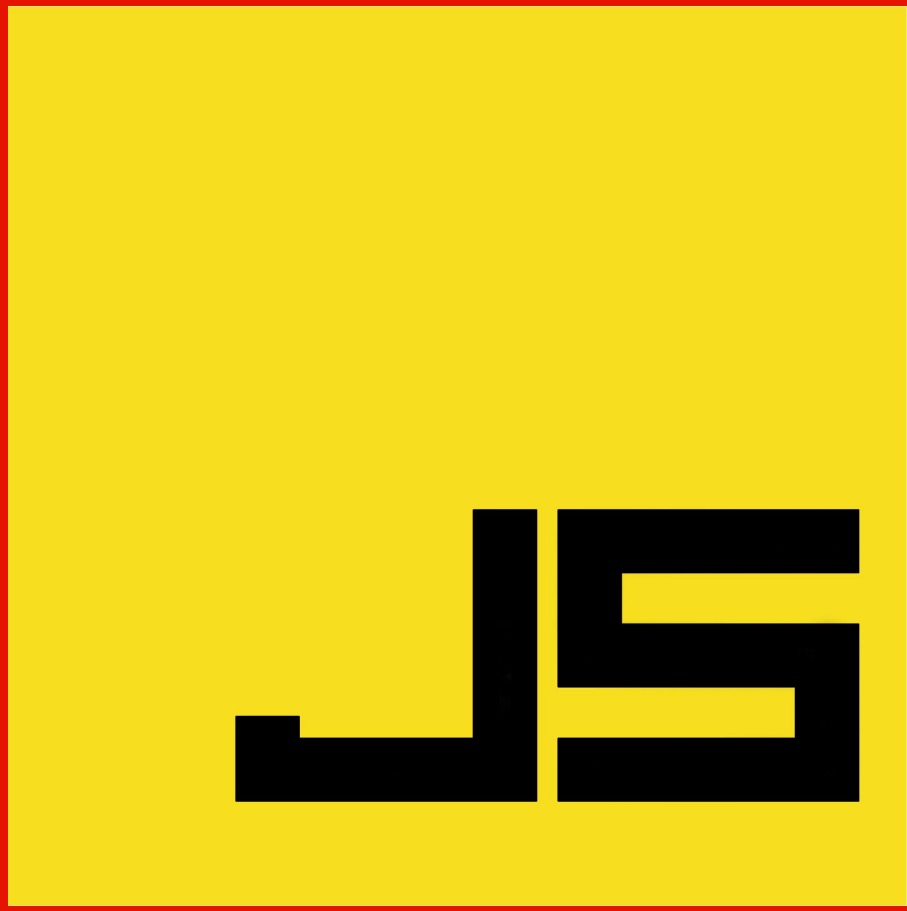
JavaScript Developer Foundation



Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

http://www.microsoft.com/about/legal/permissions/

DirectX, Internet Explorer, Microsoft, Microsoft Corporate Logo, MSDN, Outlook, Silverlight, SkyDrive, SQL Server, Visual Studio, Visual Studio design 2012, Windows, Windows Azure, Windows Live, Windows Phone, Windows Server, Windows Vista, Xbox 360 and Zune are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Agenda

