Electronics and
Computer Science

UNIVERSITY OF
Southampton

Introduction to Typescript

COMP3207/COMP6244: Cloud Application Development

Dr A. Rezazadeh

October 19

Session Objectives and takeaways

- JavaScript
- Why TypeScript?
- What is TypeScript?
- How to get started fast
- What's new in TypeScript

JavaScript

- Created by Netscape in 1995 as an extension of HTML for Netscape Navigator 2.0.
  - JavaScript had as its main function the manipulation of HTML
  - But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript.

Experts' View on classical JavaScript

"JavaScript is the assembly language of the Web."

Erik Meijer (software architect)

"You can write large programs in JavaScript. You just can't maintain them."

Anders Hejlsberg (father of C#)

Modern JavaScript vs classical JavaScript

- Large scale development in vanilla JavaScript is hard.
- not designed as a programming language for big application
- does not have static typing
- lack structuring mechanisms like classes, modules, interfaces
- JavaScript is notorious for being the world's most misunderstood programming language.
- ECMAScript is the specification on which JavaScript is based on.
  - As a specification, it means ECMAScript is a blueprint to which JavaScript engines (implementations) must adhere to.
  - ES6 (or ECMAScript 2015), ES7 (or ECMAScript 2016), ES8 (or ECMAScript 2017) and ES.Next.

JavaScript vs TypeScript

- **TypeScript** is a typed superset of JavaScript that *transpiled* (converte to plain JavaScript.
- **TypeScript** is a language for application-scale JavaScript development.
- **JavaScript is TypeScript**: Whatever code is written in JavaScript can be converted to TypeScript by changing the extension from **.js** to **.ts**.

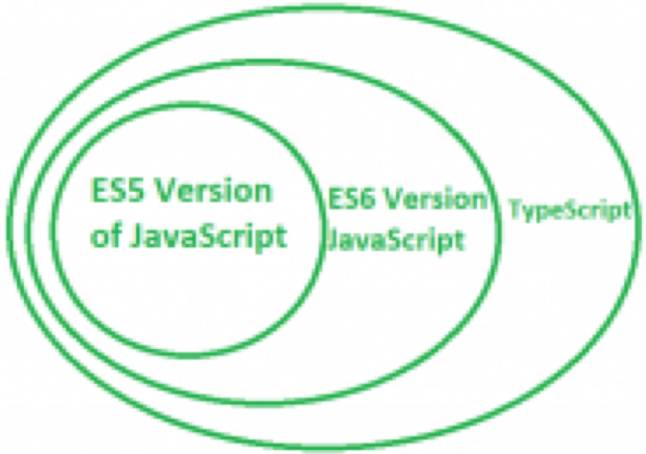
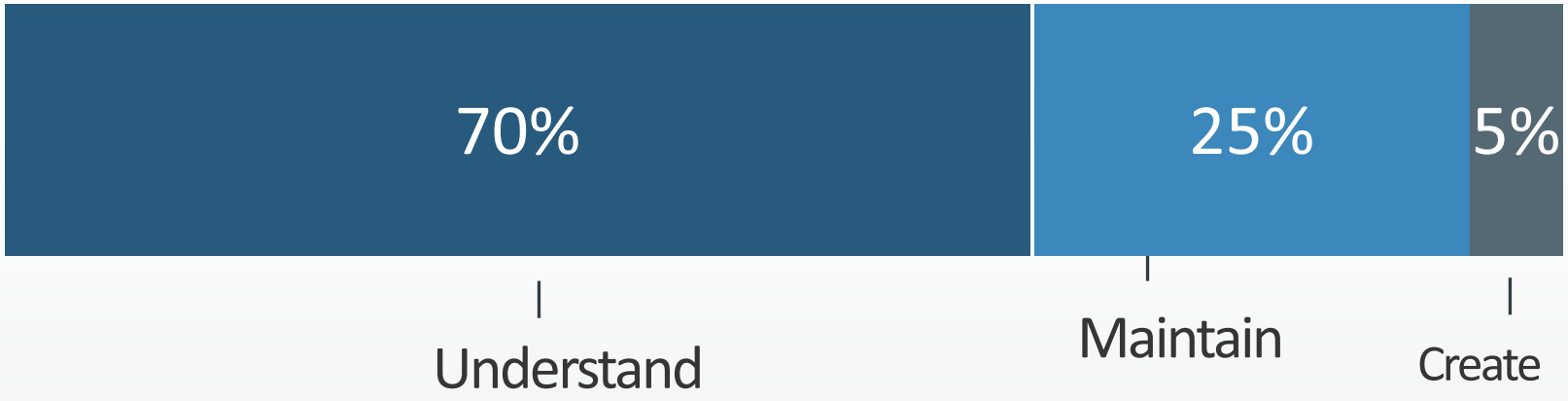


Fig: TypeScript is a SuperSet of JavaScript

Why TypeScript?

