AANSA ALI

EVALUATION OF ALTERNATE PROGRAMMING LANGUAGES TO JAVASCRIPT

Masters of Science Thesis

Examiner: Professor Kari Systä
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 14 August 2013.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology
ALI, AANSA: Evaluation of alternate programming languages to JavaScript
Master of Science Thesis, 55 pages, 17 Appendix pages
August 2013
Major subject: Software Systems
Examiner: Professor Kari Systä
Keywords: WWW,JavaScript, DART, TypeScript, CoffeeScript, object oriented, inheritance, concurrency

The development of web applications for desktop and mobile has surged in recent years. The most popular web scripting language is JavaScript because all the browsers support it and its role as a scripting language of the WWW. It is a powerful and flexible language. However, it also has some shortcomings. For this reason, over the last few years many different web scripting languages have appeared, they give the solutions to the shortcomings of JavaScript.

In this thesis a number of emerging web scripting languages are surveyed and the most popular option, CoffeeScript, TypeScript and Dart, are evaluated in detailed level. We will explain what a scripting language is and how it works, JavaScript's problems in developing a web application, list of available scripting languages for web clients, the motivation behind these languages and their features that they add to JavaScript.

In order to show the results, an example web application is developed in all the languages. The main conclusion extracted of this thesis is that these languages address the shortcomings of the JavaScript such as they all have the compile time checking for errors, CoffeeScript adds the syntactic sugar to JavaScript syntax, object-orientation, inheritance. TypeScript and Dart have the type checking, modules and generics. Dart also supports the concurrency with isolates. It is easy to develop and maintain the complex and large scale applications in these languages.

To my parents, without their permission and support it was not possible to come to Finland for studies.

To my family, for being always there when I needed.

To Faisal Khan, who kept me highly motivated for the thesis work.

To my supervisor Kari Systä, without his assistance and aid, the work related to this thesis would not have been finished.

To everyone, who teach me and support me in every part of my life.

To my friends, who helped me with the corrections.

To the unknown person, whose following quote encourages me every time I feel disappointed:

"Don't worry one or two years down your life you won't even remember you had to go through this☺ "

Tampere, August 14, 2013

Aansa Ali

CONTENTS

1. Introduction ... 1
2. Scripting languages ... 3
  2.1. History of scripting languages ... 3
  2.2. Properties of scripting languages ... 3
  2.3. Scripting engine ... 4
3. What is JavaScript? ... 6
  3.1. History ... 6
  3.2. How it works? ... 7
  3.3. Paradigms ... 7
  3.4. Functional programming in brief ... 7
  3.5. JavaScript and Functional Programming ... 8
    3.5.1. Anonymous functions ... 8
    3.5.2. High-order functions ... 8
    3.5.3. Recursion ... 9
    3.5.4. Closure ... 9
  3.6. Object oriented paradigm in brief ... 10
  3.7. Java Script and Object oriented Programming ... 10
    3.7.1. Core Objects ... 10
    3.7.2. Custom Objects ... 10
    3.7.3. Prototype-based programming ... 10
    3.7.4. Object-orientation through Prototypes in JavaScript ... 11
    3.7.5. Inheritance ... 12
    3.7.6. Abstraction ... 12
    3.7.7. Encapsulation ... 12
  3.8. Why JavaScript is not enough? ... 13
    3.8.1. No Module ... 13
    3.8.2. No Visibility Control ... 13
    3.8.3. Weak Type System ... 13
    3.8.5. No support for generics ... 14
    3.8.6. Not enough polymorphism ... 14
    3.8.7. Development of a large application is hard ... 14
    3.8.8. Maintainability is hard ... 15
    3.8.9. Equality comparisons ... 15
    3.8.10. Number property lookups ... 15
    3.8.11. Reserved words ... 15
    3.8.12. Global variables ... 16
4. Scripting languages for web clients ... 17
  4.1. List of scripting languages for web client ... 17
    4.1.1. TypeScript ... 17

4.1.2. Dart ... 18
4.1.3. CoffeeScript ... 18
4.1.4. Haxe ... 18
4.1.5. Roy ... 18
4.1.6. Clojure Script ... 19
4.1.7. Opal ... 19
4.1.8. Iced Coffee Script ... 19
4.1.9. Live Script ... 19
4.1.10. Kaffeine ... 19
4.1.11. ParenScript ... 20
4.1.12. Fay ... 20
4.1.13. Ceylon ... 20
4.2. Importance of language popularity ... 21
5. What is CoffeeScript? ... 24
5.1. History ... 24
5.2. How it works? ... 24
5.3. Features it adds to JavaScript ... 25
5.3.1. Inheritance with CoffeeScript ... 26
5.3.2. List comprehensions ... 26
5.3.3. String interpolation ... 26
5.3.4. Splats (…) ... 27
5.3.5. Number property lookup ... 27
5.3.6. Reserved words ... 27
5.3.7. Global variables ... 28
5.3.8. Compile time checking ... 28
5.3.9. Existential Operators ... 28
6. What is DART? ... 30
6.1. History ... 30
6.2. How it works? ... 30
6.3. Features it adds to JavaScript ... 30
6.3.1. Optionally typed ... 30
6.3.2. Reified generics ... 31
6.3.3. Dart is purely object oriented ... 31
6.3.4. Closures and lexically scoped functions ... 32
6.3.5. Dart has mixins ... 32
6.3.6. Building large and complex applications ... 33
6.3.7. Concurrency support with isolation ... 33
6.3.8. Snapshots ... 33
6.3.9. Reliability ... 34
6.3.10. Security ... 34
6.3.11. Best usage of memory ... 34
6.3.12. Dart supports code sharing ... 34

6.3.13. Global namespace 34
7. What is TypeScript? 35
7.1. History 35
7.2. How it works? 35
7.3. Features it adds to JavaScript 36
7.3.1. Optionally typed 36
7.3.2. Type inference 36
7.3.3. Object Orientation 37
7.3.4. Inheritance 38
7.3.5. Modularization and multi-file 38
7.3.6. Scalable application structuring 38
7.3.7. Open and Interoperable 39
7.3.8. Build and maintain large applications 39
7.3.9. Refactoring 39
8. Comparisons 40
8.1. Performance comparison 40
8.2. Robust comparison 41
9. Application 43
9.1. TODO application in JS 43
9.2. TODO application in CS 45
9.3. TODO application in Dart 46
9.4. TODO application in TypeScript 48
10. Results and Conclusion 49
References 52
APPENDIX 1: INDEX.html 56
APPENDIX 2: BASE.css 57
APPENDIX 3: JS_SCRIPT.js 59
APPENDIX 4: CS_SCRIPT.coffee 62
Appendix 5: DART_SCRIPT.dart 65
Appendix 6: TS_SCRIPT.ts 67

TERMS AND DEFINITIONS

Term	Definition
App	Application
Brevity of Code	If the number of lines of code is reduced
CS	CoffeeScript
DOM	Document Object Model
ECMA	European Computer Manufacturer's Association
IIS	Internet Information System
Intellisense	Intelligent code sense or Autocompletion of code
JCL	Job Control Languages
JS	JavaScript
TCL	Tool Command Language
TS	TypeScript
XUL	XML User Interface Language
Weak Typing	Weak typed languages are those in which variables are not of a specific data type. It should be noted that this does not imply that variables do not have types; it does mean that variables are not "bound" to a specific data type.
Strong Typing	Programming languages in which variables have specific data types are strongly typed. This implies that in strong typed languages, variables necessarily bind to a particular data type.
Strict Typing	Strictly typed languages enforce typing in all data being interacted with. Languages where variables must be declared to contain a specific type of data.
Static Typing	Static typed languages are those in which type checking is done at compile-time
Dynamic Typing	Dynamically typed languages are those in which type checking is done at run-time

1. INTRODUCTION

"It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free."

- Steve McConnell

Web scripting languages became increasingly popular in the web application programming in the last years. There are many options for server-side languages but when it comes to browser we are limited to JavaScript. Flash was used earlier to do the same but now it is fading away because it is not supported by iPhone and iPads, websites are not SEO optimized, content is not shareable on Facebook and twitter. ECMA script programming language JavaScript is used more widely in the development of web applications and in the server-side (node. js). Even there are ways to program embedded systems using it; many frameworks are available for this purpose.

JavaScript is claimed to cause problems to developers because it is error prone and errors are difficult to spot. Developing rich featured and heavy applications are not so easy with JavaScript even though there is couple of libraries and frameworks such as JQuery, Backbone.js, and Kockout.js etc. To address these problems, new programming languages have been proposed. Usually they provide the missing functionality and implement language concepts of their own. Dart, CoffeeScript and TypeScript are examples of these.

All of these alternative languages address different problems of JavaScript and provide the solutions for them. For instance if you want to get rid of nasty callbacks and a prototype approach to object oriented programming and want some syntactic sugar, CoffeeScript is the best choice. For strict typing the Dart and TypeScript would be wise choice. There are IDE available for development in the Dart and TypeScript. There are a number of alternatives available, they do not replace the JavaScript but having higher level options available is always good.

The goal of this thesis is to evaluate some of these alternative web scripting languages. In order to achieve this goal comparisons concerning syntax, semantics, language features, security and performance are made. The contribution of this thesis will be an introduction to almost all the scripting languages but the focus will be in three languages. There will be an historical background, overview of new features and how they solve the problems of JavaScript.

The TODO application is also developed in all the considered languages to clear the idea of development in all these languages. It is a basic TODO application which has features such as add TODO item, remove, sort and edit. The data is saved in the local storage.

This thesis is organised as follows. Chapter 2 presents the scripting languages in general and the history of scripting languages, Chapter 3 is about JavaScript, its main features and drawbacks of JavaScript, Chapter 4 decribes the list of available scripting languages for web clients and need of popularity of web scripting languages, Chapter 5 explains the CoffeeScript and its features, Dart and TypeScript are elaborated in Chapter 6 and Chapter 7 respectively, Chapter 8 gives the robust and performance comparisons, Chapter 9 describes the TODO application features and its implementation. Finally, Chapter 10 gives the conclusion of the thesis. In the appendix the code of TODO application in all the languages is given.

2. SCRIPTING LANGUAGES

Scripting languages are computer-programming languages designed for "scripting" the operation of a computer. Early script languages were often called batch languages or job control languages. A script is more usually interpreted than compiled, but not always. (Wikipedia)
e.g. Perl, Tcl, Python, Rexx, Visual Basic, JavaScript, Unix shell

2.1. History of scripting languages

The Scripting languages originated as job control languages (JCL) in 1960’s IBM 360 had the Job Control Language. [1] The scripts were used to control the other programs; their responsibilities include launch compilation, execution and check return codes. In 1970's scripting languages got more powerful in the UNIX world. Shell programming, AWK, Tcl/Tk, Perl are examples of that. Shell programming named sh, began as a small collection of commands that were interpreted as calls to system subprograms that performed utility functions, such as file management and simple file filtering. Another scripting language is awk developed by Al Aho, Brain Kernighan and peter Weinberger at Bell Laboratories, awk began as a report generation language but later became a more general- purpose language. The Perl language, developed by Larry Wall, was originally a combination of sh and awk. [2] Perl has grown significantly since its beginnings and is now a powerful, although still somewhat primitive, programming language. Although it is still often called a scripting language, but in actual it is similar to the imperative language, because it is always compiled into an intermediate language before its execution. In 1990’s scripting languages become more common, used in faster computers, graphical user interfaces, component based programming and internet. In mid-1990s after the first graphical browsers appeared the use of Web exploded. The scripting languages were developed to write extensions to the browser and for controlling the browser, including JavaScript (a dialect of ECMAScript) or XUL. [3]

2.2. Properties of scripting languages

There is no hard and fast line between a "scripting language" and a "programming language". The boundary between them is somewhat blurry and difficult to demarcate. However, it is possible to highlight a few characteristics of scripting languages:

“Scripting languages are interpreted or bytecode-interpreted and never compiled to native code; the memory handling is done by a garbage collector and not by a program-

mer; include high-level data types, such as lists, associative arrays and so on; the execution environment can be integrated with the program being written; the scripting programs (or simply, scripts) can access modules written in lower-level languages, such as C."

Related to the above, usually in a "scripting language" the variables are not declared explicitly, and the types of the variables are scarcely declared. Some scripting languages (such as Javascript) are coerced type, and others (such as TypeScript and python) are strongly typed and raise exceptions on type mismatches.

Not every scripting language has the whole set of these features. For example, shell scripts cannot access C modules. But it's a scripting language nevertheless.

The main idea behind the scripting languages is their dynamic nature that allows treating data as a program and vice versa. The list of the scripting languages includes: shell, awk, Perl, TCL, Python, Java, Lisp and many others.

2.3. Scripting engine

In computer science, an interpreter normally means a computer program that executes, i.e. performs, instructions written in a programming language. While interpretation and compilation are the two principal means by which programming languages are implemented, these are not fully distinct categories, one of the reasons being that most interpreting systems also perform some translation work, just like compilers.

Essentially an interpreter (or scripting engine) is the component that is responsible for turning a script into machine code at execution time (as opposed to a compiler which creates machine code prior to execution time). See in the figure 2.1.

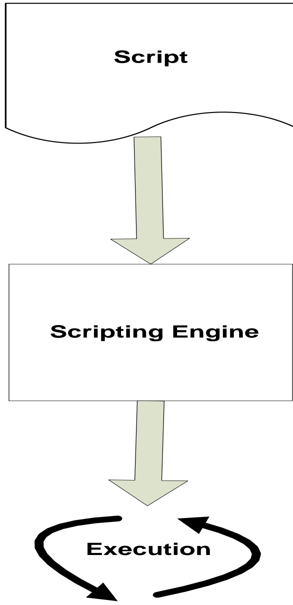

Figure2.1.*Scripting engine.*

Every scripting language has its own scripting engine, in order to implement the scripting engine, one should understand the scripting language’s nature and how it works. User needs to look at the system requirements, how a scripting engine will fit into the existing software architecture, to conclude which type of the scripting engine to use. Programmers will need a deep understanding to use scripting engine in ways that will provide similar results to simply writing executable code in a non-scripting language.

