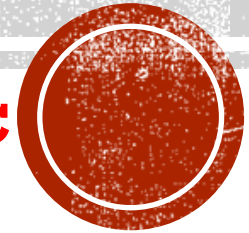
HTML MODULE 07
CREATING OBJECTS & METHODS BY USING JAVASCRIPT

Summer 2021 – Web Development using ASP .Net Core MVC



JAVASCRIPT VARIABLES

- To **declare a variable**, use the **var** keyword. We use variables to store data values.
  - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables
  - var year; //default value is **undefined** ← only two types of scope: **Global Scope** and **Function Scope**.
  - year = 2021; //**null is different than undefined**

- On your own, please read **JavaScript Identifiers** (what is allowed as variable names?)

- JavaScript has **dynamic types**, meaning variables can be used to hold different data types. The same variable can store integer, string,…
  - var carName = "Volvo";
  - carName = 2021;

- ES2015 introduced two important new JavaScript keywords: **let** and **const**. Use them when possible.
  - let year ; //default value is **undefined** ← also provide **Block Scope**
  - year = 2021;

- Some cases to consider:

```
var x = 2;

// Now x is 2

var x = 3;

// Now x is 3
```

```
let x = 2;        // Allowed
let x = 3;        // Not allowed

{
  let x = 4;    // Allowed
  let x = 5;    // Not allowed
}
```

```
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
```

```
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

- Sources:
  - https://www.w3schools.com/js/js_syntax.asp https://www.w3schools.com/js/js_datatypes.asp
  - https://www.w3schools.com/js/js_variables.asp
  - https://www.w3schools.com/js/js_let.asp See also: https://www.w3schools.com/js/js_const.asp

JAVASCRIPT HOISTING

- Check out the following examples:

```
<p id="demo"></p>

<script>
try {
  carName = "Volvo";
  var carName;
  document.getElementById("demo").innerHTML = carName;
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name + ": " + err.message;
}
</script>
```

Volvo

```
<p id="demo"></p>

<script>
try {
  carName = "Volvo";
  let carName;
  document.getElementById("demo").innerHTML = carName;
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name + ": " + err.message;
}
</script>
```

ReferenceError: Cannot access 'carName' before initialization

- Hoisting: moving declarations to the top of the current scope (either *top of script* or *top of function*).
  - This allows a variable to be declared **after** it has been used. (How does this compare to C#?)
  - "Using a **let** variable before it is declared will result in a **ReferenceError**."
  - Node: JavaScript hoists **declarations**, not **initializations**

```
<p id="demo"></p>

<script>
try {
  carName = "Volvo";
  document.getElementById("demo").innerHTML = carName;
  var carName;
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name + ": " + err.message;
}
</script>
```

Volvo

