neueda

JavaScript & TypeScript

Objectives

Part 1 - Javascript

- What is JavaScript?
- Variables
- Basic syntax (conditions & looping)
- Working with arrays
- Objects & JSON
- Functions & functional Programming
- Interacting with the DOM

Part 2 - Typescript

- What is TypeScript?
- Nullable variables & type gu
- Generics

Part 3 – Modern JS / TS tech

- The Spread operator
- Destructuring

What is JavaScript?

- Java is to JavaScript as Hedge is to Hedgehog!

- Most JavaScript applications have traditionally been written using '
  JavaScript - ECMAScript 5 (ES5)

- The ECMAScript standard has evolved a lot since then but not all br
  support it… we can use tools to make more modern JavaScript bac
  compatible.

From Java to JavaScript

JAVA

- Compiled to bytecode
- Requires a JVM
- Object oriented
- Statically typed
- Uses access modifiers
- Lots of data types
- Value equality with .equals(), object equality with ==

JAVASCRIPT

- Non-compiled (interpretec
- Requires a runtime enviror
- More like a functional prog language (objects are class
- Dynamically typed (and typ change)
- Everything is public
- Number, String, Boolean. N integer
- Strict equality with === or equivalency with ==

Running JavaScript

- JavaScript needs an **interpreter**, such as a browser
- Within an HTML file we can include a **<script></script>** block or ref external js files
- Use **console.log** to print to the console.
- **;** is optional

Data types

- JavaScript is a **dynamically typed** language.

JavaScript has the following data types:

- number
- boolean
- string
- symbol
- object
- null
- undefined

Declaring variables

- Variables do not have a fixed type
- var x = 6 (old – don't use this!)
- let x = 6 - mutable
- const x = 6 – immutable

- Arrays are defined with [ ] and can contain mixed variable types
- let sizes = ["small", "large", "extra large“, 120]

Strings

- Strings can be declared with **single or double quotes**
- Use **** to escape the quote character if needed
- String interpolation (template literals) requires **back-ticks** and **${}**
- Strings have **functions** such as:
  - trim()
  - toUpperCase() / toLowerCase()
  - indexOf() / lastIndexOf()
  - charAt
- **length** is a property
- Convert a string to a number with **+**

Arrays

- Array sizes can change over time
- Arrays can be defined using:
  - **const colors = [“red”,”green”,”blue”];**
  - **const colors = new Array(“red”,”green”,”blue”);**
- Properties in an array are accessed using **[position]**
- Manipulate arrays with functions:
  - **push** – add a value to the end
  - **pop** – remove (and return) the last value
- Arrays can also use lambda expressions for **filter**, **find**, **map**

Basic Syntax - Loops

```
for (let i = 0; i < 10; i++) {
   console.log(i);
}
```

```
for (let i = 0; i < colors4.length; i++) {
   console.log(colors4[i]);
}
```

```
for (const color of colors4) {
   console.log(color);
}
```

Basic Syntax - Conditions

- Use `===` or `!==` for strict equality
- Use `==` or `!=` for null checking

```
if (color === "RED") {
   console.log("It's red");
} else if (color === "GREEN") {
   console.log("It's green");
} else {
   console.log("It's not red or green");
}
```

```
switch (color) {
   case "RED":
      console.log("It's red
      break;
   case "GREEN":
      console.log("It's gre
      break;
   default:
      console.log("It’s nei
   break;
}
```

Objects

- Objects are a set of key, value pairs
- The values could be simple values, objects or arrays

```
const person = {
        name:'matt',
        age:26,
        foods : ['pizza', 'pasta'],
        preferences: [
            {colour: 'blue'},
            {season: 'summer'}
        ]
      };
```

Classes and Prototypes

- JavaScript is a Prototype language
- All objects inherit from **Object.prototype** by default
- Classes are syntactic sugar for prototypes

```
class Adult {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    birthday() {
        this.age = this.age + 1;
        }
}
```

```
class child extends Adu
    constructor(name, age
        super(name, age);
    this.toys = toys;
    }
}
```

Creating Functions

```
function sayHello(firstname,surname) {
   return('Hello ' + firstname + ' ' + surname);
}

const sayHello = (firstname, surname) => {
 return('Hello ' + firstname + ' ' + surname);
}

const sayHello = (firstname, surname) => 'Hello '
firstname + ' ' + surname
```

neueda

How does JavaScript access interact with HTML elemen

THE DOM

- We can manipulate the visible page from JavaScript code – we trea
  HTML elements as object (Document Object Model)
- Find elements using (for example):