3. WHAT IS JAVASCRIPT?

JavaScript is a programming language that can be inserted into HTML pages, can be executed in all modern web browsers [17]. It is used to make the web pages interactive. It turns on your visitor's computer and does not require constant downloads from the website.

3.1. History

JavaScript was created in 10 days at Netscape by Brendan Eich on May 1995. It was not always known as JavaScript: the original name was Mocha, LiveScript was the official name for the language when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995, but it was renamed JavaScript[18] on December 4, 1995 when it was deployed in the Netscape browser version 2.0B3[19]. Netscape's Navigator, Microsoft's Internet Explorer and most other popular browsers support JavaScript.

Netscape introduced an implementation of the language for server-side scripting (SSJS) with Netscape Enterprise Server, first released in December, 1994 (soon after releasing JavaScript for browsers)[20][21]. Since the mid-2000s, there has been a proliferation of server-side JavaScript implementations. Node.js is one recent notable example of server-side JavaScript being used in real-world applications [22][23].

Later on Jan 1997, Microsoft implemented its own version of JavaScript known as the Jscript. Jscript is similar to JavaScript accepted that it adds a few more additional capabilities. Jscript is compatible with JavaScript 1.2. Microsoft also included server side JavaScript support with its **Internet Information Server (IIS).**

In November 1996, Netscape submitted JavaScript to **European Computer Manufacturer's Association (ECMA)** for consideration as an industry standard, and subsequent work resulted in the standardized version named ECMAScript. In June 1997, ECMA International published the first edition of the ECMA-262specification. A year later, in June 1998, some modifications were made to adapt it to the ISO/IEC-16262 standard, and the second edition was released. The third edition of ECMA-262 (published on December 1999) is the version most browsers currently use. [24]

Over time it was clear though that Microsoft had no intention of cooperating or implementing JS in IE, even though they had no competing proposal and they had a partial (and diverged at this point) implementation on the .NET server side.

In 2005 Jesse James Garrett introduced a term “Ajax” and a set of technologies in his paper in which JavaScript was the backbone, used to create web applications where data can be loaded in the background, avoiding the need for full page reloads and resulting in more dynamic applications. This resulted in a creation of many other open source libraries such as Prototype, jQuery, Dojo and Mootools.

All of this then brings us to today, with JavaScript entering a completely new and exciting cycle of evolution, innovation and standardisation, with new developments such as the Nodejs platform, allowing us to use JavaScript on the server-side, and HTML5 APIs to control user media, open up web sockets for always-on communication, get data on geographical location and device features such as accelerometer, and more. It is an exciting time to learn JavaScript.

Today, "JavaScript" is a trademark of Oracle Corporation.[25] It is used under license for technology invented and implemented by Netscape Communications and current entities such as the Mozilla Foundation.[26]

3.2. How it works?

Support for JavaScript is built right into all the web browsers like Internet Explorer, Safari, Firefox, Google Chrome, and Netscape. It is enabled by default on all the web browsers, therefore the JavaScripts code runs automatically when the website is visited. As it is the interpreted language, so no compiler is required to create usable code. There are plenty of editors available to write the code. The script can be written in the same file as the HTML but it is recommended that to write in a separate file (using .js extension helps identify them as JavaScript) thus it can reuse again easily on multiple pages of a website. The <script> tag is used to link the JavaScript with the HTML. The same JavaScript can then be added to several pages by adding the appropriate tag into each of the pages to set up the link.

3.3. Paradigms

JavaScript is such a flexible language that it can be used to write code that follows many radically different programming paradigms such as functional programming, object-oriented programming (OOP), imperative programming etc.

3.4. Functional programming in brief

Functional programming is a style of programming which models computations as the evaluation of expressions and avoids state and mutable data. In functional programming functions are first-class, which means they are treated as the any other values, they can be passed as arguments to other functions and can be assigned to any other variable as

well as can be used in any other context where values can be used. There’s typically no layer where you process the input, store state, arranges a sequence of statements, update the state, and decide about the next step.

3.5. JavaScript and Functional Programming

JavaScript is not the truly functional language like Haskell and Lisp but it supports some construct that are typical of functional language. A goof functional programming can be done in JavaScript if these constructs are used extensively. The followings are the functional programming aspects implementation, which are not native but there implementation will not cost much.

3.5.1. Anonymous functions

The anonymous function is a function which is defined without being bound to an identifier. JavaScript is familiar with this concept. The two most common ways to create a function in javascript are by using the function declaration or function operator. Anonymous functions are created using the function operator. Here is an example:

```
var sum = function(x, y) {
    return x + y;
}
```

3.5.2. High-order functions

High-order functions are functions which accept functions as arguments or return functions. JavaScript has these functional elements built-in for a long time. Here is a basic example:

```
funct(”someArgument” , function(x) {return x;});
function funct(a, foo){
    foo(a);//it will return a
}
```

If you’re using a function as a value that you pass around. Here’s an example that shows how to combine together various functions.

```
// Function to calculate a total
var total = function(x, y) {
   return x + y;
};
// Function to add taxes
var AddTaxes = function(x) {
  return x * 1.4;
};
//function that takes the other two functions as argument
var CalcTotalPlusTaxes = function (fnTotal, fnAddTaxes, x, y) {
    return fnAddTaxes(fnCalcTotal(x, y));
};
// Execution
var result = CalcTotalPlusTaxes(CalcTotal, AddTaxes, 40, 60);
alert(result);
```

Programme4.1. *Example of combine together various functions.*

Note that you can still invoke an anonymous function without resorting to intermediate variables, as in the following code:

```
alert( (function(x) {return x * 1.4;})(100));
```

3.5.3. Recursion

Another concept which is common in almost all the modern languages is the recursion. A function calls itself inside its body: Here is an simple example

```
function factorial(n) {
    if (n &lt;= 1) return 1;
    return n * factorial(n - 1);
}
```

This is a very popular example of calculating the factorial by using the recursion that is why the detail description is skipped.

3.5.4. Closure

A closure is a function called in one context that remembers variables defined in another context, the context in which it was defined. That’s why, in the following example (the really interesting stuff is occurring in the anonymous function), when we call the hello function it will alert the name that is the part of inside function.

```
function  hello(name){
        var text = “Hello” + name; //local variable
        var someAlert = function(){
           alert(text);
        }
        return someAlert;
}
var  say = hello(“Antti”);
say();
```

The above code has a closure because the anonymous function function() { alert(text); } is declared *inside* another function, hello() . In JavaScript, if you use the function keyword *inside* another function, you are creating a closure.

3.6. Object oriented paradigm in brief

The object-oriented paradigm is a programming paradigm that promotes the efficient design and development of software systems using reusable components that can be quickly and safely assembled into larger systems. [27] The basic unit of code that is a template for creation of object is a “class”. OO paradigm includes some unique concepts which overcome the drawbacks of fellow programming paradigms. In OOP, the emphasis is on data and not on procedures. General features of OO programming are: object, Classes, Data Abstraction, Data Encapsulation, Inheritance, Modularity, Polymorphism, message passing and dynamic binding.

3.7. Java Script and Object oriented Programming

3.7.1. Core Objects

JavaScript has many objects that are part of its core for example there are objects like Math, Object, Array and String. Following is the example to generate the random number by using the Math object in JavaScript

```
alert(Math.random());
```

3.7.2. Custom Objects

There is a significant difference between custom objects in OOP language and JavaScript because in JavaScript there are no classes, it just has objects whose blueprint are that of a dictionary of data and functions. When a new object is created in JavaScript, it has an empty dictionary user can fill with anything he/she like.

It requires a little more work to gain the functionality of OOP languages for instance inheritance, encapsulation and abstraction; with the help of two approaches it is possible to exceed the capability of native JavaScript such as: Prototypes and closures.

3.7.3. Prototype-based programming

To make a new object, you just call the "copy" method on an existing object. In a prototype-based language, an object can contain both data and behavior. It's a self-contained thing.

Prototype-based programming is a style of object-oriented programming in which classes are not present, and behavior reuse (known as inheritance in class-based languages) is accomplished through a process of cloning existing objects which serve as prototypes. This model is also known as class-less, prototype-oriented, or instance-based programming. (Wiki)

3.7.4. Object-orientation through Prototypes in JavaScript

Defining a class is as easy as defining a function. As there is no explicit way of defining the class in JavaScript therefore functions can be used to somewhat simulate classes, but in general JavaScript is a class-less language. In the example below we define a new class called Person.

```
function Person() { }
```

To define properties and methods for an object created using function(), you use the this keyword. The prototype model requires that you define the public structure of the class through the JavaScript *prototype* object. The following code sample shows how to rewrite the Person class to avoid a closure.

```
// Pseudo constructor
var Person = function(name, lastname, birthdate){
    this.initialize(name, lastname, birthdate);
}
// Members
Person.prototype.initialize(name, lastname, birthdate){
    this.Name = name;
    this.LastName = lastname;
    this.BirthDate = birthdate;
}
Person.prototype.getAge = function(){
    var today = new Date();
    var thisDay = today.getDate();
    var thisMonth = today.getMonth();
    var thisYear = today.getFullYear();
    var age = thisYear-this.BirthDate.getFullYear()-1;
    if (thisMonth > this.BirthDate.getMonth())
        age = age +1;
    else
       if (thisMonth == this.BirthDate.getMonth() &&
           thisDay >= this.BirthDate.getDate())
           age = age +1;
    return age;
}
```

Programme4.2. *Example of prototype in JS.*

In the above example the constructor and the members are clearly separated and constructor is always required. Because in JavaScript there is no private member keyword, you can have the getter and setter to define the properties but these backing fields are accessible from outside in anyway.

3.7.5. Inheritance

Through inheritance, one object can inherit the characteristics of another object; this allows an existing object to be extended and similar objects to share properties and behaviors [28].

```
var animal = {eat : true};
function Dog(name){
     this.name = name;
}
Dog.prototype = animal;
var dog = new Dog('tiger');
alert( dog.eat ) // true
```

Using the prototype, Dog inherits the properties of animal allowing the dog instance to use animal’s eat property.

3.7.6. Abstraction

Abstraction is hiding the details of a process/artifact to emphasize other (more important) aspects, details, or structure. This can be achieved by inheritance (specialization), or composition. JavaScript achieves composition by letting instances of classes be the values of the attributes of other objects.

The JavaScript Function class inherits from the Object class (this demonstrates specialization of the model). And the Function.prototype property is an instance of the object (this demonstrates composition)

```
var foo = function(){};
alert('foo is a Function: '+(foo instanceofFunction));
alert('foo.prototype is an Object:
'+(foo.prototypeinstanceofObject));
```

3.7.7. Encapsulation

Encapsulation is a language mechanism to restrict the access of object’s component [29]. The object is structured to be self-contained, everything is available internally that an object needs. The internal state of the object is not directly accessible externally except through the abstraction layer [30].

```
(function() {
   var x = '';
   function myFunction () {
     alert('Hello: ' + x);
   }
   x = 'Bob';
   myFunction();
   alert(typeof x);            // string
   alert(typeof myFunction);   // function
   })();
```

```
alert(typeof x);              // undefined
alert(typeof myFunction);     // undefined
```

Whatever you declare in that self invoking function is held in a separate scope. The variable x and the function myFunction() cannot be accessed from anywhere else. The code in other JavaScript files will not see them, for example, and it would be free to declare another function myFunction() without conflicts.

3.8. Why JavaScript is not enough?

3.8.1. No Module

Module development means separating the functionality of a program into independent parts. In JavaScript there is no import statement as so no namespace. The namespace is important in modularity. This is particularly annoying when dealing with many libraries that also depend on other libraries. It forces you to know all the dependencies up front and hurts the modularity of your application.

3.8.2. No Visibility Control

In JavaScript access modifier are not there, no private, protect and public modifiers like in Java that helps in hiding, reduce dependencies, understanding which code belongs to which class, and less risk of accidental side-effects etc, it helps in modularity as well. And it is enforced by the compiler so you get some static checking about this organization and you do not have to manage by yourself.

3.8.3. Weak Type System

In JavaScript there is no type system. While defining a variable, it is not necessary to specify the type of the variable and that variable can store any type of values like strings or number. For instance if you define a variable

```
var a;
a = “hello”;
.......
a = 10;
```

There will be no error in the above code. In JavaScript it does not have type system, therefore it is hard to pick up the other peoples code and look at the interface and it is difficult to see what the functions do. It would be rather easy if it has the types in the function signatures.

3.8.4. No Static checking

This is also related to type system, when there is a type system and a compiler that enforces types that means at compile time it can help you find a whole class of errors that a programmer might not find while running through the problem but similarly, if

there is an error in an else branch or in any branching code that does not run regularly or say some sort of error handling code that will be hard to manage if the code is getting bigger and bigger. It is possible if you write unit test and integration tests to test all the branches of the program but many programmers do not do that. Let consider the following example code

```
var arraySet = [];
if(arrayset.length == 0){ //typo mistake
        arraySet[1] = "hello" ;
}
else {
        for(var i = 0; i<arrayset.length ; i++){ //typo mistake
                   //do something
        }
}
```

In if statement there is a typing error or typing mistake that condition will never be true and the for loop will also not execute.

3.8.5. No support for generics

With generics, you can tell your program, and other developers sharing the code, that a list will only take the numbers or strings but it will not take the numbers and strings all together. For example

```
var myArray = new Array();
myArray[0]= "string 1";
myArray[1] = "string 2";
myArray[2]=5;
alert(myArray[2]);
alert(myArray[1]);
```

The above code is correct is in JavaScript but it should not be in the way it is. There is no typing system though no generics.

