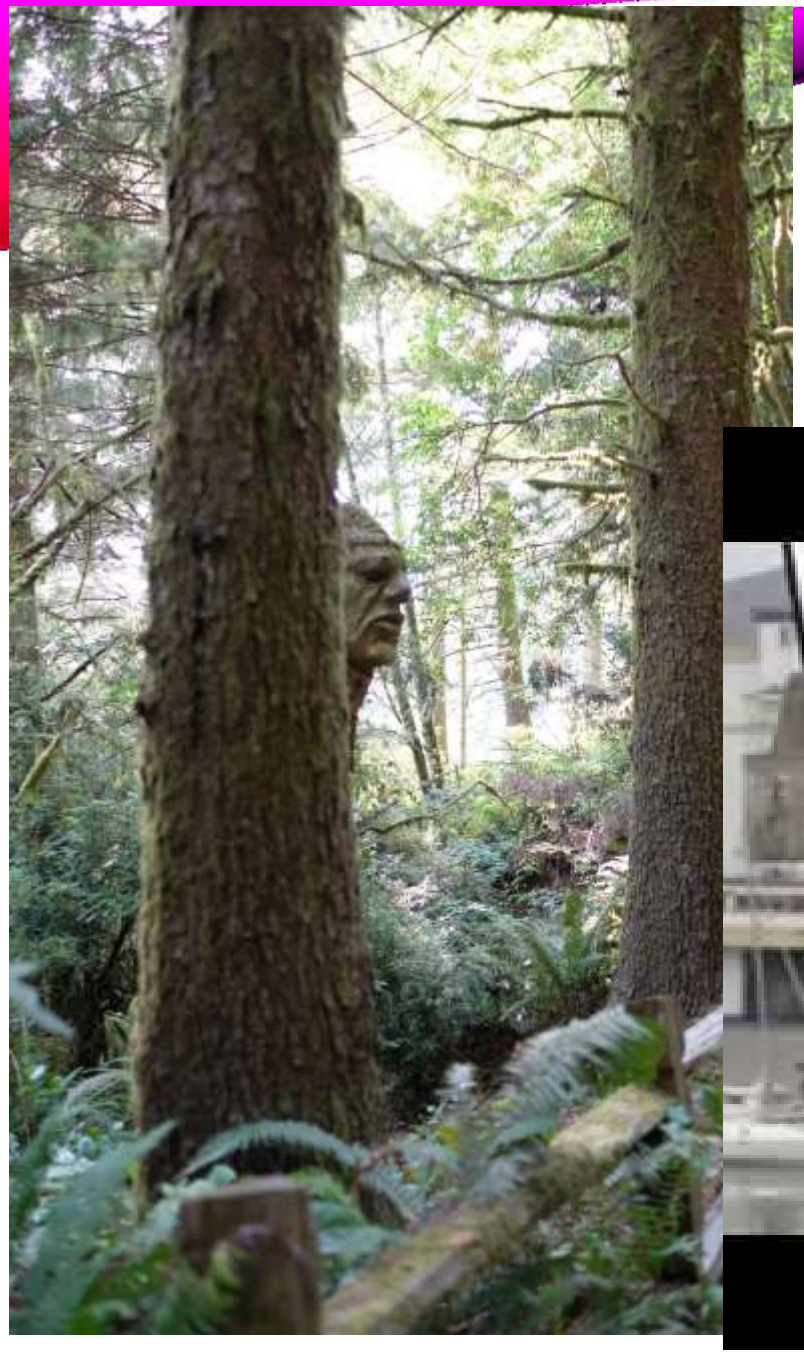