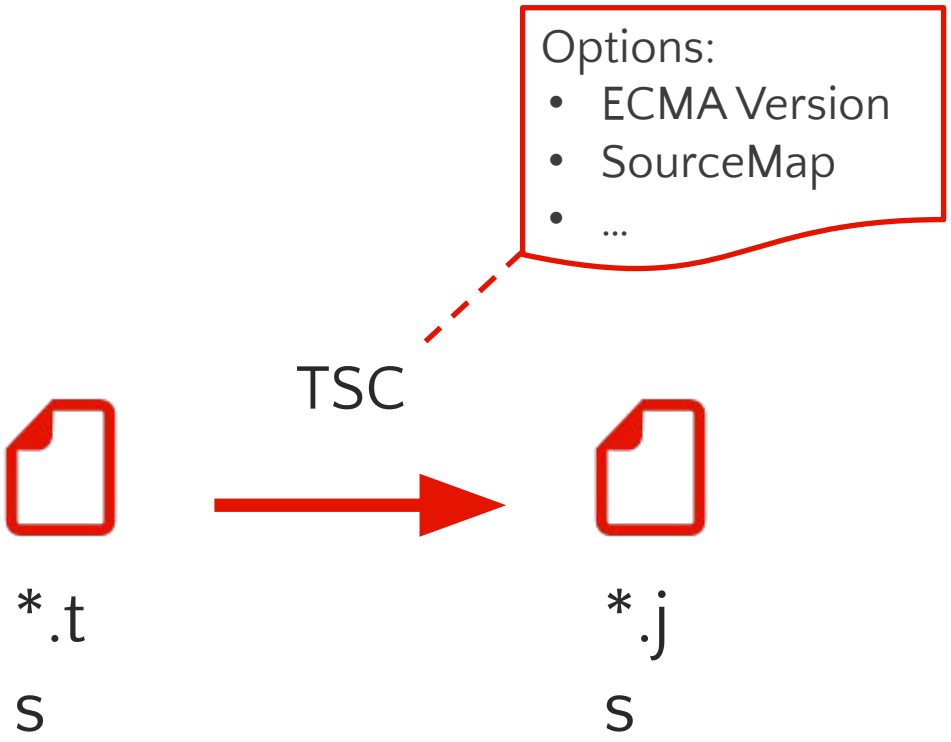
- TypeScript
- CoffeeScript
- asm.js

TypeScript

TypeScript

- Open Source
- Public Roadmap
  https://github.com/Microsoft/TypeScript/wiki/Roadmap
- Superset of JavaScript
- Transpiles to ES3, ES5, or ES6 □ no special runtime
- First preview in 2012, 1.0 in April 2014, 1.5 current

Transpilation



Tooling

- http://www.typescriptlang.org (Playground)
- Visual Studio 2012/2013 TypeScript 1.4 Extension
- Visual Studio 2015 TypeScript 1.5 Built-in
- NodeJS: Command Line | Build Automation
  `npm install -g typescript`

Overview

```
class Greeter {
    greeting: string;
    constructor (message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

var greeter = new Greeter("world");

var button = document.createElement("button");
button.innerText = "Say Hello";
button.onclick = function() {
    alert(greeter.greet());
}

document.body.appendChild(button);
```

JavaScript

```
function greeter(person) {
    return "Hello " + person;
}

var user = "Bill";

alert(greeter(user));
```

Type Annotation

```
function greeter(person: string) { // any | () => string | { a: number; b: string; }
    return "Hello " + person;
}

var user = "Bill";

alert(greeter(user));
```

Return types

```
function toupper(input: string): string { // void
    return input.toUpperCase();
}
```

Array types

```
var list1: Array<number> = [];
var list2: number[] = [];

list1.push(1);
list1.push("2"); // Error.

list2.push(1);
numArr2list2.push("2"); // Error.
```

Generics

```
function greeter<T>(person: T) {
    return "Hello " + person;
}

var user = "Bill";

alert(greeter<string>(user));
```

Interface

```
interface IPerson {
    firstName: string;
    lastName?: string; // Optional property.
}

function greeter(person: IPerson) {
    return "Hello " + person.firstName + " " + person.lastName;
}

var user = {firstName: "Bill", lastName: "Gates"};

alert(greeter(user));
```

Function type

```
// Interface.
interface SearchFunc {
    (source: string, subString: string): number;
}

var mySearch: SearchFunc = function (source: string, subString: string) {
    return source.search(subString);
}

// Annotation.
var myAdd: (a: number, b: number) => number = function (x, y) { return x + y; };
```

Array type

```
interface StringArray {
    [index: number]: string;
}

var myArr: StringArray = ["Hello", "world"];


interface Dictionary {
    [index: string]: string;
}

var myDict: Dictionary = { a: "Hello", b: "world" };
```

Class

```
class Student {
    fullName: string;
    constructor(public firstName, private middleName, public lastName = "") { // Optional param.
        this.fullName = firstName + " " + middleName + " " + lastName;
    }
}



interface IPerson {
    firstName: string;
    lastName: string;
}

function greeter(person: IPerson) {
    return "Hello " + person.firstName + " " + person.lastName;
}

var user = new Student("Bill", "H.", "Gates");

alert(greeter(user));
```

Implements

```
class Student implements IPerson {
    fullName: string;
    constructor(public firstName, private middleName, public lastName = "") { // Optional param.
        this.fullName = firstName + " " + middleName + " " + lastName;
    }
}



interface IPerson {
    firstName: string;
    lastName: string;
}

function greeter(person: IPerson) {
    return "Hello " + person.firstName + " " + person.lastName;
}

var user = new Student("Bill", "H.", "Gates");

alert(greeter(user));
```