3.8.6. Not enough polymorphism

It is not possible in JavaScript to define equality or comparison for user defined types because JavaScript assumes that everything is a string (its default behaviors of JS). Moreover there are no standard protocols for hash codes; object as tables can be used but for the keys only the string can be used. The other problem is object's behavior in Boolean context is also not possible. Callable objects are only made possible by treating functions as objects, never the other way around.

3.8.7. Development of a large application is hard

JavaScript also has features (or, more commonly, it *lacks* features) that make development of large applications somewhat more challenging than they might be in other languages. Some of these are minor grievances—the way variable scope works, for example. Others are a bit more significant; for example, JavaScript has no built-in module system. This is a particular annoyance for programs that depend on third-party

libraries, since they have no good way of ensuring that library A will not conflict somehow with library B (for example, both libraries might try to create objects with the same name).

3.8.8. Maintainability is hard

JavaScript is increasingly used to develop the large and complex application. The problem is maintainability of such application because it should have a well-defined structure and developers should adhere to a strict coding discipline in order to avoid producing a festering pile of messy code where everything depends on everything and no boundaries can be found between modules.

3.8.9. Equality comparisons

The weak equality comparison in JavaScript has some confusing behavior and is often the source of confusing bugs. The example below is taken from JavaScript Garden's equality section [31] which delves into the issue in some depth.

```
""           ==   "0"// false
0            ==   ""// true
0            ==   "0"// true
false        ==   "false"// false
false        ==   "0"// true
false        ==   undefined     // false
false        ==   null// false
null         ==   undefined     // true
" \t\r\n"    ==   0// true
```

The reason behind this behavior is that the weak equality coerces types automatically, this is all pretty ambiguous and can lead to unexpected results and bugs.

3.8.10. Number property lookups

A flaw in JavaScript's parser means that the *dot notation* of numbers is interpreted as a floating point literal, rather than a property lookup. For example, the following JavaScript will cause a syntax error:

```
8.toString();
```

The javaScript's parser is looking for another number after the dot, and so raises an unexpected token error when it encounters toString().

3.8.11. Reserved words

Certain keywords in JavaScript are reserved for future versions of JavaScript, such as const, enum and class. Using these as variable names in your JavaScript programs can cause unpredictable results; some browsers will cope with them just fine, and others will choke. It is totally depends on the different implementation of browsers.

3.8.12. Global variables

By default the JavaScript program runs in the global scope, and by default any variable created is in global scope too. It is recommended that if you want to create a local variable then use the var keyword.

```
usersCount = 1;         // Global
var groupsCount = 2;    // Global

(function(){
pagesCount = 3;       // Global
var postsCount = 4;   // Local
})();
```

This is a bit of an odd decision since the vast majority of the time you'll be creating local variables not global, so why not make it the default behavior? As it stands, developers have to remember to put var statements before any variables they're initialized, or face weird bugs when variables accidentally conflict and overwrite each other.

4. SCRIPTING LANGUAGES FOR WEB CLIENTS

4.1. List of scripting languages for web client

There are quite a few changes coming to JavaScript in the recent versions of the language standard, for example, ECMAScript 6, but also simultaneously a number of languages started to appear near JavaScript that try to address the described issues in section 3.8 and add missing functionalities of JavaScript and compile to JavaScript. If somebody wants to use OOP concepts, or add strict typing, or want to avoid the callbacks then should definitely check these languages. Below is the list:

- Coffee Script
- Type Script
- Dart
- Haxe
- Roy
- Clojure Script
- Opal
- Iced Coffee Script
- Live Script
- Kaffeine
- ParenScript
- Fay
- Ceylon

4.1.1. TypeScript

It compiles to JavaScript. It is an open source and it is designed to develop large applications. TypeScript syntax is a superset of ECMA5 syntax. Any existing JS code is the TypeCcript code. TypeScript offers optional type annotations and static typing. It has classes, explicit interfaces and easier modules exports. Classes enable programmers to express common object-oriented patterns in a standard way, making features like inheritance more readable and interoperable. It supports header files which add type information to existing JavaScript libraries.

4.1.2. Dart

The goal of the Dart is “ultimately replace JavaScript as the lingua franca of web development on the open web platform” [4]. It has the ability to be more easily tooled large scale projects and better security features. It is possible to build more complex, full featured client side web applications. It is dynamically typed language and open source. It is basically class based, single inheritance, object oriented language.

It is an optional typing and supports abstract classes, interfaces and reified generics. Dart is intended to address JavaScript's problems (which Google's engineers felt could not be solved by evolving the language) while offering better performance, the ability "to be more easily tooled for large-scale projects" and better security features. [38]

4.1.3. CoffeeScript

It adds syntactic sugar to enhance the JavaScript readability inspired by Ruby, Python and Haskell. It is a weak type checking language and adds some features like list comprehension and pattern matching. The same code of JavaScript can be written in 1/3 fewer lines without affecting the performance.

4.1.4. Haxe

It is a high level multiplatform programming language. The single code file can compile into Adobe Flash applications, JavaScript programs, C++ standalone applications, PHP, Apache CGI, and NOdeJS server-side applications. [55] It is strict typed or static typed language. It supports classic object oriented features; it has classes and interfaces like Java. It supports modularity, generics, advanced type inference for all variables including method's arguments and return types. It is open source modern language.

4.1.5. Roy

It compiles to light-weight and readable JavaScript. Roy tries to output clean JS to help with debugging, performance and reasoning. [53] It is statically typed and a small functional language. It has features like pattern matching, Structural typing, Monad syntax, Simple tagged union and Compile time meta programming. The module support is designed to unify the many module standards the JavaScript community has created, including CommonJS Modules/1.0, Asynchronous Module Definitions (AMD), and browser-based global. [53] The main motivation to develop this language was “writing correct JavaScript is hard” [5].

4.1.6. Clojure Script

It can also compile to JavaScript. It runs on the Java Virtual Machine, Common Language Runtime, and JavaScript engines. It is designed to be a general purpose language, combining the approachability and interactive development of a scripting language with an efficient infrastructure for multi threaded programming. It provides an easy access to the Java frameworks, with optional type hints and type inference, to ensure that call to Java can avoid reflection [6].

4.1.7. Opal

It compiles Ruby to JavaScript and has a compiler which can run on any browser. It is strongly typed, higher order, strict, pure functional language. It has features like parameterized structure, unrestricted overloading, selective imports and modularization [7]. It can be used for concise, simple, concurrent, secure and dynamically distributed web applications.

4.1.8. Iced Coffee Script

It compiles to readable and pretty-printed pass through JavaScript lint without warnings. It is a superset of CoffeeScript, it has clean, readable, maintainable control flow for network and asynchronous operations. There are no callback pyramids. CoffeeScript is just JavaScript but iced coffee script is doing something deeper. It adds two new keywords: *await* and *defer*. They transform code for you so that you can write code in a synchronous style. In the generated JavaScript, await and defer produce nested functions. These additions simply powerfully streamline asynchronous control flow, both on the server and on the browser.

4.1.9. Live Script

It compiles to JavaScript and compatible with CoffeeScript. It is an open source language. LiveScript aims for increased expressiveness and code beauty. It has more features than CoffeeScript like it assist in functional style programming, it supports imperative and object oriented programming, and also has an optional class system with inheritance, calls to super and more [8].

4.1.10. Kaffeine

It is progressively enhanced JavaScript syntax. [54] It is an open source language. It supports packages and classes import, useful for browser applications. It avoids nice-to-haves, concentrate on small useful feature set and pragmatism. It is hackable, modular, extendable and testable.

4.1.11. ParenScript

It is a translator form an extend subset of Common Lisp to JavaScript. It is an open source language. It works with the native JavaScript data types. There are no new types introduced, and object prototypes are not touched. It introduced minimal overhead for advanced Common Lisp features. The generated code is almost as fast as hand-written JavaScript [9].

4.1.12. Fay

Fay compiles to JavaScript and it is statically typed, lazy, pure by default. It is a proper syntactic and semantic subset of Haskell [10]. It has a trivial foreign function interface to JavaScript.

4.1.13. Ceylon

It is a JVM language compile to JavaScript. Ceylon compiles to both Java and JavaScript virtual machines, bridging the gap between client and server. It has a powerful static type system but prevents many bugs [11]. It supports modularity. It is also open source.

For the detailed analysis and a discussion only three languages are chosen CoffeeScript, TypeScript and Dart. They are the most popular and used by many developers to build larger scale applications. It will be described how they relate to each other and JavaScript in the next sections.

4.2. Importance of language popularity

Programming Language popularity is important for having a large group of possible employees in any software company. To avoid the risk of proper planning of maintenance and development of new and existing softwares programming language should not be unpopular or unknown. Consequently, which language should be used for the projects is also dependent on the factor of the popularity of programming language no matter its web or desktop application.

To determine which programming language is most widely used, and what usage means varies by context is hard to measure. It is hard to collect the scientific data on this; therefore any kind of result cannot be seen as a true scientific proof in this regard.
One language may occupy the greater number of programmer hours, a different one have more line of code and a third utilize the most CPU time. There are the numerous methods of measuring the popularity of a programming language that have been proposed:

- Counting the number of job advertisement that mention the language [12]
- The number of books sold that teaches or describe the language [13]
- Estimates of the number of existing lines of code written in the language-which may underestimate languages not often found in public searches [4]
- Counts of language references (i-e the name of the language) found in web searches.

Now, considering the languages in web programming context, we can also take existing software ecosystems into account. Aforementioned, all these languages have a very close relationship because all of them are compiled down to the JavaScript. There are relatively more docs (the core docs themselves are great), more existing code, and more tools in CoffeeScript, than any other dialect. Just as the ecosystem of JavaScript is in turn bigger than that of CoffeeScript. There are already several CoffeeScript books in the making, for example. StackOverflow registers more than three thousand questions about CoffeeScript, while other dialects are in the single or low double digits. (And 130 thousand questions about JavaScript see the figure 3.1, to prove our point about CoffeeScript still being a minnow in the big JS Sea!) There is also an increasing array of tools where CoffeeScript support comes out of the box, e.g. connect-assets.

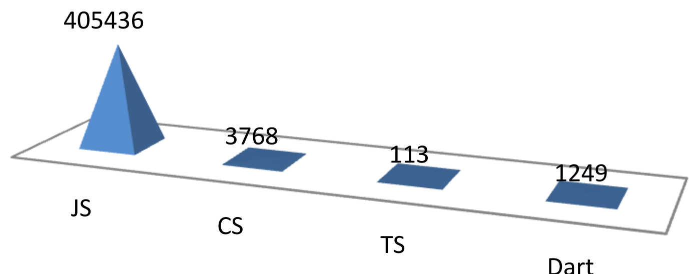

Figure3.1.*Tags on Stackoverflow*

TypeScript is fully opened sourced and is already being supported by companies other than Microsoft. As TypeScript is the superset of JavaScript it has access to all the eco-systems in JavaScript. There exists a tight integration between TypeScript and Visual Studio, All the windows store apps with JavaScript can be used as a template in windows app with TypeScript with little modifications.

Dart is also the open source and developed by the Google. One indicator of popularity is the TIOBE Programming Community Index [15]. It is a programming language ranking based on skilled engineers, courses, third party vendors and search engine ratings [16]. Dart recently crept up into the TIOBE Programming Community Index number 43 for language popularity in October 2012. As long the major browsers don't support Dart, therefore there will not be a change in the popularity of Dart.

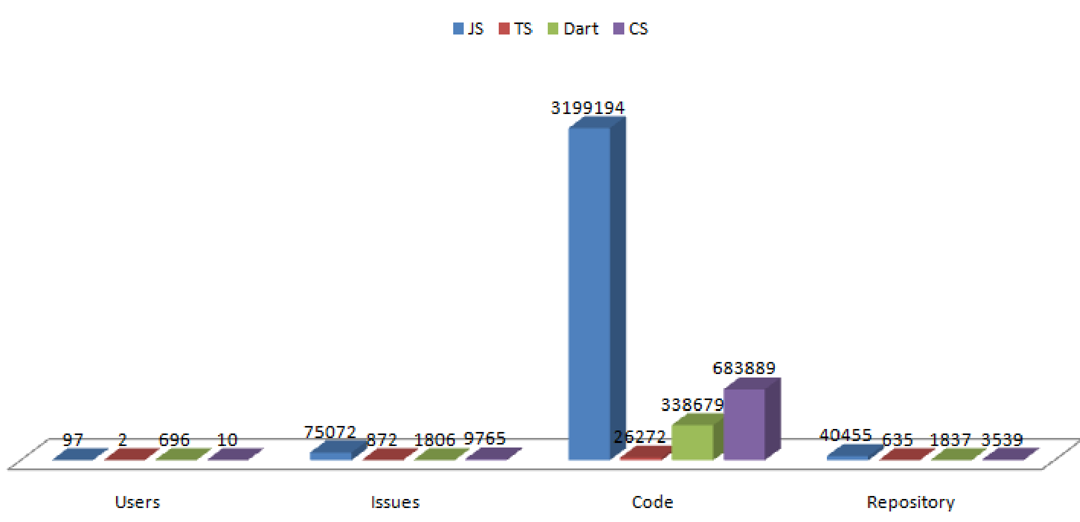

Figure3.2.*Popularity on Github*

In the figure 3.2 all the numbers are taken from the github, it shows that there are number of users of Dart even though it is quite a new language. As there are many users

that’s why they log many issues. If we see the code files and the repositories available then the CoffeeScript has over three thousand repositories. JavaScript has many users and code files because it is an old language even there are a number of web applications that are developed in JS and it is more popular in all.

5. WHAT IS COFFEESCRIPT?

CoffeeScript is a programming language that compiles to JavaScript, it means the programmer writes his code in CoffeeScript compiler compiles it and generates the JavaScript out of it that can be served up to the web browser. The golden rule of CoffeeScript: “it is just a JavaScript” [32]. There is no interpretation at run time therefore the existing JavaScript libraries can be used in CoffeeScript and vise-versa.