Why TypeScript?

- Developers should be able to focus on creating amazing things
- How Developers Spend Time



- TypeScript lets you focus on creation

TypeScript

- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
  - All JavaScript code is TypeScript code, simply copy and paste
  - All JavaScript libraries work with TypeScript
- Runs in any browser or host, on any OS, completely open source.
- Optional static types, classes, and modules
  - Enable scalable application development and excellent tooling
  - Zero cost: Static types completely disappear at run-time

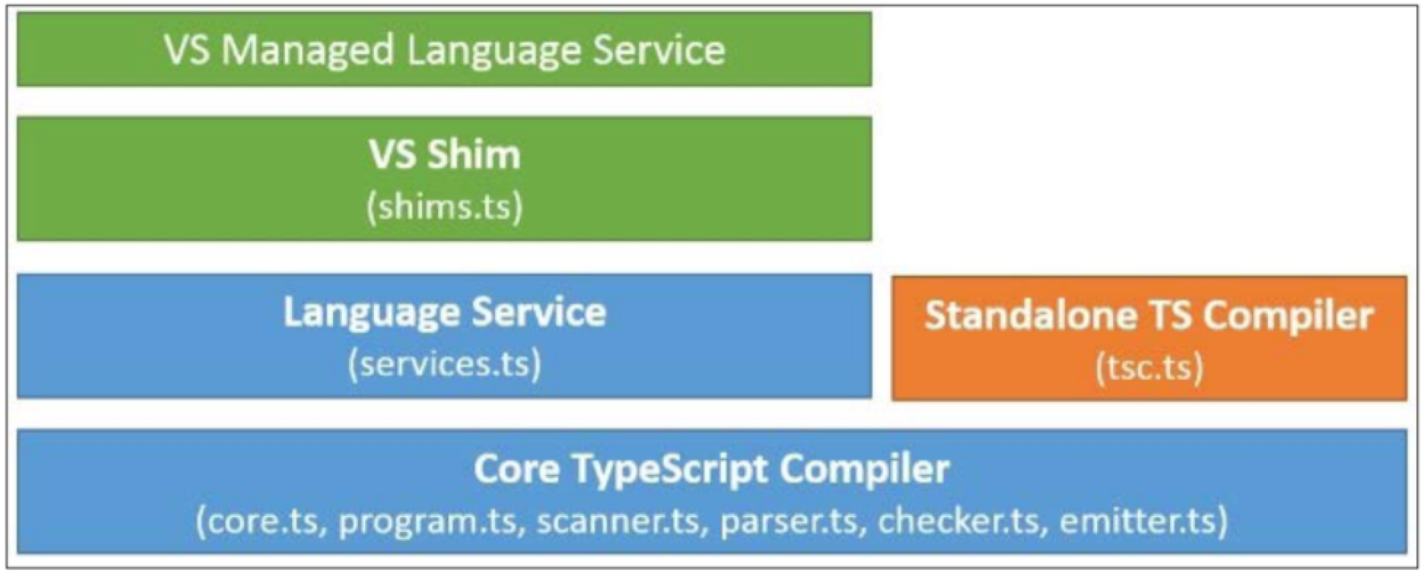
TypeScript – Highlights

- TypeScript impose no runtime overhead on emitted programs:
  - It is common to differentiate between **design** time and **execution** time when thinking about TypeScript.
  - We use the term **design** time or compile time to refer to the TypeScript code that we write while designing an application
  - We use the term **execution** time or runtime to refer to the JavaScript code executed after compiling some TypeScript code.
  - TypeScript adds some features to JavaScript, but those features are only available at design time.

TypeScript – Highlights

- Typescript generated code is highly compatible with web browsers as it targets the ECMAScript 3 specification by default.
- It also supports ES5, ES6, ES7, ES8 and so on.
- In general, we can use the TypeScript features when compiling to any of the available compilation targets
- Some features will require ECMAScript 5 or a higher version as the compilation target.