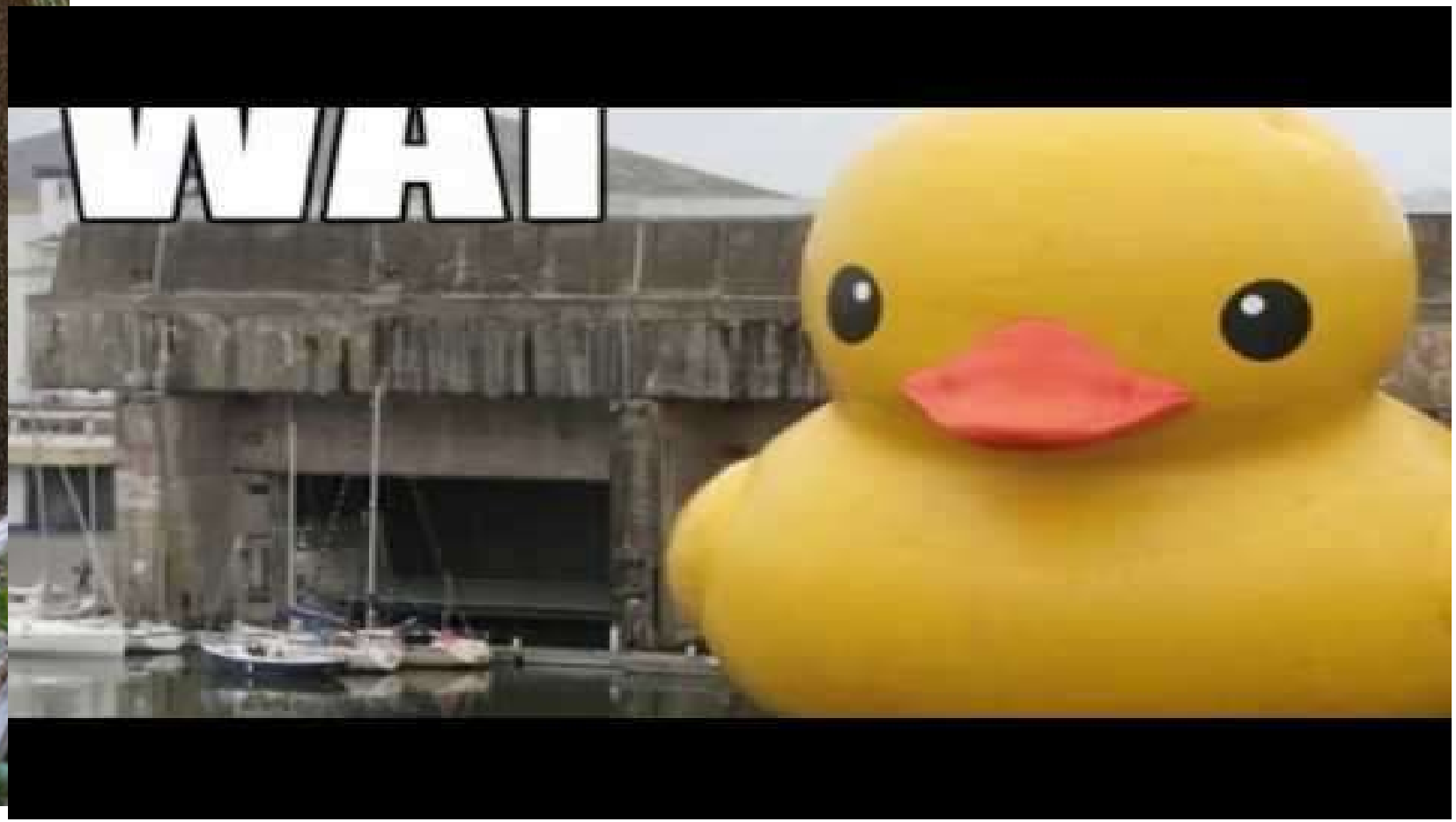

