we believe that JSJS’s list of features and Javascript’s ubiquity is a deadly combination that make it compelling tool for programmers.

## Lexical Conventions

### Data Types

| Data Types | Example |
|---|---|
| num | 42, -42, 4.2, -0.00042 |
| bool | true, false |
| string | “jsjs”, “is”, “the”, “best”, “!!!” |
| unit | () |

### Operators

| Operator Name | Syntax | Applicable Data Types |
|---|---|---|
| Integer Arithmetic | a + b, a - b, a * b, a / b | num |
| Modulo | a % b | num |
| Equal to | a == b | num, bool, string |
| Not Equal to | a != b | num, bool, string |
| Comparison | a <= b, a < b, a >= b, a > b | num, bool, string |
| Logical AND | a && b | bool |
| Logical OR | a \|\| b | bool |
| Logical NOT | !a | bool |
| Concat | ^ | string |

**Functions**

| Function type | Syntax |
|---|---|
| Function expression | `def square (x: num) : num = x * x;` |

**Syntax**

```
// simple expression
def square (x: num) : num = x * x;

// function multi-line
def cube (x: num): num = {
  val sq = square(x);
  x * sq;
}

// equivalent to def sq (x: num) : num = x * x;
val sq : (num) -> num = (x: num) : num => x * x;

// defining map
def map (f: (T) -> U, l: list T): list U {
   if (List.empty?(l))
   then { []; }
   else {
     val res = f(List.hd(l));
     List.cons(res, map(f, List.tl(l)));
   }
}

// applying map
val squares : list num = map((x: num) : num => x * x, [1, 2, 3, 4]);

// finding factorial
def fact (n: num) : num = {
  if (n == 0) then {
     1;
  } else {
     n * fact(n-1);
  }
}

// summing a list
def sumlist (xs: list num) : num = {
  val aux = (total: num, l: list num) => {
    if (List.empty?(l))
    then { total; }
    else { aux (total + List.hd(l), List.tl(l)); }
  };
  aux(0, xs);
}

// printing a list
def printlist (xs: list string) : unit = {
  List.iter((x: string) : unit => { println(x); } , xs);
}
```