TypeScript Components



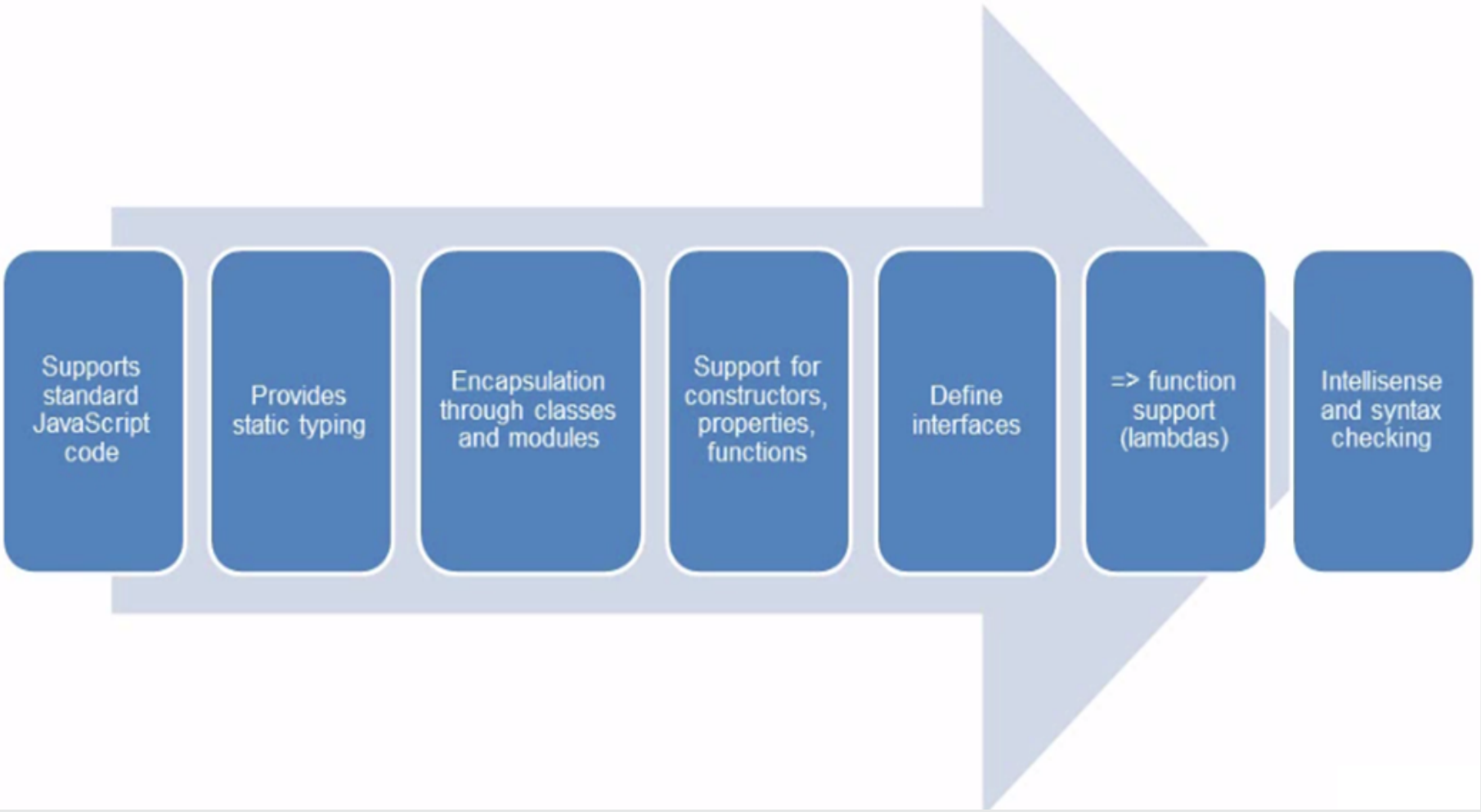
- The TypeScript language is internally divided into three main layers.
- Each of these layers is, in turn, divided into sublayers or components.

- **Language**: Features the TypeScript language elements.
- **Compiler**: Performs the parsing, type checking, and transformation of your TypeScript code to JavaScript code.
- **Language services**: Generates information that helps editors and other tools provide better assistance features, such as IntelliSense or automated refactoring.
- **IDE integration (VS Shim)**: The developers of the IDEs and text editors must perform some integration work to take advantage of the TypeScript features.

TypeScript – Highlights

- **TypeScript Language**
  - Type Annotation
  - Any and Primitive Type
  - Interface, Implementation
  - Class, constructor
  - Opt. Parameter, overloads
  - Event handler
  - Get accessor, private, static
  - Arrow function, lambda
  - Module
  - Typed definitions
  - And more…
- **Tool Features**
  - Type Inference
  - Intellisense, statement comp.
  - Compile on Save
  - Preview Pane
  - ECMAScript version
  - Redirect JavaScript output
  - Generate declaration files

What does TypeScript offer ?