TYPESCRIPT
Presented by Clarke Bowers
Ecola Beach Oregon

ABOUT THE PRESENTER

- Clarke D. Bowers
- http://www.cbsoftwareengineering.com/
- mailto: clarke@cbsoftwareengineering.com
- 35 years of industry experience
- Has developed everything from embedded systems using assembly language to enterprise data warehouses and everything in between…

SCRIPTING WAT



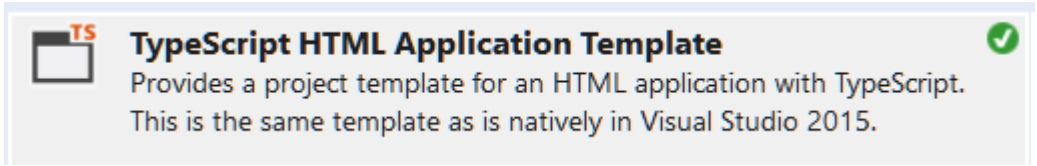


WHAT IS TYPESCRIPT?

- TypeScript extends from the syntax and semantics of JavaScript. Full interop between TS and JS since TS compiles into JS.
- TypeScript runs on browsers, in Node.js, or in any JavaScript engine that supports ECMAScript 3 (or newer). TypeScript offers support for the latest JavaScript features (ECMAScript 2015).
- Types enable JS developers to use highly-productive development tools and practices like static checking and code refactoring when developing JavaScript applications. TS allows Intellisense to work in Visual Studio.
- Types and type inference allows a few type annotations to make static verification possible. Types let you define interfaces between software components.

HOW DO I GET IT?

- https://www.typescriptlang.org/
  - Version 2.7 is current



- Built into Visual Studio 2015 SP3 and 2017
  - Project template for learning, VS2017, search online for *TypeScript*
  - https://richnewman.wordpress.com/2017/05/09/html-application-with-typescript-project-template-for-visual-studio-2017/
- NodeJs
  - npm install -g typescript
- OData 4: http://www.odata.org/documentation/
- Also add *postman* to *Chrome,* It will be handy for one of the examples

HELLO WORLD

- Create text file with *ts* extension
- Compile with *tsc*
  - Usage: tsc [options] [file ...]
- Execute with *node*
  - Usage: node [options] [-e script | script.js | -] [arguments]

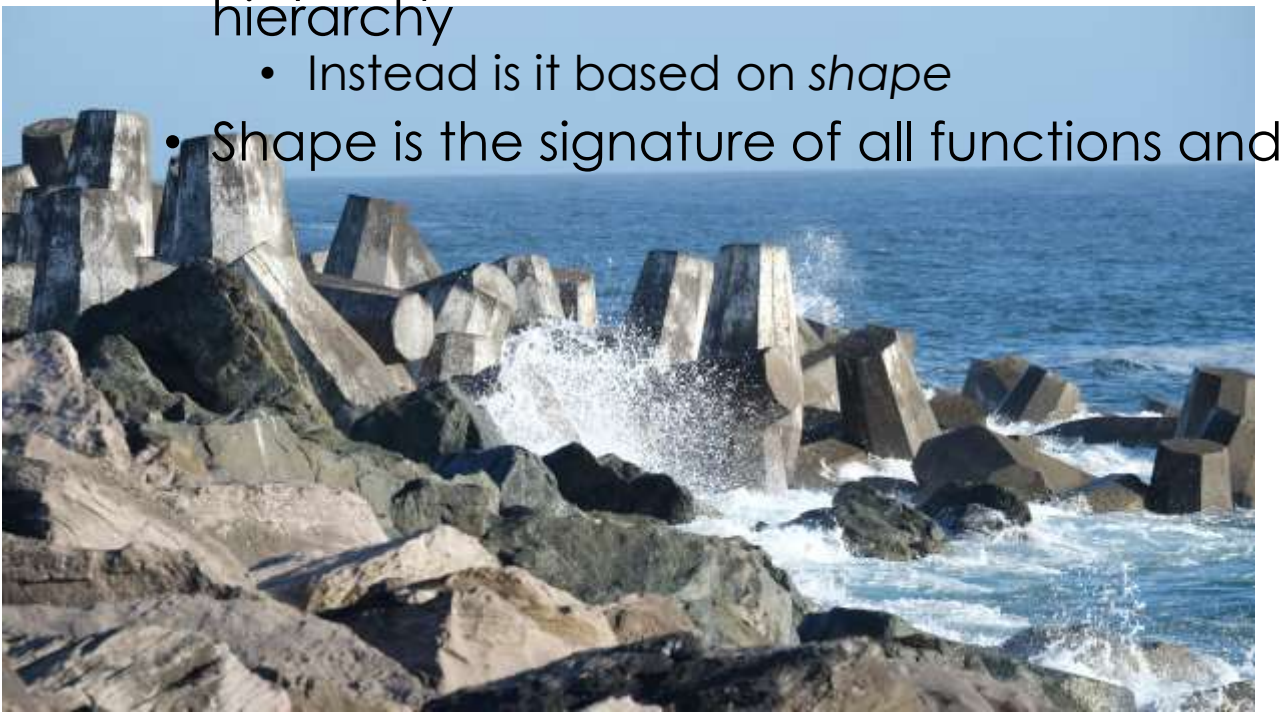
```
console.log("hello world!");
```

```
C:\...\NodeExamples>tsc HelloWorld.ts
```

```
C:\...\NodeExamples>node HelloWorld.js
hello world!
```