Method

```
class Student {
    fullName: string;
    constructor(public firstName, private middleName, public lastName) {
        this.fullName = firstName + " " + middleName + " " + lastName;
    }
    greet() {
        return "Hello " + this.firstName + " " + this.lastName;
    }
}



var user = new Student("Bill", "H.", "Gates");

alert(user.greet());
```

Methods in interfaces

```
class Student implements IPerson {
    fullName: string;
    constructor(public firstName: string, private middleName: string, public lastName: string) {
        this.fullName = firstName + " " + middleName + " " + lastName;
    }
    greet(): string {
        return "Hello " + this.firstName + " " + this.lastName;
    }
}

interface IPerson {
    firstName: string;
    lastName: string;
    greet(): string;
}
```

Getters/Setters

```
class Employee {
    private _fullName: string;

    get fullName(): string {
        return this._fullName;
    }

    set fullName(value: string) {
        this._fullName = value;
    }
}

var employee = new Employee();
employee.fullName = "Bob Smith";

alert(employee.fullName);
```

Module

```
module Sayings {
    export class Student {
        fullName: string;
        constructor (public firstName, private middleName, public lastName) {
            this.fullName = firstName + " " + middleName + " " + lastName;
        }
        greet() {
            return "Hello " + this.firstName + " " + this.lastName;
        }
    }
}


var user = new Sayings.Student("Bill", "H.", "Gates");

alert(user.greet());
```

Static

```
module Sayings {
    export class Student {
        fullName: string;
        constructor (public firstName, private middleName, public lastName) {
            this.fullName = firstName + " " + middleName + " " + lastName;
        }
        greet() {
            return "Hello " + this.firstName + " " + this.lastName;
        }
        static goodbye(name: string) {
            return "Goodbye " + name;
        }
    }
}
```

```
var user = new Sayings.Student("Bill", "H.", "Gates");

alert(Sayings.Student.goodbye(user.fullName));
```

External objects

```
module Sayings {
    export declare class JSEncrypt {
        constructor(options: any);
        encrypt(input: string): string;
        decrypt(input: string): string;
    };
}
```

Multiple files

```
//---------- Validation.ts
module Validation {
    export interface Validator {
    }
}


//---------- StringValidator.ts
/// <reference path="Validation.ts" />
module Validation {
    export class StringValidator implements Validator {
    }
}


//---------- Test.ts
/// <reference path="Validation.ts" />
/// <reference path="StringValidator.ts" />
var stringValidator: Validation.StringValidator = new Validation.StringValidator();
```

Inheritance

```
class animal {
    eats: bool;
    constructor (public name) { };
    move(meters) {
        alert(this.name + " moved " + meters + "m.");
    }
}

class dog extends animal {
    constructor(name) { super(name); };
    move() {
        alert("Barks ...");
        super.move(5);
    }
}
```

Multiple inheritance

```
interface Shape {
    color: string;
}

interface PenStroke {
    penWidth: number;
}

interface Square extends Shape, PenStroke {
    sideLength: number;
}

var square = <Square>{};
square.color= "blue";
square.sideLength = 10;
square.penWidth = 5.0;
```

Enums

```
enum Color {
    red,
    blue,
    green,
}

var myColor = Color.red;
alert(Color[myColor]); // red.
```

Function parameters

```
function buildName(firstName: string, ...restOfName: string[]) {
    return firstName + " " + restOfName.join(" ");
}

var employeeName = buildName("George", "Alexander", "Louis", "of Cambridge");
```

this

```
var person = {
    name: "Bill Gates",
    greeter: function () {
        return function () {
            alert(this.name); // Error.
        };
    }
}

var person = {
    name: "Bill Gates",
    greeter: function () {
        return function () {
            alert(this.name);
        }.bind(this); // Solution 1.
    }
}

var greet = person.greeter();
greet();
```

Lambda

```
var person = {
    name: "Bill Gates",
    greeter: function () {
        var _this = this;

        return function () {
            alert(_this.name); // Solution 2.
        };
    }
}

var person = {
    name: "Bill Gates",
    greeter: function () {
        return () => { // Solution 3.
            alert(this.name);
        };
    }
}

var greet = person.greeter();
greet();
```

Overloading

```
function add(a: string, b: string): string;
function add(a: number, b: number): number;
function add(a: any, b: any): any {
    return a + b;
}

alert(add(1,  2));
alert(add("Hello ", "world"));
alert(add("Hello ", 1234)); // Error.
```