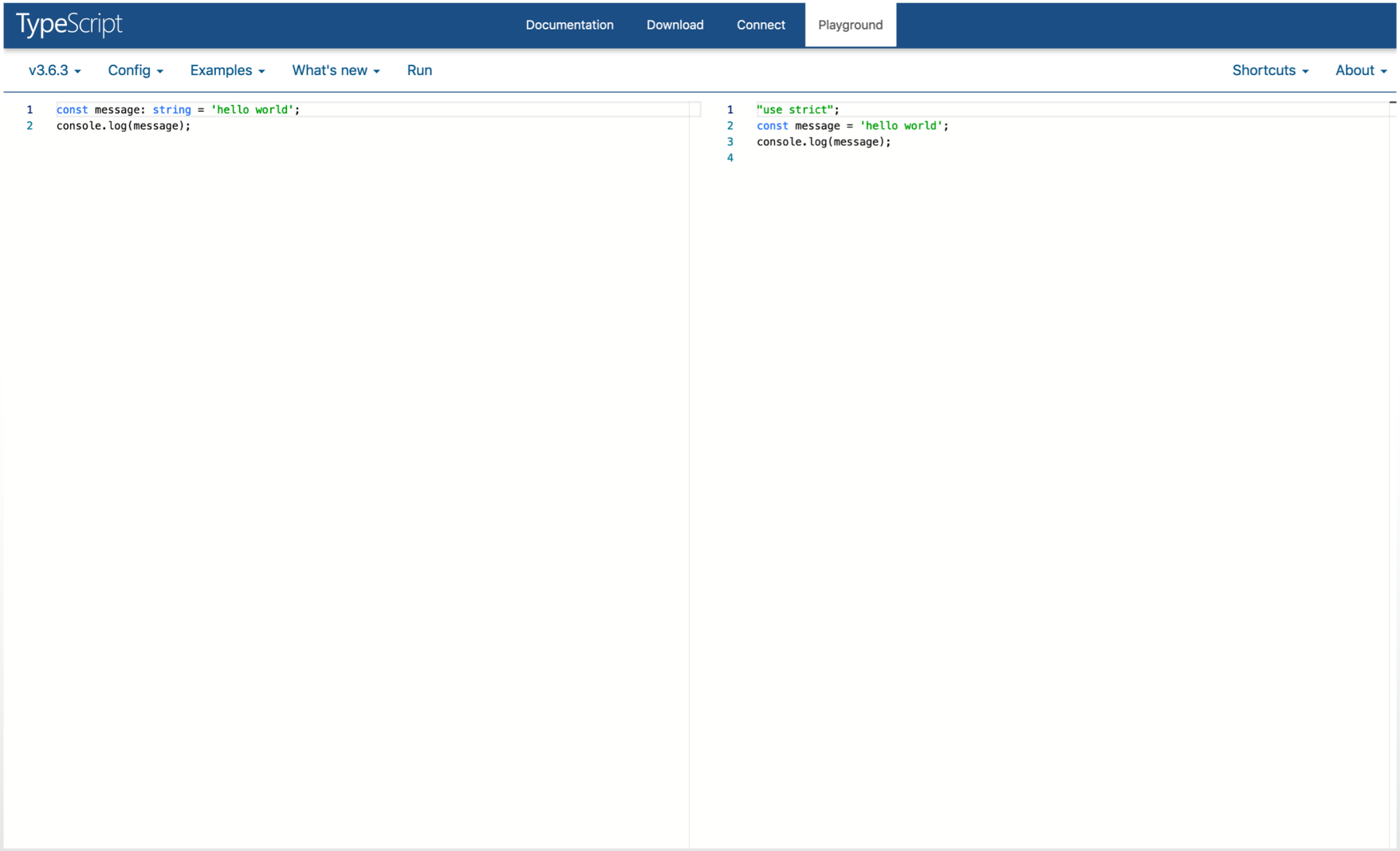


TypeScript language features

- Now that you have learned about the purpose of TypeScript, it's time to get our hands dirty and start writing some code.
- Before you can start learning how to use some of the basic TypeScript building blocks, you will need to set up your development environment.
- The easiest and fastest way to start writing some TypeScript code is to use the online editor available on the official TypeScript website at https://www.typescriptlang.org/play/

  *Called **Playground***, as you can see in the following screenshot:

TypeScript Playground



TypeScript – Working Offline

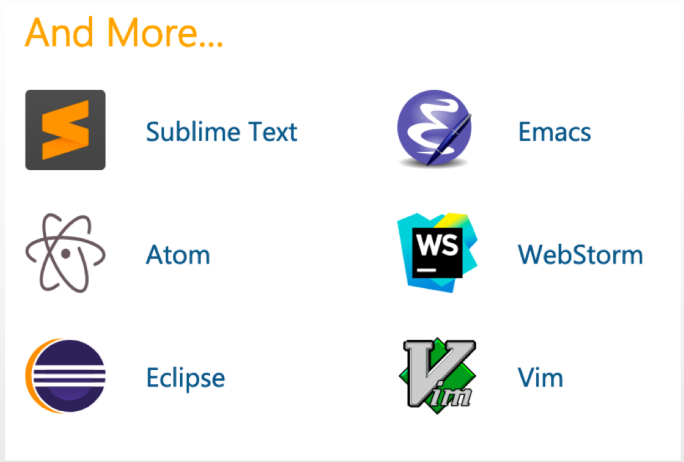
- Download Visual Studio 2019 or Visual Studio 2017 or Visual Studio Code for MacOS.
- Go to **Node.js** Download Website, download your suitable version of Node and install it. After installing **Node.js:** The command-line TypeScript compiler can be installed as a Node.js package.