INHERITANCE & POLYMORPHISM

- Supports classes and interfaces
- Use keyword *extends* to perform subclassing
- Use keyword *implements* to support a interface
- Polymorphism is not based on class or interface hierarchy
  - Instead is it based on *shape*
- Shape is the signature of all functions and fields



```
class Animal {
    name: string;
    constructor(theName: string) { this.name = theName; }
    move(distanceInMeters: number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}
//inheritance
class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 5) {
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}
```

VISUAL STUDIO CLOCK PROJECT

- Template for TypeScript project
- Shows the current time
- Updates ever 500ms
- Starts with a naked HTML page
- Adds elements in the constructor
- Update code to
  - Add a button
  - Hook the event handler to stop the clock
  - Note that *this* is not implicit so be careful how to call a method on it.
  - Timer token is not defined

```
class Greeter {
    element: HTMLElement;
    span: HTMLElement;
    timerToken: number;
    button: HTMLElement;

constructor(element: HTMLElement) {
    this.element = element;
    this.element.innerHTML += "The time is: ";
    this.span = document.createElement('span');
    this.element.appendChild(this.span);
    this.span.innerText =
            new Date().toUTCString();
```


McArthur-Burney Falls

ASP.NET & ENTITY FRAMEWORK

- Create a ASP.NET Web Application
  - Select Empty
  - Checked are WebApi
  - Select No Authentication
- Add Entity Framework Nuget package
- Restore the AdventureWorks DB to local SQL Server 2016
- Add connection string to web.config
- Add Address, Person and EmailAddress to model

```
<connectionstrings>
   <add
       name="AdventureWorks"
       connectionString= “
    Data Source=TALLTOWER\MSSQLSERVER2016;
     Initial Catalog=AdventureWorks2016CTP3;
     Integrated Security=True"
     />
</connectionstrings>
```

ASP.NET & ODATA

- Create a Data Model from the Adventure Works DB
- Create a new controller of type ODataController
- Return IQueryable of your entity

```
public class AddressController : ODataController
{
    private AdventureWorksContext db =
        new AdventureWorksContext();

    [EnableQuery]
    public IQueryable<Address> Get()
    {
        return db.Address;
    }
```

ODATA STARTUP

- Adverting end point: http://localhost:61068/odata/$metadata/
  - Remove other end-points from WebApiConfig.cs
- Query syntax
  - http://localhost:61068/odata/Address?$filter=City eq 'Bothell‘
  - http://localhost:61068/odata/Person(1)
  - http:// localhost:61068/odata/Person?$filter=FirstName eq 'Ken'&$expand=EmailAddress

```
ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
builder.EntitySet<Address>("Addresses");
builder.EntitySet<Person>("People");
//include all entities even when they have no endpoint
builder.EntitySet<EmailAddress>("EmailAddresses");
config.Routes.MapODataServiceRoute("ODataRouter", "odata", builder.GetEdmModel());
config.AddODataQueryFilter();
```