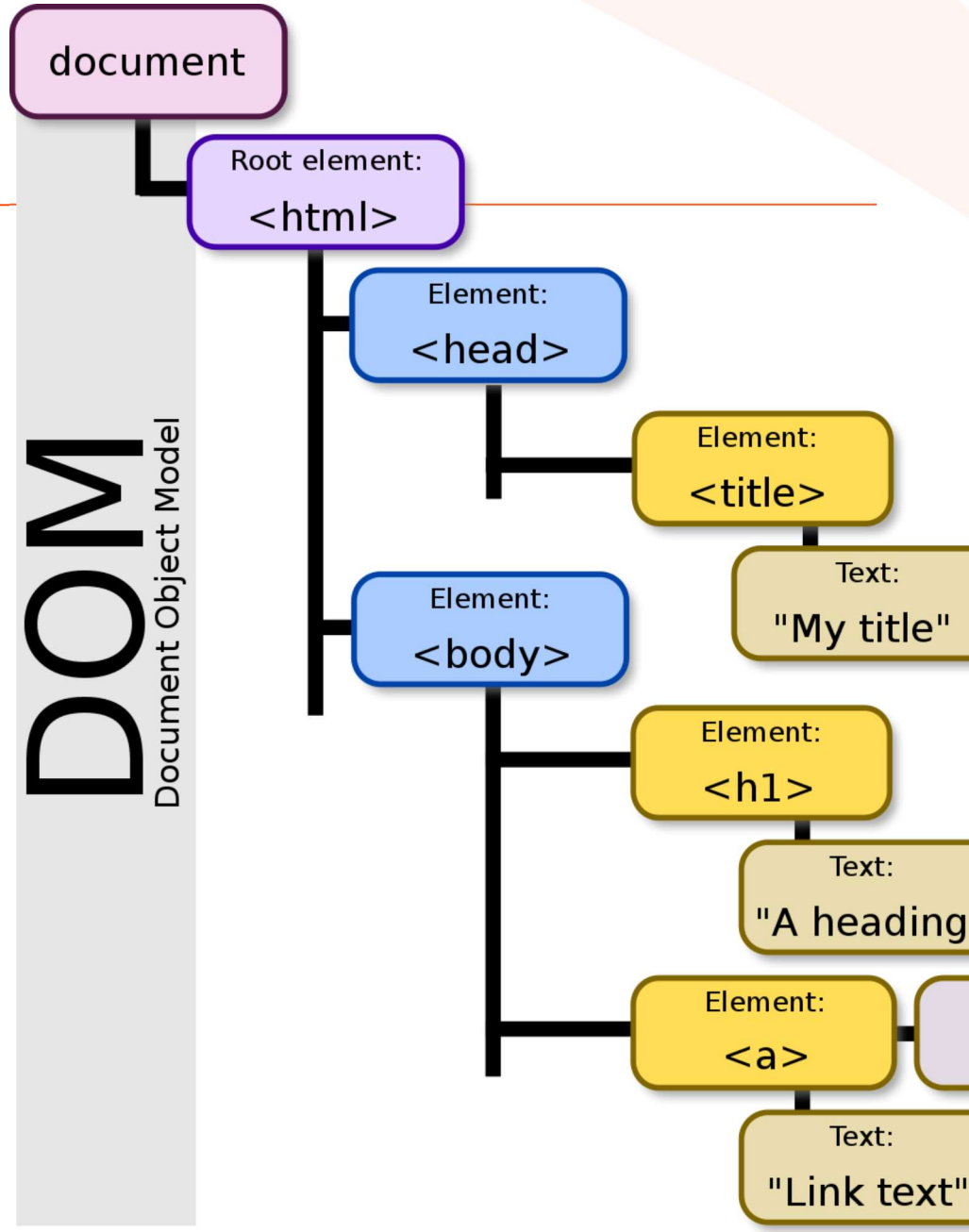
```
const someElement = document.getElementById('para1');
const someElements = document.getElementsByClassName('
```

- Trigger code when HTML events occur using (for example):

```
<button onclick="someFunction();">blue</button>
```

The DOM

Tree Structure: The DOM represents an HTML document as a tree of elements.



JavaScript Challenge

Create an HTML page which contains the following functionality:

In a game you can score 1, 2 or 3 points in each round.