5.1. History

CoffeeScript is almost four years old. On December 13, 2009, Jeremy Ashkenas made the first Git commit of CoffeeScript with the comment: "initial commit of the mystery language."[33] It was first developed on Christmas Day 2009 in Ruby using lexer and parser libraries. On February 21, 2010, the 0.5 version was released the CoffeeScript compiler was rewritten in CoffeeScript and since then has been self-hosting. Ashkenas takes great pride in knowing that an update to the CoffeeScript compiler is compiled with (the previous version of) CoffeeScript and the resulting code (the new version of the compiler) then compile the source code again. This self-hosing / bootstrapping process highlights how mature the CoffeeScript compiler already is. [34] On December 24, 2010, Ashkenas announced the release of stable 1.0.0 to Hacker News, the site where the project was announced for the first time. [35][36]

5.2. How it works?

The code written in CoffeeScript first compiles and generates a JavaScript code out of it that can execute in any host (browser) or in any JavaScript engine. The compiler is written in CofeeScript and run on Node.js. There is another way to run the CoffeeScript code. In the HTML page just write your CS code between:

```
<script type="text/coffeescript">
    ...
</script>
```

Or include your CS file:

```
<script type="text/coffeescript" src="demo.cs"></script>
```

You also have to include the compress and minified version of the compiler as “coffee-script.js”.

```
<script type="text/javascript" src="coffee-script.js"></script>
```

But this is not recommended way to use and run the CoffeeScript code. The other and recommended way is first compiled the CS code and include the generated .js file in HTML page.

5.3. Features it adds to JavaScript

As it is already mentioned that CoffeeScript is just a JavaScript and has all the features that JavaScript has therefore, it is useless to describe all of them again. Why to invent the wheel again? Rather to define the incentives that it includes to JavaScript.
In brief, CoffeeScript lets the programmer to write the same program with less lines of code than JavaScript. It's got a lot of sorts of lightweight add-ons like Ruby style string interpolation and Python style list comprehension. It makes a lot of common tasks much easier than JavaScript. Pass around a lot of functions, so CoffeeScript provides a very brief way of expressing those. In the next section, all of these good parts of CoffeeScript are illustrated with examples.

First of all, we will see how a JavaScript code can be converted to the working CoffeeScript code with simple manipulations of the existing code. It includes the following steps:

- Remove the semicolon at the end of every statement because there is no statement termination in CoffeeScript.
- Remove the var statements
- Remove the curly braces and use the indentation instead, CoffeeScript objects are whitespace dependent.
- Remove the return statements because return is the last expression and since everything is an expression.

  ```
  a= Foo(x) ->
     If x > 0
       42
  ```
  In the above example if the value of x will be zero or less than zero, it will assign undefined to a. But if the condition is true it will return the 42.
- Remove the "function" keyword because there is no such keyword use "->" instead.

So here is a simple JavaScript code

```
var sum = function(y) {
     return y+y;
}
```

In CoffeeScript it will be written as:

```
sum = (y) ->y+y
```

5.3.1. Inheritance with CoffeeScript

Classes are simpler in CoffeeScript than JavaScript because they are not prototypical here. Classical-looking inheritance is a case of the extends keyword, and the super () method call to invoke the parent class' behavior:

```
class Human
     eat: -> console.log "food"

class Boy extends human
     eat: -> console.log "Meat"

class Girl extends human
     eat: ->
             super()
             console.log 'Girl wants fun on weekends.'

new human().eat()
new Boy().eat()
new Girl().eat()
```

Programme5.1. *Example of inheritance in CS.*

You'll notice that in the example programme 5.1, we're using the super () keyword. Behind the scenes, this is translated into a function call on the class' parent prototype, invoked in the current context. In this case it will be Girl._super._eat.call which is the Human’s.

5.3.2. List comprehensions

You have an array of objects and want to map them to another array; in those scenarios you can use the list comprehensions. In CoffeeScript, a simple for loop can avoid recalculating list.length on every iteration. However, CoffeeScript also overloads the for loop with the ability to do Comprehensions. Comprehensions allow you to manipulate the items being iterated over.

```
teacher_course = [{name: "Imed", course: "programming 1"},
{name: "Tapio",course: "Artificial Intelligence"},
{name: "Terhi",course: "Utilizations of DS"},
{name: "Henri",course: "Analysis of Algorithms"},
{name: "Maarit", course: "Principles of programming languages"}]

names = (instructor.nameForInstructorInTeacher_course)
# => [ 'Imed’, ' Tapio ', ’Terhi ', ' Henri ', ’Maarit’]
```

5.3.3. String interpolation

CoffeeScript's multi-line string blocks are similar to Pythons triple-quoted strings (with interpolation of variables done with #{varname}):

```
hello = """
     multi-line
     string
     block. #{varname}
     """
```

or

```
name=“Jermy”
alert (“I am #{name}”)
```

5.3.4. Splats (…)

CoffeeScript adds splats (...), which is a way of dealing with variable number of arguments in JavaScript methods.

```
course = (programming1, OOP...) ->
  print programming1, OOP
```

5.3.5. Number property lookup

In JavaScript the dot notation after the number is interpreted as floating point (see section 4.8.10). The solution to this problem is to use either parenthesis or add an additional dot.

```
(6).toString()
6..toString()
```

CoffeeScript parser will automatically deal with this issue while detecting the double dots whenever the programmer accesses the properties of numbers.

5.3.6. Reserved words

If a programmer uses the reserved words an object’s property, CoffeeScript neatly side steps this issue, by detecting, and escaping it if necessary.

For example, let's say programmer was to use the reserved keyword class as a property on an object, CoffeeScript might look like this:

```
myObj = {
     remove: "keyword detected!"
}
myObj.class = ->
```

The CoffeeScript parser notices, there is a reserved keyword, and quotes it for you:

```
var myObj;
myObj = {
     "remove": "keyword detected!"
};
myObj["class"] = function() {};
```

5.3.7. Global variables

By default every variable that is created is local. It is very difficult to create the global variables implicitly. There is an explicit way to create global variables by assigning them as properties on window.

This is how the global variables are created

```
class window.Teacher
constructor: ->
```

The var keyword is reserved in CoffeeScript and will trigger a syntax error if used. Let's have a look at an example of CoffeeScript's variable assignment [37]:

```
outerScope = true
do ->
innerScope = true
```

Compiles down to:

```
var outerScope;
outerScope = true;
(function() {
     var innerScope;
     return innerScope = true;
})();
```

Notice how CoffeeScript initializes variables (using var) automatically in the context their first used. Whilst it's impossible to shadow outer variables, you can still refer to and access them. You need to watch out for this, be careful that you're not reusing the name of an external variable accidentally if you're writing a deeply nested function or class.

5.3.8. Compile time checking

CoffeScript has the ability to catch the syntax errors on compile time that is really helpful in cutting down on production issues. Developers who are accustomed to writing JavaScript, Ruby, Python or any other dynamically typed languages may feel that this is unnecessary but coming from a background of strongly typed languages, we believe that compile time safety is invaluable.

5.3.9. Existential Operators

In JavaScript programmers have to check the null and undefined values by themselves. CoffeeScript provide the “?” operator which can use on a single variable even in a long chain of property accessors to check if a given property is not null and is defined. Here is an example code:

```
student =
```

```
name: "franklin Jones"
age: 18

#Using the existential operator
console.log student.non?.existent.property
```

It will check whether the property “non” is defined by the student or not. If all of the properties exist then you'll get the expected result, if the chain is broken, undefined is returned instead of the TypeError that would be raised otherwise. Existential operator can also be used on methods:

```
console.log student.name.reverse?()
```

6. WHAT IS DART?

Dart is an open-source web programming language. The goal of Dart is "ultimately to replace JavaScript as the lingua franca of web development on the open web platform." [38] Dart has the ability to build more complex, full featured client side web applications. It is class based, single inheritance, object oriented language. It is an optional typing and supports abstract classes, interfaces and reified generics. Dart is intended to address JavaScript's problems (which Google's engineers felt could not be solved by evolving the language) while offering better performance, the ability "to be more easily tooled for large-scale projects" and better security features. [38]

6.1. History

Dart is developed by Google, Google had announced one more programming language in history (in 2009) Go when it was not satisfied with the working of C or C++, they released Dart at the GOTO conference in Aarhus, 2011 October 10–12.

6.2. How it works?

There are three ways to run the Dart application. First, The Dart VM reads and executes source code, which means there is no compile step between edit and run. As with other popular scripting languages, it’s very quick to iterate with Dart. The Dart VM runs on the command line and servers, and can be embedded into browsers. Second, it can compile to JavaScript like other languages with the help of dart2js compiler and can run in web browsers. Third, the Dart SDK ships with a version of the Chromium web browser modified to include a Dart virtual machine. This browser can run Dart code directly without compilation to JavaScript. It is currently not intended for general-purpose use, but rather as a development tool for Dart applications. [39]

6.3. Features it adds to JavaScript

6.3.1. Optionally typed

Dart has the type annotations, but it is not strictly typed language. Types provide many benefits such as it is much easier for people to read your code if it has judiciously placed type annotations, tools can leverage type annotations in various ways. In particu-

lar, they can help provide nice features such as name completion and improved navigation in IDEs.

Dart's inclusion of an optional type system means you can use type annotations when you want, or skip them when that's easier. The big advantage is it can be used as static typing and if the programmer makes any mistake in types and pass the bad argument to any library while writing the code, checked mode will detect that at the point where you made the mistake. Dart rules are much less strict. It is recommended to use the type where they make sense. The most valuable thing a programmer can do is add types to the headers of public members of your libraries. Next, do the same for the private ones. Even if nobody else has to maintain the code it will be helpful if programmer leave the code and come back in a few weeks or months. In both cases, he doesn't necessarily have to add types of the bodies of methods or functions. Users of the library get value from type signatures, even if they are not 100% accurate.

On the other hand dart is dynamically typed language too, the programmer can stop the static checker or compile time checking of the code by using the type "dynamic" which is default type given when no type is explicitly given by the programmer.

The spec says: Dart programs may be statically checked. The static checker will report some violations of the type rules, but such violations do not abort compilation or preclude execution.

6.3.2. Reified generics

Dart supports reified generics. Think of generics as type annotations for your collections. With generics, you can tell your program and other developers sharing the code that a List of strings will only contain the string elements. Objects of generic type carry their type arguments with them at run time. Passing type arguments to a constructor of a generic type is a runtime operation.

As it is described earlier that the types are optional so if you don't want the types the generics will not force you to.
For example

```
new List();
```

and

```
new List<num>();
```

They both are correct but in the second, the list will only store the numbers.

6.3.3. Dart is purely object oriented

The Dart language is clear: everything is an object. It is easy to explain how everything works without having to deal with primitives as a special case. Even calling + on two

numbers is modelled as a method call. Numbers, Booleans, and even null are all objects. [40]

Dart lets you put any number of public classes into a file. It is not forced to have the name of the file is same of the class name and it is also not forced to put one public class in one file. It is possible to organize the project files and content in any way you want.

6.3.4. Closures and lexically scoped functions

Closure functions that can naturally access variables in their enclosing scope without having to write verbose anonymous inner classes.

Here is an example of a closure in action. The makeAdder function returns a function that closes around makeAdder‘s parameter. [40]

```
makeAdder(int x) {
     adder(int y) => x + y;
     return adder;
}

main() {
     var add2 = makeAdder(2);
     var result = add2(3);
     print(result); // 5
}
```

You can use simplified makeAdder by returning an anonymous function:

```
makeAdder(int x) {
     return (int y) => x + y;
}
```

6.3.5. Dart has mixins

In object-oriented programming languages, a mixin is a class which contains a combination of methods from other classes. How such combination is done depends on language, but it is not by inheritance. [52] If a combination contains all methods of combined classes it is equivalent to multiple inheritance. No need to pollute the inheritance hierarchy with utility classes. Use Dart’s mixins to slide in functionality that is clearly not an is-a relationship. [40] Dart supports a basic form of mixins as of the M3 release in early 2013 [41]. The language designers expect to expand on mixin’s abilities in future versions of the language.

Classes that extend an object, don’t have constructors, and don’t call super can be a mixin, it should come as no surprise that an abstract class (with a few restrictions) is itself a mixin. Here is an example of a Persistable mixin [40]:

```
abstract class Persistable {
     save() { ... }
```

```
        load() { ... }
        toJson();
}

class Hug extends Object with Persistable {
  Map toJson() => {'strength':10};
}

main() {
        var embrace = new Hug();
        embrace.save();
}
```

Restrictions on mixin definitions include:

1. Must not declare a constructor
2. Super class is an Object
3. Contains no calls to super

6.3.6. Building large and complex applications

Dart scales from small scripts to large, complex apps. [42] You can quickly write prototypes that evolve rapidly, and you also have access to advanced tools, reliable libraries, and good software engineering techniques. Web development is very much an iterative process. With the reload button acting as your compiler, building the seed of a web app has been often a fun experience of writing a few functions just to experiment. As the idea grows, you can add more code and structure. Thanks to Dart's support for top-level functions, optional types, classes, and libraries, your Dart programs can start small and grow over time. Tools such as Dart Editor help you refactor and navigate your code as it evolves. [42]

6.3.7. Concurrency support with isolation

A Thread is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Dart is a single threaded programming language. Each isolate is a unit of work. It has its own memory allocation. Sharing memory between isolates is not possible Dart support the concurrent execution with the help of isolation (processes without overhead). Each isolate can pass over messages to the others. When an isolate receives a message it processes it in a way similar to events handling. Isolates allow a single app to use multi-core computers effectively.

6.3.8. Snapshots

Snapshots are supposed to provide nice startup improvements for large applications; the whole application code does not require to be downloaded every time you start the application. Dart addresses this with the heap snapshot feature. An application's heap is walked and all objects are written to a file. Every time when the application code is loaded, just before calling the main, it takes the snapshot of the heap and the Dart virtual

machine use that the snapshot file to quickly load an application. The snapshot facility is also used to serialize object graphs sent between Isolates in the Dart VM. [43]

6.3.9. Reliability

Every Isolate is single threaded; splitting up the application into multiple and independent processes or isolates helps in achieving reliability. For example if one isolates crashes it will not affect the others and it can restart again.