Union Types

```
function f(x: number | number[]) {
  if (typeof x === "number") {
    return x + 10;
  }
  else {
    // Return sum of numbers.
  }
}
```

Type Aliases

```
type PrimitiveArray = Array<string|number|boolean>;
type MyNumber = number;
type Callback = () => void;
```

Type Definition files

```
module zoo {
   function open(): void;
}

declare module "zoo" {
   export = zoo;
}
```

CoffeeScript

Overview

```
# Assignment:
number = 42
opposite = true

# Conditions:
number = -42 if opposite

# Functions:
square = (x) -> x * x

# Arrays:
list = [1, 2, 3, 4, 5]

# Objects:
math =
  root: Math.sqrt
  square: square
  cube: (x) -> x * square x
```

asm.js

JavaScript

```
function DiagModuleJS(stdlib, foreign?, heap?) {
    // Variable Declarations.
    var sqrt = stdlib.Math.sqrt;

    // Function Declarations.
    function square(x) {
        x = +x;
        return +(x * x);
    }

    function diag(x, y) {
        x = +x;
        y = +y;
        return +sqrt(+square(x) + +square(y));
    }

    return { diag: diag };
}
```

use asm

```
function DiagModule(stdlib, foreign, heap) {
    "use asm";

    // Variable Declarations.
    var sqrt = stdlib.Math.sqrt;

    // Function Declarations.
    function square(x) {
        x = +x;
        return +(x * x);
    }

    function diag(x, y) {
        x = +x;
        y = +y;
        return +sqrt(+square(x) + +square(y));
    }

    return { diag: diag };
}
```

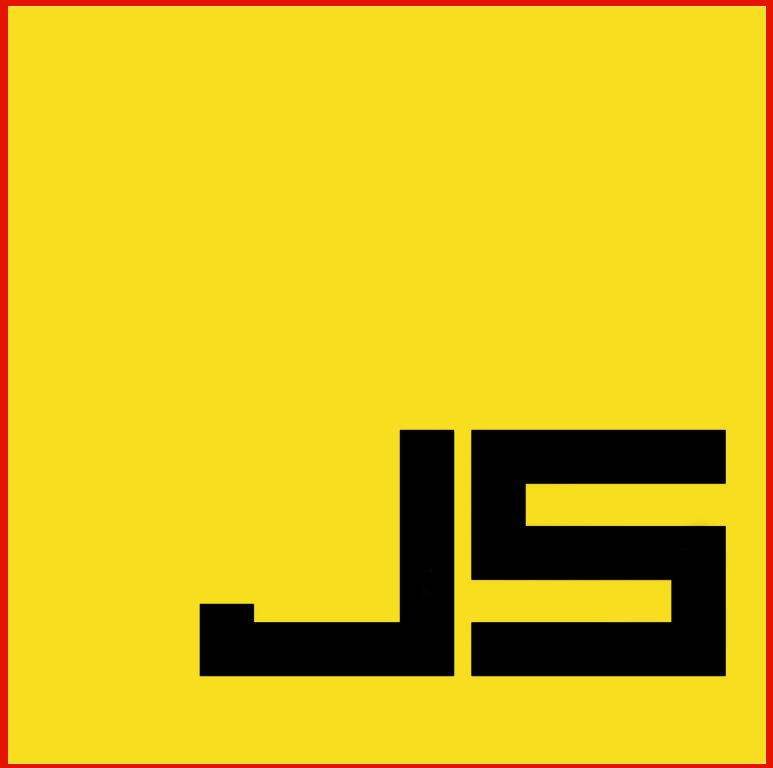
Speed comparison

```
var limit = 1000000;

var diagJs = DiagModuleJS({ Math: Math }).diag;
var start = +new Date();
for (var i = 0; i < limit; i++) {
    var result = diagJs(10, 100);
}
alert("JS: " + (+new Date() - start) + " ms");


var diag = DiagModule({ Math: Math }).diag;
var start = +new Date();
for (var i = 0; i < limit; i++) {
    var result = diag(10, 100);
}
alert("asm.js: " + (+new Date() - start) + " ms");
```

Characteristics

- Only for number types
- Data stored on heap
- No garbage collection or dynamic types
- Fallback to JS if not supported

- ~1.5x slower than compiled C++ (like C# or Java)
- ~4–10x faster than latest browser builds running JS



Microsoft