INSTALL

```
npm install -g typescript
```

COMPILE

```
tsc helloworld.ts
```



Basic Features of TypeScript

- **Types:** As we have already learned, TypeScript is a typed superset of JavaScript.
- **Interfaces:** All JavaScript objects can be described with interfaces
- **Classes:**
  - Allows using common OOP idioms
  - Compatible with future versions of JavaScript
- **Generics**: Lightweight reusable code with no runtime overhead
- **Modules:**
  - Group related interfaces/classes/functions together
  - Move code out of the global namespace

TypeScript – Quick Start

- Install globally TypeScript

```
npm install -g typescript
```

- Open your IDE (Visual Studio Code understands TS)
- Create your first ".ts" file
- Write some TypeScript
- Run the compiler

```
tsc myfile.ts
```

- You have JavaScript

Introducing TypeScript

- Lets start with simple JavaScript

```
function Greeter(greeting) {
    this.greeting = greeting;
}

Greeter.prototype.greet = function() {
    return "Hello, " + this.greeting;
}

let greeter = new Greeter("world");

let greeting = greeter.greet();

console.log(greeting)
```

Introducing TypeScript

- Adding types

```
function Greeter(greeting: string) {
    this.greeting = greeting;
}

Greeter.prototype.greet = function() {
    return "Hello, " + this.greeting;
}

let greeter = new Greeter("world");

let greeting = greeter.greet();

console.log(greeting)
```

Introducing TypeScript- Adding Classes

```
class Greeter {
  private greeting: string;

  constructor (greeting: string) {
    this.greeting = greeting
  }

  greet () {
    return 'Hello, ' + this.greeting
  }
}

let greeter = new Greeter('world')

let greeting = greeter.greet()

console.log(greeting)
```

Introducing TypeScript- Adding Interfaces

```
interface Person {
  firstName: string,
  lastname: string
}

function sayHello (person: Person) {
  return 'Hello from ' + person.firstName + ' ' + person.lastname
}

let hello = sayHello({
  firstName: ‘Jack',
  lastname: ‘Handerson'
})

console.log(hello)
```

Introducing TypeScript- Adding Unions

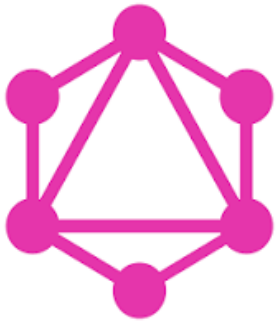
```
type nameOrNameArray = string | string[];

function sayHello (name: nameOrNameArray) {
  if (typeof name === 'string') {
    console.log(name);
  } else {
    console.log(name.join(' '));
  }
}

sayHello([‘Jack', ‘Handerson'])
```

Introducing TypeScript - Other Features

- private, public or protected modifiers
- readonly modifier
- get or set for properties
- static methods and properties
- abstract classes
- enum data type
- and many more here: https://www.typescriptlang.org/docs/handbook/basic-types.html

Frameworks Adopted TypeScript

Angular



GraphQL

React

Vue.js

NativeScript

Tools Supporting TypeScript

Visual Studio


Visual Studio 2017


Visual Studio Code


Visual Studio 2015

And More...


Sublime Text


Atom


Eclipse


Emacs

WebStorm

Vim

Types

- As we have already learned, TypeScript is a **typed superset** of JavaScript.
- TypeScript added **optional static type annotations** to JavaScript in order to transform it into a strongly typed programming language.
- The optional static type annotations are used as **constraints** on program **entities** such as **functions**, **variables**, and **properties** so that compilers and development tools can offer better **verification** and assistance (such as IntelliSense) during software **development**.

Optional Static Type Notation

- The language element that allows us to declare the type of a variable is known as **optional static type notation**.
- For a variable, the type notation comes after the variable name and is preceded by a colon:

```
var counter;                    // unknown (any) type
var counter = 0;                // number (inferred)
var counter : number;           // number
var counter : number = 0;       // number
```

Variables, basic types, and operators

Data Type	Description
Boolean	Whereas the string and number data types can have a virtually unlimited number of different values, the Boolean data type can only have two. They are the literals `true` and `false`. A Boolean value is a truth value; it specifies whether the condition is true or not. `var isDone: boolean = false;`
Number	As in JavaScript, all numbers in TypeScript are floating point values. These floating-point numbers get the type `number`. `var height: number = 6;`