STRONGLY TYPED ODATA

- Add address class for Odata JSON deserialization
- FuryTech. OdataTypescriptServiceGenerator
  - Scaffolding TypeScript model classes and services from an OData Metadata XML
  - https://github.com/FuryTechs/FuryTech.OdataTypescriptServiceGenerator

```
class Address {
    public AddressID: number;
    public AddressLine1:
      string;
    public City: string;
};
```

ODATA REQUEST

- Use strongly typed XMLHttpRequest to query with Odata
- Parse the response from JSON into an JS object
- Grab a strongly typed array of addresses

```
load() {
    let Http = new XMLHttpRequest();
    let url = “http://localhost:61069/odata/Address?”
    + “$filter=City eq 'Bothell‘”;
    Http.open('get', url, false);
    Http.send();
    let odataResponse = JSON.parse(Http.response);
    let addresses: Address[] = odataResponse.value;
    this.span.innerHTML = "Id: " + addresses[0].AddressID +
    " Address: " + addresses[0].AddressLine1 +
    " City: " + addresses[0].City;
}
```

MODULES

- Separate files
- Similar the C language header files
- Export the declaration
- Import where you need to reference
- Has the disadvantage of path accuracy
- ECMA Script 2015 required for modules support
  - Otherwise you need a JS library to support

```
export class Address {
    public AddressID: number;
    public AddressLine1:
      string;
    public City: string;
};

import { Address } from
   "./Models/address";
```

TYPE DECLARATION

```
/// <reference path="./Scripts/typings/jquery/jquery.d.ts" />
```

- Libraries can generate a *header* file: *.d.ts
  - Visual Studio Intellisense support these files
- DefinitelyTyped is a repository of type definitions for common JS libraries
  - http://definitelytyped.org/
  - Nuget packages
- Referencing a type definition file adds no JS code from the library file
  - It just provides strong types for the TypeScript compiler
- Very strange syntax to reference a type definition file
  - Triple-slash directives are single-line comments containing a single XML tag. The contents of the comment are used as compiler directives.

AJAX CALLBACK

- Use Jquery Ajax method
- getJSON performs HTTP Get and deserializes into JSON object
- Assign to strongly type variable

```
load() {
    let url = http://localhost:61068/odata/Address? +
        “$filter=City eq 'Bothell'";
    $.getJSON(url, (odataResponse, textStatus, xhr)
      => {
        let addresses: Address[] = odataResponse.value;
        his.span.innerHTML = "Id: " +
          addresses[0].AddressID +
          " Address: " + addresses[0].AddressLine1 +
          " City: " + addresses[0].City;
```

ITERATORS , FOR/OF

- C# foreach equivalent
- TypeScript for…of
- Enumerates a collection

```
public renderHeaders(
      columnHeaders: string[]) {
    let tr = document.createElement("tr");
    this.table.appendChild(tr);
    this.headerRow = tr;
    for (let columnHeader of columnHeaders) {
        let th =
            document.createElement("th");
        th.innerText = columnHeader;
        tr.appendChild(th);
} }
```

GENERICS

- Very similar syntax to C#
- Generic classes
- Generic functions
- Functions overloads work poorly because JavaScipt cannot distinguish

```
public renderData<T>(
    dataRows: T[],
    renderRow: (rowElement: HTMLTableRowElement,
                rowData: T) => void) {
    for (let dataRow of dataRows) {
        if (!this.headerRow) {
            this.renderHeadersFromData(dataRow);
        }
        let tr = document.createElement("tr");
        this.table.appendChild(tr);
        renderRow(tr, dataRow);
    }
```

ITERATORS , FOR/IN

- For/in enumerates *keys*
- Great for object properties
- Works well for arrays indexed by string (maps)

```
public renderHeadersFromData<T>(
        dataRow: T) {
    let tr = document.createElement("tr");
    this.table.appendChild(tr);
    this.headerRow = tr;
    for (let key in dataRow) {
        let th =
            document.createElement("th");
        th.innerText = key;
        tr.appendChild(th);
    }
```


Lake Mono Valley, California