6.3.10. Security

It is also related to isolates. The code that is not trustworthy can be run in isolate and the communication with that isolate must do with the message passing, which will be enhanced with the capability-style mechanism that permits which isolates will communicate of which port. The isolate must assign a port to send messages. The communication between isolates is not possible without ports.

6.3.11. Best usage of memory

Each isolate's heap is standalone; all objects in that heap clearly belong to that isolate. When one isolate is launched in memory and it finishes its task can be deallocated from the memory in one go, there is no need to call the garbage collector. There is one more benefit related to splitting the application into small isolates, by this way they take the less memory than the whole application take as full. Each heap is governed by its own GC with the effect that a full GC run in one Isolate only stop the world in that Isolate, the other Isolates won't notice. Hence having one heap per isolate improves the modularity: each Isolate controls its own GC pause behavior and is not affected by some other component.

6.3.12. Dart supports code sharing

With the Dart package manager (pub) and language features such as libraries, you can easily discover, install, and integrate code from across the web and the enterprise. And it was not possible in the traditional web programming workflows.

6.3.13. Global namespace

In Dart there is a possibility to make the namespaces; you can develop the global namespaces in the 'library' scope. You have a keyword "library" and only what is public is visible outside of it. A library can be made of multiple files, and contain multiple classes and functions. Once you have defined your own libraries you can use them in your program with the help of 'import' statement. Then the main () function that is the starting point of the application.

7. WHAT IS TYPESCRIPT?

TypeScript is a syntactic sugar for JavaScript. TypeScript syntax is a superset of ECMA5 syntax. Any existing JS code is the TypeCcript code. TypeScript offers optional type annotations and static typing. It has classes, explicit interfaces and easier modules exports. Classes enable programmers to express common object-oriented patterns in a standard way, making features like inheritance more readable and interoperable.

7.1. History

TypeScript is developed by Microsoft. The first preview build of TypeScript was launched in October 2012. The most publicly recognizable name behind TypeScript is Microsoft Technical Fellow Anders Hejlsberg, the father of C# and Turbo Pascal. But Hejlsberg isn't the one who came up with the idea for TypeScript. TypeScript is actually the product of a team of about 50 people, headed by Microsoft Technical Fellow Steve Lucco. [44]

Typescript is the open source language that is available under an Apache 2.0 open-source license.

7.2. How it works?

TypeScript is compiled, rather than interpreted. Typescript has a compiler tsc that is written in Typescript compiles the Typescript code and generates the idiomatic JavaScript that can execute in any host (browser) or in any JavaScript engine. There is also an alpha version of a client-side compiler in JavaScript, which executes TypeScript code on the fly, upon page load.[45] Code can be written between:

```
<script type="text/typescript">
    ...
</script>
```

or include the TS file:

```
<script type="text/typescript" src="demo.ts"></script>
```

And include these two js files also:

```
<script type="text/javascript" src="typescript.min.js"></script>
<script type="text/javascript"
src="typescript.compile.min.js"></script>
```

All JavaScript code is the TypeScript code simply copy and paste, it will work. All JavaScript libraries work with TypeScript Node.js, JQuery, Backbone etc. Microsoft provides the plugin for Visual Studio 2012 and WebMatrix that helps in code completion, refactoring and debugging of TypeScript code. The online Cloud9 IDE also supports TypeScript.

7.3. Features it adds to JavaScript

The features TypeScript adds to JavaScript development are small, but yield large benefits to .NET developers who are accustomed to similar features in the languages they use for regular Windows application development. [46]

7.3.1. Optionally typed

TypeScript adds optional static types to JavaScript. Types are used to place static constraints on program entities such as functions, variables, and properties so that compilers and development tools can offer better verification and assistance during software development. These type annotations are like the JSDoc comments found in the Closure system [46], but in TypeScript they are integrated directly into the language syntax. This integration makes the code more readable and reduces the maintenance cost of synchronizing typed annotations with their corresponding variables. [47] TypeScript provides static typing through type annotations to enable type checking at compile-time. This is optional and can be ignored to use the regular dynamic typing of JavaScript. [48]

```
function square(x: number): number {
    return x*x;
}
```

TypeScript's static compile-time type system closely models the dynamic run-time type system of JavaScript, allowing programmers to accurately express the type relationships that are expected to exist when their programs run and have those assumptions prevalidated by the TypeScript compiler. TypeScript's type analysis occurs entirely at compile-time and adds no run-time overhead to program execution.

7.3.2. Type inference

The TypeScript type system enables programmers to express limits on the capabilities of JavaScript objects, and to use tools that enforce these limits. To minimize the number of annotations needed for tools to become useful, the TypeScript type system makes extensive use of type inference. For example, from the following statement, TypeScript will infer that the variable 'i' has the type string.

```
var i =”abc”;
```

Let’s take another example of plain JS code

```
function process(y) {
        y.name = “foo”;
        var x= y*y;
        alert(y);
}
```

TypeScript takes the data type of variable y as “any”. You can perform any function on that, assign any value to it or pass to any other function. In the above code if you put the data type in the parameter like function process(y:string){…} then it will give an error on line 2 because there is no property name on string available. Therefore, if you want to give the specific type to a variable it will allow you to do the certain operations.
It is possible to have the user defined types in TypeScrtipt. For example

```
interface person{
        a:string;
        c?=:number;
        clear() : void;
        sum(x:number,y:number): number;
}
```

The above interface have two functions and two variables, a class that will implement this interface must have to overload the both functions and assign the value to the first variable but the second variable is optional (? is for optional).
You can have the function overloading which is not allowed in JS. If a programmer implements the above interface in his code somewhere and mistype the sum() function as sumx() JS will not prompt any error. Imagine there are 200 lines of code and programmer does not know at which line the error is occurring. But in TS at compile time it will show the error and the line number where it is occurring. This is all possible with the help of type inference in TS.

7.3.3. Object Orientation

The great strength on TypeScript is to introduce the concept of classes and inheritance. In JavaScript inheritance is done by the prototypes as described in the section 4.7, but it tends to be more verbose, a bit confusing and far from elegant. Following is the simple class example in TypeScript

```
class Person {
     name: string;
     constructor(name: string) {
     this.name = name;
     }
     sayHello() {
            alert("Hello, my name is " + this.name);
```

```
    }
}
```

This compiles to the following JavaScript code

```
Var Person = (function () {
    function Person(name) {
        this.name = name;
    }
    Person.prototype.sayHello = function () {
        alert("Hello, my name is " + this.name);
    };
    return Person;
})();
```

7.3.4. Inheritance

In TypeScript inheritance is achieved by simply using the “extend” keyword to extend from the base class. Consider a simple example where inheriting the student class from the person class. It is possible to override the methods and a derived class will inherit all the public members from its base class.

```
class person {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
    sayHello() {
        alert("Hello, my name is " + this.name);
    }
}
class Student extends Person {
    sayHello() {
        super.sayHello();
        alert("and I am a student!");
    }
}
var student: Student = new Student("Antti");
student.sayHello();
```

Programme7.1. *Example of inheritance in TS.*

7.3.5. Modularization and multi-file

In JavaScript to hide the names and encapsulate the private data the closures are used as described in section 4.5.4. TypeScript provides built-in support for CommonJS and AMD modules. It is quite simple to import and export TypeScript files, just like you would in a server-side language. Importing JavaScript libraries is a bit trickier, but can still be done.

7.3.6. Scalable application structuring

JavaScript was originally designed to be a client-side scripting language for web pages, and for many years it was limited to event handlers that scripted a Document Object

Model (DOM). As a result, JavaScript is missing many of the features necessary to be able to productively write and maintain large-scale applications, namely those that create distinct contracts between components and developers. How this issue is resolved in TypeScript is to have Classes, modules and interfaces that enable clear contract between components.

7.3.7. Open and Interoperable

All JavaScript code is TypeScript. You can copy-and-paste any JavaScript code into the TypeScript file because TypeScript produces standards-compliant JavaScript, TypeScript is consistent with our commitment to ensuring that developers can use the same markup and script for a more interoperable web: the output of the TypeScript compiler runs on any browser, in any host, on any operating system. [49]

TypeScript is open. The language is available under the Open Web Foundation's Final Specification Agreement (OWFa 1.0). Microsoft's implementation of the compiler is also available on CodePlex (with git) under the Apache 2.0 license.

7.3.8. Build and maintain large applications

TypeScript is designed to meet the need of the JavaScript programming team that build and maintain the large applications such as web applications. Web applications are becoming an increasingly important part of everyday computing. TypeScript helps programmer to define the interfaces between software components and to gain insight into the behavior of existing JavaScript libraries. It also enables teams to reduce the name conflicts by organizing their codes into dynamically –loadable modules.

7.3.9. Refactoring

Refactoring is another area where TypeScript has a distinct advantage over JavaScript. For example, when you change the name of a class or interface in TypeScript, it's possible to find almost all uses of that class or interface in the application code and change the name as needed. Though, as with many things, there are exceptions. For example, while you can find changes in a name as part of the code in a class or interface, you won't find it within an object literal. However, the compiler will find the required name change and tell you about it, so your code will still be fixed before it reaches the production environment.

8. COMPARISONS

8.1. Performance comparison

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

- Bill Gates (co-founder of Microsoft)

CoffeScript compiles directly to JavaScript, meaning that there is always a one to one equivalent in JS for any CoffeeScript source. Therefore, its maximum possible speed equals to the speed of JavaScript.

Dart is claimed to be more efficient than JavaScript. There are a few reasons that are as follows:

In languages such as Ruby or JavaScript, the program structure can change at runtime. Classes can get a new method, functions can be eval()'ed into existence, and more. This makes it harder for runtime to optimize their code, because the structure is never guaranteed to be set.

Prototypal inheritance is harder to optimize than more traditional class-based languages. It is suspected this is because there are many years of research and implementation experience for class-based VMs.

Interestingly, V8 (Chrome's JavaScript engine) uses hidden classes as part of its optimization strategy. Of course, JS doesn't have classes, so the object layout is more complicated in V8.

Object layout in V8 requires a minimum of 3 words in the header. In contrast, the Dart VM requires just 1 word in the header. The size and structure of a Dart object are known at compile time. This is very useful for VM designers.

Another example: in Dart, there are real lists (aka arrays). You can have a fixed length list, which is easier to optimize than JavaScript's not-really-arrays and always variable lengths.

Another performance dimension is start-up time. As web apps get more complex, the number of lines of code goes up. The design of JavaScript makes it harder to optimize startup, because parsing and loading the code also executes the code. In Dart, the language has been carefully designed to make it quick to parse. Dart does not execute code as it loads and parses the files.

This also means Dart VMs can cache a binary representation of the parsed files (known as a snapshot) for even quicker startup.

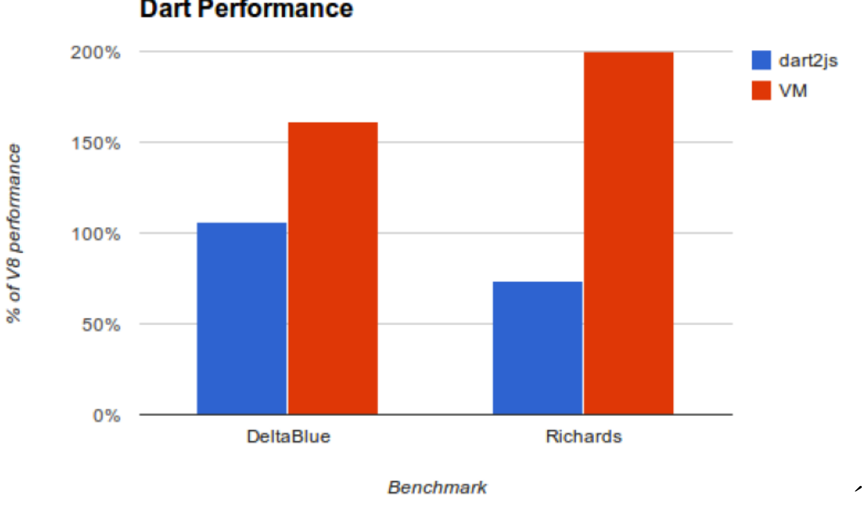

Figure3.2.*Dart2js and VM benchmark [50]*

The chart in the figure is currently showing chart Dart has surpassed JavaScript performance on two significant Octane-based*benchmarks -- Richards and DeltaBlue

- DeltaBlue is a one-way constraint solver, originally written in Smalltalk by John Maloney and Mario Wolczko. The main focus in DeltaBlue is on polymorphism and object-oriented programming. [50]
- Richards is an OS kernel simulation benchmark, originally written in BCPL by Martin Richards. The main focus in Richards is on property access and calling functions and methods. [50]

*Octane is a modern benchmark that measures a JavaScript engine's performance by running a suite of tests representative of today's complex and demanding web applications. The octane's goal is to measure the performance of JavaScript code found in large, real-world web applications.

8.2. Robust comparison

Strict mode is a new feature of ECMAScript 5 that allows to run JavaScript program or a function to run in a strict Context. What does this strict mode do? This strict context throws more exceptions and warnings than the normal context. It gives indications to

the developer when they are making mistakes in the code or straying from the best practices. It helps in reducing the bugs, improve performance and increase security.

In Coffee Script all these things check at the compile time. The CoffeeScript compiler does the strict mode syntax checks itself, at compile time. This is good. You don't want to wait until you run the generated JavaScript output to discover your errors. For example: You can't accidentally create a variable. This feature provides stronger security.

Dart has the isolate library which supports spawning and communication with isolates. As described earlier it is a mechanism for concurrency. Security can be achieved with the isolates too.