String	You use the string data type to represent text in TypeScript. You include string literals in your scripts by enclosing them in single or double quotation marks. Double quotation marks can be contained in strings surrounded by single quotation marks, and single quotation marks can be contained in strings surrounded by double quotation marks.
	`var name: string = "bob";`
	`name = 'smith';`

Array	TypeScript, like JavaScript, allows you to work with arrays of values.
	Array types can be written in one of the two ways. In the first, you use the type of the elements followed by `[]` to denote an array of that element type:
	`var list:number[] = [1, 2, 3];`
	The second way uses a generic array type, `Array`:
	`var list:Array<number> = [1, 2, 3];`

Enum	An enum is a way of giving more friendly names to sets of numeric values. By default, enums begin numbering their members starting at 0, but you can change this by manually setting the value of one to its members.
	`enum Colour {Red, Green, Blue};`
	`var c: Colour = Colour.Green;`

Variables, basic types, and operators

Data Type	Description

Any	The any type is used to represent any JavaScript value. A value of the any type supports the same operations as a value in JavaScript and minimal static type checking is performed for operations on any values.
	`var notSure: any = 4;  notSure = "maybe a string instead";  notSure = false; // okay, definitely a boolean` The any type is a powerful way to work with existing JavaScript, allowing you to gradually opt in and opt out of type checking during compilation. The any type is also handy if you know some part of the type, but perhaps not all of it. For example, you may have an array but the array has a mix of different types:
	`var list:any[] = [1, true, "free"];  list[1] = 100;`

Void	The opposite in some ways to any is `void`, the absence of having any type at all. You will see this as the return type of functions that do not return a value. `function warnUser(): void {  alert("This is my warning message"); }`

Other Types

- JavaScript's primitive types also include ***undefined*** and ***null***.
- In JavaScript, ***undefined*** is a property in the **global scope** that is assigned as a value to variables that have been **declared** but have **not yet been initialized**.
- The value ***null*** is a literal (not a property of the global object). It can be assigned to a variable as a representation of no value.

Examples

```
var TestVar;  // variable is declared but not initialized
alert(TestVar);     // shows undefined
alert(typeof TestVar); // shows undefined
var TestVar = null;    // variable is declared and value
null is   assigned as value
alert(TestVar);        // shows null
alert(typeof TestVar); // shows object
```

- In TypeScript, we will not be able to use `null` or `undefined` as types:

```
var TestVar : null; // Error, Type expected
var TestVar : undefined; // Error, cannot find name
undefined
```

Var, *let*, and *const*

- When we declare a variable in TypeScript, we can use the `var`, `let`, or `const` keywords:

```
var mynum : number = 1;

let isValid : boolean = true;

const apiKey : string = "0E5CE8BD-6341-4CC2-904D-C4A94ACD276E";
```

- Variables declared with `var` are scoped to the nearest function block (or global, if outside a function block).
- Variables declared with `let` are scoped to the nearest enclosing block (or global if outside any block), which can be smaller than a function block.
- The `const` keyword creates a constant that can be global or local to the block in which it is declared. This means that constants are block scoped. You will learn more about scopes later.

Union types

- TypeScript allows you to declare union types:

```
var path : string[] | string;
path = '/temp/log.xml';
path = ['/temp/log.xml', '/temp/errors.xml'];
path = 1; // Error
```

- Union types are used to declare a variable that is able to store a value of **two** or **more** types.
- In example, we have declared a variable named path that can contain a single path (string), or a collection of paths (array of string).
- We assigned a string and an array of strings without errors;
  - However we got a compilation error when assign a numeric value because the union type didn't declare a number as one of the valid types of the variable.

Type Guards

- We can examine the type of an expression at runtime by using the `typeof` or `instanceof` operators. The TypeScript language service looks for these operators and will change type inference accordingly when used in an `if` block:

```
var x: any = { /* ... */ };
if(typeof x === 'string') { console.log(x.splice(3, 1)); //
Error, 'splice' does not exist on 'string'
}
// x is still any
x.foo(); // OK
```

- If the type of x results to be string, we will try to invoke the method splice, which is supposed to a member of the x variable. TypeScript will automatically assume that x must be a string and let us know that the splice method does not exist on the type string. This feature is known as **type guards**.

Type Aliases

- TypeScript allows us to declare type aliases by using the type keyword:

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

- Type aliases are exactly the same as their original types; they are simply alternative names.
- Type aliases can help us to make our code more readable but it can also lead to some problems.
  - If you work as part of a large team, the indiscriminate creation of aliases can lead to maintainability problems.

Arithmetic operators

- There following arithmetic operators are supported by the TypeScript programming language. In order to understand the examples, you must assume that variable ***A*** holds **10** and variable ***B*** holds **20**.