- You will track the total score, and average score for each round
- Provide 3 buttons the user can click on to score each round
- Bonus: When 10 rounds have been played, hide or disable the butto

What is Typescript?

- A **superset** of JavaScript
- Cannot be executed directly in the browser or Node.js
- Can be **compiled** to JavaScript

Benefits:

- Type checking - allows us to catch possible errors / unwanted behav
  compile time
- Can use modern JS features
- Modern IDEs have (configurable) typescript support

Setting up VS Code

First install typescript with

```
npm install –g typescript
```

Consider installing the following extensions:

- **ESLint** – gives us code quality checks
- **Path Intellisense** – helps us with path imports
- **Prettier** – code formatter (with shortcut key SHIFT+ALT+F)

Setting up VS Code

Because typescript needs to be compiled we need to set up our dev environment:

Create an index.html file that loads as script file

Create a typescript file

Run this to keep compiling the TypeScript to JavaScript:

```
tsc -w myCode.ts
```

Typescript Syntax Basics – Defining Variabl

- Specify variable types and function return types with :

```
const myName: string = "Matt";
let myAge: number = 21;
```

```
const  birthday = (age: number) :
number => {
    return age + 1;
}
```

- You can chain variable types with |

```
let matt : string | number;
```

Typescript Syntax Basics – Nullable Variabl

- The IDE will check for variables that have not been assigned

```
let daveAge : number;
birthday(daveAge);
```

- It also doesn't allow variables to have the value null

```
let daveAge : number = null;
```

- To allow a variable to have the value null, define it with **| null**

```
let daveAge : number | null = null;
```

Typescript Syntax Basics – Nullable Variabl

- The IDE will not let you pass nullable variables to functions by defa
- You can:
  - Wrap them in an **if** statement

```
if (myAge != null) birthday(myAge);
```

  - Provide an alternative with **??** (nullish coalescing)

```
birthday(myAge ?? 25);
```

  - Insist that the item is not null with **!** (warning – this is dangerous!)

```
birthday(myAge!);
```

Typescript Syntax Basics – Optional Param

- You can define function parameters as optional by using **| null** or **?**

```
const birthday = (age: number | null) : number => {...
```

```
const birthday = (age?: number) : number => {...
```

- Typescript has an additional data type of **any** – but this is best avoi

Typescript - Types

- A type is a cut-down interface, that you can create JS objects from

```
type Transaction = {
    date : Date;
    amount : number;
    description : string;
}

const transaction1 : Transaction = {
   date : new Date(),
   amount : 100,
   description : "Deposit"};
```

Typescript – Intersection Types

- You can use combine types or interfaces to create new types with &

```
type Teacher = {name: string, role: string};
type Employee = {id : number, salary : number};
type SchoolEmployee = Teacher & Employee;
```

Typescript - Generics

- You can use Generics with classes

```
class AccountManager<T> {
    account : T;

    constructor(account : T) {
        this.account = account;
    }
}
const checkingAccountManager = new AccountManager<CheckingAccount>(ne
CheckingAccount(1, 100, [1,2,3]));
```

Typescript - Enums

- You can use Enums to restrict data values

```
const enum TransactionType { Deposit, Withdrawal, Transfer };
```

Typescript – Nullable Chaining

- You can use **?** before property or function names to allow null safe

```
type User = { name : string, password? : string, active : boolean}
const user : User = {name: "Matt", active: true};
console.log(user.password?.length || "no password set");
```

Typescript – Casting

- You can cast a data type or tell the compile what a type is using the keyword – this is particularly useful when working with the DOM:

```
const description = document.getElementById('description') as
HTMLInputElement;
```

Typescript Challenge

Create an HTML page which contains the following functionality:

Display a bank account with a list of transactions.

Have the ability to add transactions and maintain the balance

Spread Operator

- The spread operator will split an array into a list of its elements, or properties out of an object.
- This is useful if you want to add to an array

```
const smallArray = [1,2,3];
const largerArray = [...smallArray, 4, 5];
```

- Or change part of an object

```
let customer = {id:3, name:"Matt", totalSpend: 331.22, active: true};
customer = {...customer, totalSpend: 388.19};
```

Destructuring

- Destructuring lets you assign the values of an array to variables

```
const smallArray = [1,2,3];
const [one,two,three] = [smallArray];
```

- or to extract parts of an object as variables

```
const customer={name:"matt", age:21, active:true};
const {age, name} = customer;
```