9. APPLICATION

The TODO application is developed in all the considered languages to do the deep analysis. First of all we should define the features that will be present in the to-do list then there is the implementation description in all languages. TODO application has the following features:

- Add a new to-do item and its priority (low, high etc)
- Remove a to-do item
- Edit a to-do item/priority
- Sort the to-do list
- Structure object to store the to-do item and its priority
- Save the to-do object using browser's local storage
- Clear the entire list

If we think about TODO list what we need is:

- An Input to place our to-do
- A button to add our to-do
- A button to delete that to-do
- A link to clear all to-do's
- A placeholder unordered list where out to-do's will be placed on the list items.

All the above items are in HTML and with the help of JavaScript we can fill all this up with the dynamic content. In appendex1, there is the code of html and in appendix 2 the css code is presented that is pretty straight forward and self-explanatory nothing too fancy.

In the next sections the implementation details are explained in different languages.

9.1. TODO application in JS

The main idea how it is developed in JavaScript is: To store the data local storage is used, the good thing about local storage is that you can save the data on the user's computer so that when they reload the page all of their todo's will still be there and local storage actually quite simple when it comes to saving the data and making it available on the page.

- Each to-do item will be stored in local storage with the unique key of "todo-*ID*"
- Because there is an ability to sort the todo-list therefore to track the order of the list an array is used, which then get saved in the local storage with the key of "todo-orders"
- One more variable is stored in the local storage "todo-counter" to keep track of unique ID, it will remember what the next number should be.

In the JavaScript file there are all the functions to add, remove, edit and clear all etc. In appendix 2 you can find the code of JavaScript file where first there is a declaration of variables up top. When a user enters something in the input field and press the "Add to List" button, the submit event will be announced to the app that form has been submitted using the $.publish() .

9.1.1. Add a new todo item

It first checks if the value of the field that is submitted is blank or not, if there is some value it retrieves that value and store it in local storage with the key "todo-uniqueId", which is whatever the value of i is at that moment. As it is described earlier about the need of "todo-counter", it will save the same value in this counter variable for future use. Afterwards, it appends the new to-do item and its priority to the to-do list, displays it and empties the input field.

On the completion of the add functionality, it publishes "/regenerate-list" which saves the order of the to-do items. It empties the order array, go through the item list elements, get the ID and add to the array, then save it to the "todo-orders".

9.1.2. To remove an item from the list

As we want to catch the click event on the current item and the future remove link that gets created for that $.delegate is used. It announces the "/publish/" event then it retrieves the ID of the parent of the clicked element and removes the entry in the local storage based on it. It then fades out the list item and remove it from the DOM and publish "/regenerate-list/" to update "todo-orders".

9.1.3. The remove button

It listens the mouseout/mouseover event on the list elements and call fadeIn() or fadeout() appropriately. They are all of JQuery functions.

9.1.4. Clear all list

It catches the click event and publish "/clear-all/", which resets the order array back to zero, clear the local storage and remove all list elements from the DOM.

9.1.5. Edit and save the item

We use the $.inlineEdit() function of JQuery on save, it retrieves the ID of the parent of the edited element and re-save the edited value.

9.1.6. Reorder and save item

For this functionality the $.sortable() of JQuery is used which is pretty simple and then on the “stop” event, it publishes “regenerate-list” to update “todo-orders”.

9.1.7. Load to-do list

It sets the value of orderList to be the value of the “todo-orders” and convert it into an array for each item in the array it creates the new list element and retrieve the value of the key using local storage.getItem().

9.2. TODO application in CS

The same application as JavaScript is created in the CoffeeScript. It has the same features and has the same HTML and CSS as in appendix 1 and 2. The idea of development is also the same. The data will be stored in the local storage.

As described earlier the CoffeeScript address the many problems of JavaScript. It omitted the curly braces and parenthesis and introduces the indentation that makes the code more readable, structured and short in length. In terms of language features it introduces the classes and inheritance, shortcuts for most common JavaScript code, for example it uses ‘@’ in place of ‘this’, ‘::’ for prototypes and there is no ternary operator, etc.

In appendix 3, there is a code for the application in CoffeeScript. It is compiled and the generated javaScript is used to run the application in a web browser. It can be seen from the code that it is a completely different language than JavaScript. It is not the superset or a subset of JavaScript that can be easily mixed with JavaScript. But it is usually compiled into JavaScript or can be executed outright.

From our brief experience of writing the same application in CoffeeScript, it has been observed that programming in CoffeeScript requires the more advanced knowledge of JavaScript. Because it produces the JavaScript after compilation so most of the time you have to think about the produced JavaScript code from your CoffeeScript code that it should be correct enough to work and run your application properly.

The code is more like Pythan and Ruby. But if we compare the lines of code with javaScript they are almost 2/3 number of lines of JavaScript version.

At the beginning of the file there is a declaration of variables. How the flow of the application works is already described in the JavaScript section. Here only the syntax and the language features are described that are different than JavaScript.
In the add function the structure object todo Object is defined in the same way as in JavaScript and the storing procedure of the structure object in local storage is also the same but syntax is little different. In load function the ternary operator is replaced with the 'if then else' statement to achieve the same functionality. There is no return statement because CoffeeScript consider the last statement or expression of any function as a return statement/expression. Instead of "this" the "@" is used.

There is no forEach(function(){…}) in CoffeeScript as a result it has been observed that the core logic of the application is better.

9.3. TODO application in Dart

It is a separate language like CoffeeScript but the syntax is similar to Java or c#. It has classes, objects, maps, lists etc. There is a main() function that is starting point of any application. The code can be compiled to JavaScript or can be run directly to Dart VM.
In the todo application the HTML and CSS file is the same as used in JavaScript and CoffeeScript application but unlike in CoffeeScript the code is not compiled to JavaScript. Therefore, there is a change in the script tag of HTML file in appendix1.

```
<script type="application/dart" src="todo.dart"></script>
```

It is already mentioned that the dart team claimed the high performance of the dart applications because the compiler produced more optimal JavaScript code than what you would write yourself. While having a look at the appendix 5 it is clearly seen that the brevity of code is not the goal of Dart like CoffeeScript, instead the language tries to achieve the better maintainability and the speed.

To develop the application the dart editor is used and it has the dartium browser also. Where the application can run and test while developing. The Dart editor has the ability to highlight the different lexiums of code and ability of code assistance. Right now there is no plugin for the other popular IDEs for Dart. In the future the plugins might appear.

The development is completely different than JavaScript. In this application the features are almost the same as described in JavaScript section. For the persistent data the LocalStrage is used here also but the language has its own functions and its own way to use the Local Storage.

First of all, let's see how the dart is communicating with the HTML. In appendix 5, the every first line is the import statement. Dart has the import directives which imports the specified library, making all the functions in that library available. Here, import 'dart:html'; imports Dart's HTML classes and functions for programming the DOM. All Dart web apps need the Dart HTML library. The other import directive provides the drag and drop functions and classes, this is basically for sorting the todo list elements.

The next section of the code is to declare the variables. Then the main() function that is the starting point of the application. To get the value of the input field and the list elements, it is using the query() function. The query() is a top-level function provided by the Dart HTML library that gets an Element object from the DOM.

9.3.1. Load the ToDo Items

The main() function then calling the loadToDoItems() function which loads the todo items from the local storage. First it checks if there is any to-do item stored in the local storage. If the condition is true then it iterates the localstorage list and get all the todo items one by one. To create the list item in HTML it used the LIElement(). After creating the LIElement and assigning it the value from local storage it then appends that to the toDolist.

9.3.2. Add the new ToDo item

It is similar to the load function and called from the main(). It first checks the input field should not empty. If the condition is true, it creates the new LIElement and assigns it the value from the input field. The same value is stored in the local storage with the same key format that is used in JS and CS applications. Then, it appends the list element to the toDolist to make it visible on the HTML page. At the end, it clears the input field and increment the key value.

9.3.3. Remove the todo-item

There is not specific function written to remove the todo-item rather the functionality is provided when the user clicks the todo-item then it will be removed from the list. To do this an onclick event is called and it removes the item from the local storage and as well as from the DOM.

9.3.4. Remove all todo-items

There is a button to delete all the todo items. When it is clicked the click event fires and in the listener it calls the removeAll() function. Where it clears the todoList at DOM and empties the local storage.

9.3.5. Sort the todo-list

In JS and CS this sortable list functionality is achieved with the JQuery. In Dart, it provides its own library for drag and drop and sortable list.

9.4. TODO application in TypeScript

TypeScript is a superset of JavaScript. It has type annotations, classes and modules. TypeScript does not replace the JavaScript like Dart but any valid JavaScript code is valid TypeScript code, it can compile without any errors and warnings. In the todo application the HTML and CSS file is the same as used in JavaScript, CoffeeScript and Dart applications, like in CoffeeScript the code is also compiled to JavaScript. However, unlike CoffeeScript, TypeScript has a good tool support in the form of a Visual Studio editor and potentially plugins for other IDEs.

We created a new file with .ts extension and copy the code of the JavaScript todo application and paste it in the TypeScript file and then compiled the code it compiled successfully without any errors and warnings and generated the .js file. The generated .js file is then used in the application it gives the same results.

Converting the JavaScript code into more idiomatic TypeScript code is very fast and simple. Developing in TypeScript feels a lot like developing in JavaScript but it is a bit safer because of type annotations and the compiler.

The main idea to develop the todo application in TypeScript is the same as in JS. The data is stored in the local storage and it has the same features that are described in section 9. This version of TODO application is developed by using Backbone.js and JQuery. JQuery is used for all DOM manipulation and TypeScript classes are used to create Backbone models and views.

10. RESULTS AND CONCLUSION

JavaScript is the scripting language that is most widely used in the development of web applications. It is fast and responsive scripting language. The applications can be usable and responsive, even in the absence of the internet connection. JavaScript provides the seamless integration with user plug-ins. It is also an easy language to learn, there is a supporting online community of JavaScript developers and information resources. But we cannot forget the other side of the picture. There are many pitfalls and ugly sides of JavaScript such as development and maintainability of large scale complex applications etc. There is a huge need for JavaScript improvements that is why a number of scripting languages came into existence.

All the considered languages (CoffeeScript, TypeScript and Dart) can be compiled into JavaScript although it is a high level programming language but here it plays a role of a low level language. These scripting languages improve the maintainability of large application by providing the features of object oriented programming with the help of classes and modules. Many of them for example Dart and TypeScript introduced the optional static types. These languages improve the performance and achieve the code brevity.

Dart and TypeScript have the IDE but there is no better IDE support for CoffeeScript. These scripting languages are the generic and can be used by both client and server side. They support all the libraries of JavaScript and preserve the functions as first-class citizen which seems to be a very powerful feature in JavaScript.

Which scripting language is best and should be used to develop the new web application? The answer to this question varies. For the enterprise applications, website development and If you are using the Microsoft tools for development, TypeScript would be a good choice, it uses the JQuery and Ajax for dynamic updation. A major part of benefit of TypeScript comes from the tooling within Visual Studio.

One can consider development of large scale applications in Dart because it is a clean modern language with extensive libraries out of the box. It has a cleaner DOM API, full fledge client side web applications can be developed. It has its own editor and browser to run the applications but what if it does not run properly or fast enough on some devices? This is the risk that comes with the Dart.

CoffeeScript provide structure to the code and reduce unnecessary code. It is close to the regular JavaScript, it has no interoperability or performance issues. The CoffeeScript is most widely adapted alternative language to JavaScript. Coffee Script can be used to develop medium to large scale applications. System and server applications can also possible to develop that run on top of node.js. It is a programming language does not include the DOM related convenience functions; however you can use CoffeeScript with your favorite toolkit JQuery and Ajax for dynamic updation. It has ability to call the web services. There are relatively more resources available and more users in the community to answer questions.

Dart and TypeScript are still moving fast and changing and have only their first versions available for use. All these three languages have the better structure of writing the application code. The debugging is easy in Dart and TypeScript.

Google Trend [51] provides some idea of the relative interest in each language. In the figure 9.1 the web search interest is shown over past two years worldwide. The number 100 represents the peak search interest. According to the graph the CoffeeScript has the maximum search interest in all.

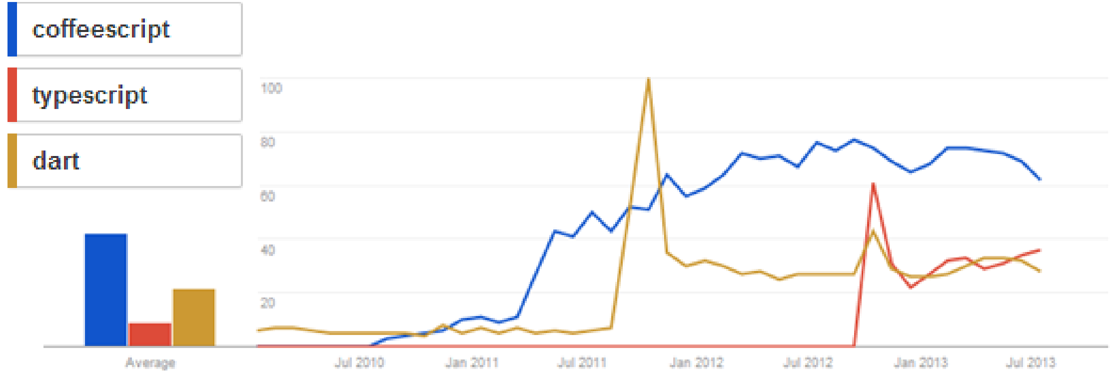

Figure9.1.*user's interest over time [51]*

To conclude, here is the table that compares the language features in JavaScript, CoffeeScript, TypeScript and Dart.