Operator	Description	Example
+	This adds two operands	A + B will give 30
-	This subtracts the second operand from the first	A - B will give -10
*	This multiplies both the operands	A * B will give 200
/	This divides the numerator by the denominator	B / A will give 2
%	This is the modulus operator and remainder after an integer division	B % A will give 0
++	This is the increment operator that increases the integer value by 1	A++ will give 11
--	This is the decrement operator that decreases the integer value by 1	A-- will give 9

Comparison Operators

- ***A*** holds **10** and variable ***B*** holds **20**.

Operator	Description	Example
==	This checks whether the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is false. A == "10" is true.
===	This checks whether the value and type of two operands are equal or not. If yes, then the condition becomes true.	A === B is false. A === "10" is false.
!=	This checks whether the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.

Comparison Operators

- ***A*** holds **10** and variable ***B*** holds **20**.

Operator	Description	Example
>	This checks whether the value of the left operand is greater than the value of the right operand; if yes, then the condition becomes true.	(A > B) is false.
<	This checks whether the value of the left operand is less than the value of the right operand; if yes, then the condition becomes true.	(A < B) is true.
>=	This checks whether the value of the left operand is greater than or equal to the value of the right operand; if yes, then the condition becomes true.	(A >= B) is false.
<=	This checks whether the value of the left operand is less than or equal to the value of the right operand; if yes, then the condition becomes true.	(A <= B) is true.

Logical operators

- ***A*** holds **10** and variable ***B*** holds **20**.

Operator	Description	Example
&&	This is called the logical AND operator. If both the operands are nonzero, then the condition becomes true.	(A && B) is true.
||	This is called logical OR operator. If any of the two operands are nonzero, then the condition becomes true.	(A || B) is true.
!	This is called the logical NOT operator. It is used to reverse the logical state of its operand. If a condition is true, then the logical NOT operator will make it false.	!(A && B) is false.

Bitwise Operators

- Assume ***A*** holds **2** and variable ***B*** holds **3**.

Operator	Description	Example
&	This is called the Bitwise AND operator. It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2
|	This is called the Bitwise OR operator. It performs a Boolean OR operation on each bit of its integer arguments.	(A | B) is 3
^	This is called the Bitwise XOR operator. It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A ^ B) is 1
~	This is called the Bitwise NOT operator. It is a unary operator and operates by reversing all bits in the operand.	(~B) is -4
<<	This is called the Bitwise Shift Left operator. It moves all bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying by 2, shifting two positions is equivalent to multiplying by 4, and so on.	(A << 1) is 4
>>	This is called the Bitwise Shift Right with sign operator. It moves all bits in its first operand to the right by the number of places specified in the second operand.	(A >> 1) is 1
>>>	This is called the Bitwise Shift Right with zero operators. This operator is just like the >> operator, except that the bits shifted in on the left are always zero,	(A >>> 1) is 1

Bitwise Operators

- One of the main reasons to use bitwise operators in languages such as C++, Java, or C# is that they're extremely fast.

- In TypeScript and JavaScript Bitwise operators are less efficient
  - This is because it is necessary to cast from floating point representation (how JavaScript stores all of its numbers) to a 32-bit integer to perform the bit manipulation and back.

Assignment operators

Operator	Description	Example
=	This is a simple assignment operator that assigns values from the right-side operands to the left-side operand.	C = A + B will assign the value of A + B into C
+=	This adds the AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C+ A
-=	This subtracts the AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C- A
*=	This multiplies the AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C* A
/=	This divides the AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C/ A
%=	This is the modulus AND assignment operator. It takes the modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C% A

Flow control statements

- The following code snippet declares a variable of type Boolean and name `isValid`. Then, an `if` statement will check whether the value of `isValid` is equal to `true`. If the statement turns out to be true, the `Is valid!` message will be displayed on the screen.

```
var isValid : boolean = true;

  if(isValid) {
    alert("is valid!");
  }
```

The double-selection structure (if...else)

- If the statement turns out to be true, the message `Is valid!` will be displayed on the screen. On the other side, if the statement turns out to be false, the message `Is NOT valid!` will be displayed on the screen.

```
var isValid : boolean = true;

if(isValid) {

   alert("Is valid!");
}
else {

   alert("Is NOT valid!");
}
```

The Inline Ternary Operator (?)

- The inline ternary operator is just an alternative way of declaring a double-selection structure.

```
var isValid : boolean = true;
var message = isValid ? "Is valid!" : "Is NOT
valid!";
alert(message);
```

- It checks whether the variable or expression on the left-hand side of the operator `?` is equal to true.
- If the statement turns out to be true, the expression on the left-hand side of the character will be executed and the message `Is valid!` will be assigned to the message variable.
- If the statement is false, the expression on the right-hand side of the operator will be executed and the message, `Is NOT valid!` will be assigned to the message variable.