Features	**JavaScript**	**CoffeeScript**	**TypeScript**	**Dart**
Static Type Checking	-	-	X	X
Classes	-	X	X	X
Interfaces	-	-	X	X
Modules	-	X	X	X
List comprehensions	-	X	-	-
String interpolations	-	X	-	X
Splats/Rest	-	X	X	-

parameters(...)				
Intellisense	-	-	X	X
Code Brevity	-	X	-	-
Stable	-	X	-	-
Open Source	-	-	X	X
Compile to JS	-	X	X	X
Better speed	-	-	-	X

REFERENCES

[1] IBM System/360 Operating System Job Control Language (C28-6529-4) . [Online] [Cited: Febuary 13, 2013.] http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/360/os/R01-08/C28-6539-4_OS_JCL_Mar67.pdf

[2] Robert W. Sebesta, Concepts of programming languages , 10th edition, ISBN-10: 0131395319, p.773

[3] Scripting language. [Online] [Cited: Febuary 13, 2013.] http://en.wikipedia.org/wiki/Scripting_language

[4] Future of JavaScript. [Online] [April 05, 2013.] https://gist.github.com/paulmillr/1208618

[5] Roy. [Cited: November 28, 2012.] http://roy.brianmckenna.org/

[6] Clojure. [Online] [Cited: Febuary 16, 2013.] http://clojure.org/

[7] Opal. [Online] [Cited: Febuary 16, 2013.] http://opalrb.org/

[8] LiveScript. [Online] [Cited: Febuary 16, 2013.] www.livescript.net

[9] ParenScript. [Online] [Cited: Febuary 16, 2013.] http://common-lisp.net/project/parenscript/

[10] Fay. [Online] [Cited: Febuary 16, 2013.] https://github.com/faylang/fay/wiki

[11] Ceylon. [Online] [Cited: Febuary 16, 2013.] http://ceylon-lang.org/

[12] SSL/Computer Weekly IT salary survey: finance boom drives IT job growth [Online] [Cited: June 14, 2013.] http://computerweekly.com

[13] Counting programming languages [Online] [Cited: June 14, 2013.] http://radar.oreilly.com

[14] Bieman, J.M.; Murdock, V., Finding code on the World Wide Web: a preliminary investigation, Proceedings First IEEE International Workshop on Source Code Analysis and Manipulation, 10 Novmenber 2001, Florance, Itlay, ISBN 0-7695-1387-5, p.225

[15] TIOBE programming community index [Online] [Cited: June 14, 2013.] http://www.tiobe.com/index.php/tiobe index,2013.

[16] Klaus Purer, The web scripting language shootout . Vienna University of Technology [Online] [Cited: May 10, 2013.] http://klau.si/sites/default/files/php-vs-python-vs-ruby.pdf

[17] JavaScript. [Online] [Cited: May 11, 2013.] http://www.w3schools.com/js/js_intro.asp

[18] Press release announcing JavaScript, "Netscape and Sun announce Javascript", PR Newswire, December 4, 1995 [Online] [Cited: May 11, 2013.] http://web.archive.org/web/20070916144913/http:/wp.netscape.com/newsref/pr/newsrelease67.html

[19] Innovators of the Net: Brendan Eich and JavaScript. [Online] [Cited: May 11, 2013.] http://web.archive.org/web/20080208124612/http://wp.netscape.com/comprod/columns/techvision/innovators_be.html

[20] Chapter 2: Getting Started. Server-Side JavaScript Guide. Netscape Communications Corporation. 1998. [Online] [Cited: May 11, 2013.] http://docs.oracle.com/cd/E19957-01/816-6411-10/getstart.htm

[21] Mike Morgan, Using Netscape™ LiveWire™ Chapter 6: Netscape Internet Application Framework, Special Edition. Que. 1996, ISBN: 0-7897-0743-8, p.329

[22] Server-Side Javascript: Back With a Vengeance. Read Write Web. December 17, 2009. [Online] [Cited: May 28, 2013.] http://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance#awesm=~od54eeDY6HDFtR

[23] Node's goal. [Online] [Cited: May 19, 2013.] http://nodejs.org/about/

[24] ECMAScript 3rd Edition specification. [Online] [Cited: May 26,2013.] http://nodejs.org/about/

[25] USPTO. [Online] [Cited: May 28, 2013.] http://tsdr.uspto.gov/#caseNumber=75026640&caseType=SERIAL_NO&searchType=statusSearch

[26] Sun Trademarks. [Online] [Cited: May28, 2013.] http://www.oracle.com/us/sun/index.htm

[27] Schach, Stephen ,Object-oriented and Classical Software Engineering, Seventh Edition. McGraw-Hill. ISBN 0-07-319126-4.

[28] E Balagurusamy, Object Oriented Programming with C++, 2008, Tata McGraw-Hill Education, ISBN 0-07-066907-4, p. 529

[29] John C. Mitchell, Concepts in programming languages, Cambridge University Press, 2003, ISBN 0-521-78098-5, p.522

[30] Michael Lee Scott, Encapsulation. Programming language pragmatics, Edition 2, 2006, ISBN 0-12-633951-1, p. 481

[31] Ivo Wetzel and Zhang Yi Jiang, Equality and Comparisons. JavaScript Garden [Online] [Cited: April 02,103] http://bonsaiden.github.io/JavaScript-Garden/

[32] CoffeeScript. [Online] [Cited: April 02, 2013.] www.coffeescript.org/

[33] Initial commit of the mystery language. [Online] [April 02, 2013.] https://github.com/jashkenas/coffee-script/commit/8e9d637985d2dc9b44922076ad54ffef7fa8e9c2

[34] Jeremy Ashkenas, CoffeeScript Design Decisions,2011 [Online] [April 04, 2013.] http://isolani.co.uk/blog/javascript/FullFrontal2011CoffeeScriptDesignDecisions

[35] Jeremy Ashkenas , CoffeeScript 1.0.0 announcement posted on Dec 24, 2010 at Hacker News.

[36] Jeremy Ashkenas, Original CoffeeScript announcement posted on Dec 24, 2009 at Hacker News.

[37] Alex MacCaw, The Little Book on CoffeeScript, O'Reilly Media, 2012, p62 [Online] [April 05, 2013.] http://arcturo.github.io/library/coffeescript/index.html

[38] Future of JavaScript. [Online] [April 05, 2013.] https://gist.github.com/paulmillr/1208618

[39] Dartium. [Online] [Cited: April 18, 2013.] http://www.dartlang.org/tools/dartium/

[40] Kathy Walrath & Seth Ladd, Dart: Up and Running, O'REILLY Media , 2012, ISBN 978-1-4493-3084-2, p152

[41] Mixins. [Online] [Cited: April 22, 2013.]

http://www.dartlang.org/articles/mixins/#mixins

[42] Dart. [Online] [Cited: April 22, 2013.] http://www.dartlang.org/docs/dart-up-and-running/contents/ch01.html

[43] Dart. [Online] [Cited: April 22, 2013.] http://www.infoq.com/articles/google-dart

[44] TypeScript. [Online] [Cited: June 21,2013.] http://www.zdnet.com/who-built-microsoft-typescript-and-why-7000005206/

[45] TypeScript Compile [Online] [Cited: June 21,2013.] https://github.com/niutech/typescript-compile

[46] Microsoft, TypeScript language specification, version 09, May2013

[47] TypeScript. [Online] [Cited: June 20, 2013.] http://en.wikipedia.org/wiki/TypeScript

[48] TypeScript: JavaScript Development at Application Scale. [Online] [Cited: June 20, 2013.] http://blogs.msdn.com/b/somasegar/archive/2012/10/01/typescript-javascript-development-at-application-scale.aspx

[49] TypeScript: Making .NET Developers Comfortable with JavaScript . [Online] [Cited: June 21, 2013.] http://msdn.microsoft.com/en-us/magazine/jj883955.aspx

[50] Performance. [Online] [Cited: June 26, 2013.] http://www.dartlang.org/performance/

[51] Google Tend. [Online] [Cited: July 05, 2013.] http://www.google.com/trends/explore#cat=0-5-31&q=coffeescript%2Ctypescript%2Cdart&date=1%2F2010%2048m&cmpt=q

[52] Sven Apel, Thomas Leich, Marko Rosenmuller, and Gunter Saake, Combining Feature-Oriented and Aspect-Oriented Programming to Support to Software Evaluation. University of Magdeburg, Germany, 25 July 2005

[53] Brian McKenna, A Statically Typed, Functional Language for JavaScript, IEEE Internet Computing, vol. 16, no. 3, pp. 86-91, May-June 2012

[54] Kaffeine. [Online] [Cited: August 08, 2013.] http://weepy.github.io/kaffeine/

[55] Hexe. [Online] [Cited: August 12,2013.] http://haxe.org/

APPENDIX 1: INDEX.HTML

```
<html>
    <head>
       <title>To-do List</title>
       <meta charset="utf-8" />
       <link rel="stylesheet" href="css/base.css" type="text/css" />

      <script type="text/javascript" src="js/jquery-1.9.1.min.js"></script>
      <script type="text/javascript" src="js/jquery-ui-1.10.2.min.js"></script>
       <script src="js/jquery.inlineedit.js"></script>
       <script src="js/pubsub.js"></script>
      <!-- script file, change this file while running app in different lanagauges
->
       <script type="text/javascript" src="js/js_script.js"></script>

    </head>
    <body>
          <div id="container">
            <h1>To-Do List</h1>
            <form id="todo-form">
                <input id="todo" type="text" />
                <input id="priority" type="text" />
                <input id="submit" type="submit" value="Add to List">
            </form>
            <ul id="show-items"></ul>
            <a href="#" id="clear-all">Clear All</a>
        </div>

    </body>
</html>
```

APPENDIX 2: BASE.CSS

```
body {
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    font-weight: 300;
    font-size: 12px;
}

a,
a:link {
    outline: none;
}

h1 {
    font-weight: 100;
    font-size: 72px;
    margin: 20px 0;
}

#container {
    width: 800px;
    text-align: center;
    margin: 20px auto;
}

input[type="text"] {
    height: 30px;
    width: 350px;
    padding: 10px;
    margin-right: 10px;
    font-size: 24px;
}

input[type="submit"] {
    height: 56px;
    padding: 10px;
    border: 1px solid #333;
    background-color: #ccc;
    font-size: 24px;
    cursor: pointer;
    outline: none;
}

ul {
    margin: 15px auto 0;
    padding: 0;
    width: 545px;
}

ul li {
    list-style-type: none;
    font-size: 24px;
    cursor: move;
    background-color: #efefef;
    margin-bottom: 5px;
    padding: 10px;
    text-align: left;
```

```
}

ul li span {
    cursor: text;
}

ul li a,
ul li a:link {
    float: right;
    display: none;
    text-decoration: none;
    color: #f03;
}

ul li a:hover {
    text-decoration: underline;
}
```

APPENDIX 3: JS_SCRIPT.JS

```
$("document").ready(function() {
    var i = Number(localStorage.getItem('todo-counter')) + 1,
        j = 0,
        k,
        $form = $('#todo-form'),
        $removeLink = $('#show-items li a'),
        $itemList = $('#show-items'),
        $editable = $('.editable'),
        $clearAll = $('#clear-all'),
        $newTodo = $('#todo'),
             $newPriority = $('#priority'),
        order = [],
        orderList;

    // Load todo list
     orderList = localStorage.getItem('todo-orders');

     orderList = orderList ? orderList.split(',') : [];

    for( j = 0, k = orderList.length; j < k; j++) {
                    var retrievedObject = localStorage.getItem(orderList[j]);
                    var testObject_r = JSON.parse(retrievedObject);

       $itemList.append(
             "<li id='" + orderList[j] + "'>"
            + "<span class='editable'>"
            + testObject_r.one
            + "</span> <a href='#'>X</a></li>"
         );
    }


           // Add todo
    $form.submit(function(e) {
             alert("hi");
        e.preventDefault();
        $.publish('/add/', []);
    });
         // Remove todo
    $itemList.delegate('a', 'click', function(e) {
        var $this = $(this);

        e.preventDefault();
        $.publish('/remove/', [$this]);
    });

    // Sort todo
    $itemList.sortable({
        revert: true,
        stop: function() {
            $.publish('/regenerate-list/', []);
        }
    });
```

```
// Edit and save todo
    $editable.inlineEdit({
        save: function(e, data) {
                var $this = $(this);
                localStorage.setItem(
                    $this.parent().attr("id"), data.value
                );
            }
    });

    // Clear all
    $clearAll.click(function(e) {
        e.preventDefault();
        $.publish('/clear-all/', []);
    });

    // Fade In and Fade Out the Remove link on hover
    $itemList.delegate('li', 'mouseover mouseout', function(event) {
        var $this = $(this).find('a');

        if(event.type === 'mouseover') {
            $this.stop(true, true).fadeIn();
        } else {
            $this.stop(true, true).fadeOut();
        }
    });
      // Subscribes
    $.subscribe('/add/', function() {
      var testObject = { 'one': $newTodo.val(), 'two': $newPriority.val() };
     if ($newTodo.val() !== "") {
            // Take the value of the input field and save it to localStorage
            localStorage.setItem(
                "todo-" + i, JSON.stringify(testObject)
            );

               // Set the to-do max counter so on page refresh it keeps going up
instead of reset
            localStorage.setItem('todo-counter', i);
               // Retrieve the object from storage
                   var retrievedObject = localStorage.getItem("todo-" + i);
                   var testObject_r = JSON.parse(retrievedObject);

                   // Append a new list item with the value of the new todo list
            $itemList.append(
                "<li id='todo-" + i + "'>"
                + "<span class='editable'>"
                + testObject_r.one
                + " </span><a href='#'>x</a></li>"
            );
                      $.publish('/regenerate-list/', []);

            // Hide the new list, then fade it in for effects
            $("#todo-" + i)
                .css('display', 'none')
                .fadeIn();
```

```
                // Empty the input field
                $newTodo.val("");

                i++;
                 }
        });

           $.subscribe('/remove/', function($this) {
            var parentId = $this.parent().attr('id');

            // Remove todo list from localStorage based on the id of the clicked parent
element
            localStorage.removeItem(
                "'" + parentId + "'"
            );

            // Fade out the list item then remove from DOM
            $this.parent().fadeOut(function() {
                $this.parent().remove();

                $.publish('/regenerate-list/', []);
            });
        });
           $.subscribe('/regenerate-list/', function() {
            var $todoItemLi = $('#show-items li');
            // Empty the order array
            order.length = 0;

            // Go through the list item, grab the ID then push into the array
            $todoItemLi.each(function() {
                var id = $(this).attr('id');
                order.push(id);
            });

            // Convert the array into string and save to localStorage
            localStorage.setItem(
                'todo-orders', order.join(',')
            );
        });
           $.subscribe('/clear-all/', function() {
            var $todoListLi = $('#show-items li');

            order.length = 0;
            localStorage.clear();
            $todoListLi.remove();
        });

    });
```