The multiple-selection structure (switch)

- The switch statement evaluates an expression, matches the expression's value to a case clause, and executes statements associated with that case.
- A switch statement and enumerations are often used together to improve the readability of the code.

```
enum AlertLevel{

   info,
   warning,
   error
}
```

The multiple-selection structure (switch)

```
function getAlertSubscribers(level : AlertLevel){
     var emails = new Array<string>();
     switch(level){
       case level.info:
          emails.push("cst@domain.com");
          break;
       case level.warning:
          emails.push("development@domain.com");
          emails.push("sysadmin@domain.com");
          break;
       case level.error:
         emails.push("development@domain.com");
         emails.push("sysadmin@domain.com");
         emails.push("management@domain.com");
         break;
       default:
         throw new Error("Invalid argument!");
}
     return emails;
   }
```

Loops

- While loop - Expression is tested at the top of the loop

```
var i : number = 0;
while (i < 5) {
    i += 1;
console.log(i);
}
```

- **do...while - expression is tested at the bottom of the loop**

```
var i : number = 0;

do {

    i += 1;
   console.log(i);
} while (i < 5);
```

Unlike the `while` loop, the `do-while` expression will execute at least once regardless of the requirement value as the operation will take place before checking if a certain requirement is satisfied or not.

Counter controlled repetition (for)

- The `for` statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement or a set of statements executed in the loop.

```
for (var i: number = 0; i < 9; i++) {
    console.log(i);
}
```

Functions

- Just as in JavaScript, TypeScript functions can be created either as a **named** function or as an **anonymous** function.

```
// named function
function greet(name? : string) : string {
  if(name){
    return "Hi! " + name;
  }
  else
  {
    return "Hi!";
  }
}
```

```
// anonymous function
var greet = function(name? : string) : string {
  if(name){
    return "Hi! " + name;
  }
  else
  {
    return "Hi!";
  }
}
```

The arrow (=>) operator

- There is an alternative function syntax, which uses the arrow (=>) operator after the function's return type and skips the usage of the `function` keyword.

```
var greet = (name?: string) : string => {
    if(name){
      return "Hi! " + name;
    }
    else
    {
      return "Hi! my name is " + this.fullname;
    }
};
```

Classes

- ECMAScript 6 of JavaScript, adds class-based object orientation to JavaScript.
- TypeScript also allows developers to use class-based object orientation.
- When we declare a class in TypeScript, all the methods and properties are public by default.

Class

```
class Character {
  fullname : string;
  constructor(firstname : string, lastname : string) {
    this.fullname = firstname + " " + lastname;
  }
  greet(name? : string) {
    if(name)
    {
      return "Hi! " + name + "! my name is " + this.fullname;
    }
    else
    {
      return "Hi! my name is " + this.fullname;
    }
  }
}

var spark = new Character("Jacob","Keyes");
var msg = spark.greet();
alert(msg); // "Hi! my name is Jacob Keyes"
var msg1 = spark.greet("Dr. Halsey");
alert(msg1); // "Hi! Dr. Halsey! my name is Jacob Keyes"
```

Interfaces

- In TypeScript, we can use interfaces to enforce that a class follow the specification in a particular contract.

```
interface LoggerInterface{
  log(arg : any) : void;
}
```

```
class Logger implements LoggerInterface{
  log(arg){
    if(typeof console.log === "function"){
      console.log(arg);
    }
    else
    {
      alert(arg);
    }
  }
}
```

Interfaces

```
interface UserInterface{
  name : string;
  password : string;
}
```

```
var user : UserInterface = {
  name : "",
  password : "" // error property password is missing
};
```

Putting Everything Together – Modules

```
module Geometry{
  export interface Vector2dInterface {
    toArray(callback : (x : number[]) => void) : void;
    length() : number;
    normalize();
  }
  export class Vector2d implements Vector2dInterface {
    private _x: number;
    private _y : number;
    constructor(x : number, y : number){
      this._x = x;
      this._y = y;
    }
    toArray(callback : (x : number[]) => void) : void{
      callback([this._x, this._y]);
    }
    length() : number{
      return Math.sqrt(this._x * this._x + this._y * this._y);
    }
    normalize(){
      var len = 1 / this.length();
      this._x *= len;
      this._y *= len;
    }
  }
}
```

Resources

- Getting Started with TypeScript

  https://blog.teamtreehouse.com/getting-started-typescript

- TypeScript Documentation

  http://www.typescriptlang.org/docs/home.html

  Typescript Quick Start
  https://www.typescriptlang.org/docs/tutorial.html

- TypeScript Deep Dive (free ebook)

  https://basarat.gitbooks.io/typescript/content/

  https://github.com/basarat/typescript-book

Resources – Cont.

- What's new in TypeScript

  https://www.typescriptlang.org/docs/handbook/release-notes/overview.html

  TypeScript Roadmap

  https://github.com/Microsoft/TypeScript/wiki/Roadmap