APPENDIX 4: CS_SCRIPT.COFFEE

```
$("document").ready ->
  i = Number(localStorage.getItem("todo-counter")) + 1
  j = 0
  k = undefined
  $form = $("#todo-form")
  $removeLink = $("#show-items li a")
  $itemList = $("#show-items")
  $editable = $(".editable")
  $clearAll = $("#clear-all")
  $newTodo = $("#todo")
  $newPriority = $("#priority")
  order = []
  orderList = undefined

  # Load todo list
  orderList = localStorage.getItem("todo-orders")
  orderList = (if orderList then orderList.split(",") else [])
  j = 0
  k = orderList.length

  while j < k
    retrievedObject = localStorage.getItem(orderList[j])
    testObject_r = JSON.parse(retrievedObject)
    $itemList.append "<li id='" + orderList[j] + "'>" + "<span class='editable'>" +
testObject_r.one + "</span> <a href='#'>X</a></li>"
    j++

  # Add todo
  $form.submit (e) ->
    alert "hi"
    e.preventDefault()
    $.publish "/add/", []


  # Remove todo
  $itemList.delegate "a", "click", (e) ->
    $this = $(this)
    e.preventDefault()
    $.publish "/remove/", [$this]


  # Sort todo
  $itemList.sortable
    revert: true
    stop: ->
      $.publish "/regenerate-list/", []


  # Edit and save todo
  $editable.inlineEdit save: (e, data) ->
    $this = $(this)
    localStorage.setItem $this.parent().attr("id"), data.value
```

```
# Clear all
  $clearAll.click (e) ->
    e.preventDefault()
    $.publish "/clear-all/", []


  # Fade In and Fade Out the Remove link on hover
  $itemList.delegate "li", "mouseover mouseout", (event) ->
    $this = $(this).find("a")
    if event.type is "mouseover"
      $this.stop(true, true).fadeIn()
    else
      $this.stop(true, true).fadeOut()


  # Subscribes
  $.subscribe "/add/", ->
    testObject =
      one: $newTodo.val()
      two: $newPriority.val()

    if $newTodo.val() isnt ""

      # Take the value of the input field and save it to localStorage
      localStorage.setItem "todo-" + i, JSON.stringify(testObject)

      # Set the to-do max counter so on page refresh it keeps going up instead of
reset
      localStorage.setItem "todo-counter", i

      # Retrieve the object from storage
      retrievedObject = localStorage.getItem("todo-" + i)
      testObject_r = JSON.parse(retrievedObject)

      # Append a new list item with the value of the new todo list
      $itemList.append "<li id='todo-" + i + "'>" + "<span class='editable'>" +
testObject_r.one + " </span><a href='#'>x</a></li>"
      $.publish "/regenerate-list/", []

      # Hide the new list, then fade it in for effects
      $("#todo-" + i).css("display", "none").fadeIn()

      # Empty the input field
      $newTodo.val ""
      i++

  $.subscribe "/remove/", ($this) ->
    parentId = $this.parent().attr("id")

    # Remove todo list from localStorage based on the id of the clicked parent
element
    localStorage.removeItem "'" + parentId + "'"

    # Fade out the list item then remove from DOM
    $this.parent().fadeOut ->
```

```
$this.parent().remove()
      $.publish "/regenerate-list/", []

  $.subscribe "/regenerate-list/", ->
    $todoItemLi = $("#show-items li")

    # Empty the order array
    order.length = 0

    # Go through the list item, grab the ID then push into the array
    $todoItemLi.each ->
      id = $(this).attr("id")
      order.push id

    # Convert the array into string and save to localStorage
    localStorage.setItem "todo-orders", order.join(",")

  $.subscribe "/clear-all/", ->
    $todoListLi = $("#show-items li")
    order.length = 0
    localStorage.clear()
    $todoListLi.remove()
```

APPENDIX 5: DART_SCRIPT.DART

```
import 'dart:html';
import 'package:html5_dnd/html5_dnd.dart';

InputElement toDoInput;
UListElement toDoList;
ButtonElement deleteAll;
int i=0;
var list;

void main() {
  toDoInput = query('#todo');
  toDoList = query('#show-items');

  loadTodoItems();

  toDoInput.onChange.listen(addToDoItem);
  deleteAll = query('#clear-all');
  deleteAll.onClick.listen((e){
    toDoList.children.clear();
    removeAll();
  });
}
//load the todo lits
void loadTodoItems()
{

  if(!window.localStorage.isEmpty){

    i = window.localStorage.length-1;
    for(int j=0; j < window.localStorage.length; j++ )
    {
      if(window.localStorage["todo-$j"] != "")
      {
        var newToDo = new LIElement();
        newToDo.text = window.localStorage["todo-$j"];
        toDoList.children.insert(j,newToDo);
        //sortable list
        SortableGroup sortGroup=new SortableGroup();
        sortGroup.installAll(queryAll('#sortable-list $newToDo'));
        sortGroup.onSortUpdate.listen((SortableEvent event){
          print('elements are sorted');
        });
        newToDo.onClick.listen((e) {
          //when click on item, remove it
          newToDo.remove();
          window.localStorage.remove("todo-$j");

        });
      }

    }
  }
}
```

```
//add the todo item
void addToDoItem(Event e) {

  if(toDoInput.value != "")
  {
    var newToDo = new LIElement();
    newToDo.text = toDoInput.value;
    window.localStorage["todo-$i"]= toDoInput.value;
    //sortable list
    SortableGroup sortGroup=new SortableGroup();
    sortGroup.installAll(queryAll('#sortable-list $newToDo'));
    sortGroup.onSortUpdate.listen((SortableEvent event){
      print('elements are sorted');
    });
    newToDo.onClick.listen((e){
      //when click on item, remove it
      newToDo.remove();
      window.localStorage.remove("todo-$i");
    });
    toDoInput.value = '';
    //toDoList.children.add(newToDo);
    toDoList.children.insert(i,newToDo);
    i++;
  }
}
//delete all the todo items
void removeAll(){
  window.localStorage.clear();
}
```

APPENDIX 6: TS_SCRIPT.TS

```
declare module Backbone {
    export class Model {
        constructor (attr? , opts? );
        get(name: string): any;
        set(name: string, val: any): void;
        set(obj: any): void;
        save(attr? , opts? ): void;
        destroy(): void;
        bind(ev: string, f: Function, ctx?: any): void;
        toJSON(): any;
    }
    export class Collection<T> {
        constructor (models? , opts? );
        bind(ev: string, f: Function, ctx?: any): void;
        length: number;
        create(attrs, opts? ): any;
        each(f: (elem: T) => void ): void;
        fetch(opts?: any): void;
        last(): T;
        last(n: number): T[];
        filter(f: (elem: T) => boolean): T[];
        without(...values: T[]): T[];
    }
    export class View {
        constructor (options? );
        $(selector: string): JQuery;
        el: HTMLElement;
        $el: JQuery;
        model: Model;
        remove(): void;
        delegateEvents: any;
        make(tagName: string, attrs? , opts? ): View;
        setElement(element: HTMLElement, delegate?: boolean): void;
        setElement(element: JQuery, delegate?: boolean): void;
        tagName: string;
        events: any;

        static extend: any;
    }
}
interface JQuery {
    fadeIn(): JQuery;
    fadeOut(): JQuery;
    focus(): JQuery;
    html(): string;
    html(val: string): JQuery;
    show(): JQuery;
    addClass(className: string): JQuery;
    removeClass(className: string): JQuery;
    append(el: HTMLElement): JQuery;
    val(): string;
    val(value: string): JQuery;
    attr(attrName: string): string;
}
declare var $: {
```

```
    (el: HTMLElement): JQuery;
    (selector: string): JQuery;
    (readyCallback: () => void ): JQuery;
};
declare var _: {
    each<T, U>(arr: T[], f: (elem: T) => U): U[];
    delay(f: Function, wait: number, ...arguments: any[]): number;
    template(template: string): (model: any) => string;
    bindAll(object: any, ...methodNames: string[]): void;
};
declare var Store: any;
// Todo Model
// ----------

// Our basic **Todo** model has `content` and `order` attributes.
class Todo extends Backbone.Model {

    // Default attributes for the todo.
    defaults() {
        return {
            content: "empty todo..."
        }
    }

        // Ensure that each todo created has `content`.
    initialize() {
        if (!this.get("content")) {
            this.set({ "content": this.defaults().content });
        }
    }

       // Remove this Todo from *localStorage* and delete its view.
    clear() {
        this.destroy();
    }
}
// Todo Collection
// ---------------
// The collection of todos is backed by localStorage
class TodoList extends Backbone.Collection<Todo> {

    // Reference to this collection's model.
    model = Todo;

    // Save all of the todo items under the `"todos"` namespace.
    localStorage = new Store("todos-backbone");

     // We keep the Todos in sequential order, despite being saved by unordered
    // GUID in the database. This generates the next order number for new items.
    nextOrder() {
        if (!this.length) return 1;
        return this.last().get('order') + 1;
    }

    // Todos are sorted by their original insertion order.
```

```
    comparator(todo: Todo) {
        return todo.get('order');
    }

}
// Create our global collection of **Todos**.
var Todos = new TodoList();
// Todo Item View
// --------------

// The DOM element for a todo item...
class TodoView extends Backbone.View {

    // The TodoView listens for changes to its model, re-rendering. Since there's
    // a one-to-one correspondence between a **Todo** and a **TodoView** in this
    // app, we set a direct reference on the model for convenience.
    template: (data: any) => string;

    // A TodoView model must be a Todo, redeclare with specific type
    model: Todo;
    input: JQuery;

    constructor (options? ) {
        //... is a list tag.
        this.tagName = "li";

        // The DOM events specific to an item.
        this.events = {
            "click .check": "toggleDone",
            "dblclick label.todo-content": "edit",
            "click span.todo-destroy": "clear",
            "keypress .todo-input": "updateOnEnter",
            "blur .todo-input": "close"
        };

        super(options);

        // Cache the template function for a single item.
        this.template = _.template($('#item-template').html());

        _.bindAll(this, 'render', 'close', 'remove');
        this.model.bind('change', this.render);
        this.model.bind('destroy', this.remove);
    }

    // Re-render the contents of the todo item.
    render() {
        this.$el.html(this.template(this.model.toJSON()));
        this.input = this.$('.todo-input');
        return this;
    }

    // Switch this view into `"editing"` mode, displaying the input field.
    edit() {
        this.$el.addClass("editing");
```

```
            this.input.focus();
        }

        // Close the `"editing"` mode, saving changes to the todo.
        close() {
            this.model.save({ content: this.input.val() });
            this.$el.removeClass("editing");
        }

        // If you hit `enter`, we're through editing the item.
        updateOnEnter(e) {
            if (e.keyCode == 13) close();
        }

        // Remove the item, destroy the model.
        clear() {
            this.model.clear();
        }
    }

    // The Application
    // ---------------
    // Our overall **AppView** is the top-level piece of UI.
    class AppView extends Backbone.View {

        // Delegated events for creating new items, and clearing completed ones.
        events = {
            "keypress #new-todo": "createOnEnter",
            "keyup #new-todo": "showTooltip",
            "click .todo-clear a": "clearCompleted",
        };

        input: JQuery;
        allCheckbox: HTMLInputElement;
        statsTemplate: (params: any) => string;

        constructor () {
            super();
            // Instead of generating a new element, bind to the existing skeleton of
            // the App already present in the HTML.
            this.setElement($("#todoapp"), true);

            // At initialization we bind to the relevant events on the `Todos`
            // collection, when items are added or changed. Kick things off by
            // loading any preexisting todos that might be saved in *localStorage*.
            _.bindAll(this, 'addOne', 'addAll', 'render');

            this.input = this.$("#new-todo");

            Todos.bind('add', this.addOne);
            Todos.bind('reset', this.addAll);

            Todos.fetch();
        }
        // Add a single todo item to the list by creating a view for it, and
```

```
    // appending its element to the `<ul>`.
    addOne(todo) {
        var view = new TodoView({ model: todo });
        this.$("#todo-list").append(view.render().el);
    }

    // Add all items in the **Todos** collection at once.
    addAll() {
        Todos.each(this.addOne);
    }

    // Generate the attributes for a new Todo item.
    newAttributes() {
        return {
            content: this.input.val(),
            order: Todos.nextOrder()
        };
    }

    // If you hit return in the main input field, create new **Todo** model,
    // persisting it to *localStorage*.
    createOnEnter(e) {
        if (e.keyCode != 13) return;
        Todos.create(this.newAttributes());
        this.input.val('');
    }

    tooltipTimeout: number = null;
    // Lazily show the tooltip that tells you to press `enter` to save
    // a new todo item, after one second.
    showTooltip(e) {
        var tooltip = $(".ui-tooltip-top");
        var val = this.input.val();
        tooltip.fadeOut();
        if (this.tooltipTimeout) clearTimeout(this.tooltipTimeout);
        if (val == '' || val == this.input.attr('placeholder')) return;
        this.tooltipTimeout = _.delay(() => tooltip.show().fadeIn(), 1000);
    }
}

// Load the application once the DOM is ready, using `jQuery.ready`:
$(() => {
    // Finally, we kick things off by creating the **App**.
    new AppView();
});
```