```
<p id="demo"></p>

<script>
try {

  document.getElementById("demo").innerHTML = carName;
  var carName= "Volvo";
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name + ": " + err.message;
}
</script>
```

undefined

- Source: https://www.w3schools.com/js/js_hoisting.asp

JAVASCRIPT STRICT MODE

- "Strict mode makes it easier to write "secure" JavaScript."

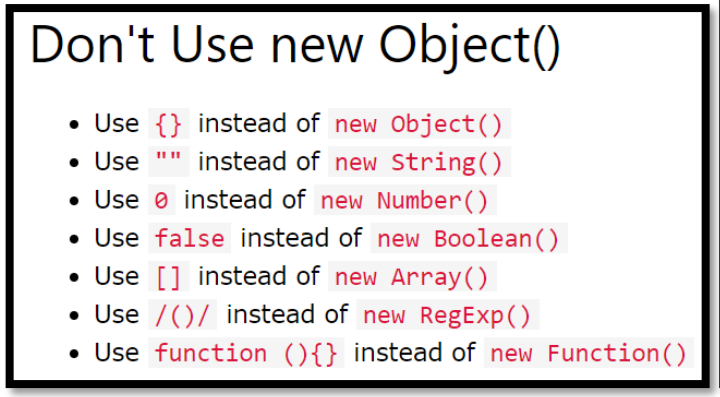
- To use **strict mode**, add **"use strict";** at the beginning of a **script** or a **function**.
  - If it is declared at the **beginning of a script**, then it has **global scope** (it affects all code in the script)
  - If it is declared **inside a function**, then it has **local scope** (it will only the code inside that function)

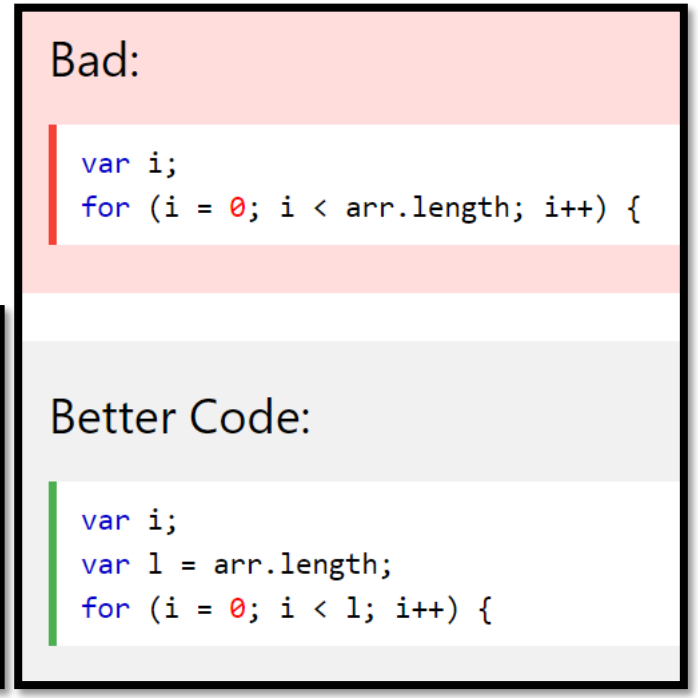
- What it does:
  - one can not use undeclared variables or objects.
    - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_variable
    - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_local
  - one can not delete a variable (or object).
    - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_delete
  - one can not have a function with a duplicate parameter.
    - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_duplicate
  - one cannot use **eval** as a variable name
    - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_eval
  - …

- Source: https://www.w3schools.com/js/js_strict.asp

JAVASCRIPT COMMON MISTAKES

- You may want to review this on your own …
  - https://www.w3schools.com/js/js_mistakes.asp
- Also, this:
  - https://www.w3schools.com/js/js_performance.asp
- And this:
  - https://www.w3schools.com/js/js_best_practices.asp





IIFE (Immediately Invoked Function Expression)

- How can you **define** a function?

```
<p id="demo"></p>

<script>

function toCelsius(fahrenheit) {
  document.getElementById("demo").innerHTML =
    "The temperature is " +(5/9) * (fahrenheit-32) + " Celsius";
}
```

- How do you **call** it?

```
toCelsius(100);
```

- Can you **define** and **call** the function in one statement?

```
<p id="demo"></p>

<script>

(function toCelsius(fahrenheit) {
  document.getElementById("demo").innerHTML =
    "The temperature is " +(5/9) * (fahrenheit-32) + " Celsius";
}) (100);

</script>
```

- Source: https://developer.mozilla.org/en-US/docs/Glossary/IIFE

JAVASCRIPT DATA TYPES

- On your own you may want to read in more depth the sources shown below.

- JavaScript has **primitive data types**:
  - string: let name="Alex"; ←one can use single or double quotes!
  - number: let pi=3.14159, y=name.length;
  - boolean: let isEven=true;
  - undefined: let x; //x is undefined

- JavaScript also has **complex data types**:
  - function: function f() {}
  - object: [1, 2, 3, 4] ← in JavaScript arrays are objects. Review **indexOf** …
    {name: 'Alex', major: "Computer Science", gpa: 2.85}

- Sources:
  - https://www.w3schools.com/js/js_datatypes.asp
  - https://www.w3schools.com/js/js_strings.asp
  - https://www.w3schools.com/js/js_string_methods.asp
  - https://www.w3schools.com/js/js_numbers.asp
  - https://www.w3schools.com/js/js_number_methods.asp
  - https://www.w3schools.com/js/js_arrays.asp
  - https://www.w3schools.com/js/js_array_methods.asp

JAVASCRIPT OBJECTS

- **Objects** are variables that can contain multiple values.
  - objects are **containers** for **named values** (called **properties**) and **methods**.
  - "A **method** is a **function** stored as a **property**."

- Example:
  - let student1 = {name:"Alex", major:"Art", gpa:3.56, print: **function**() {return this.name + " " + this.major;} };
  - Above, **this** refers to the "owner" of the function.

- To access those values/methods one can use either syntax shown below:
  - student1.major
  - student1["major"]

  - student1.print()

- Source: https://www.w3schools.com/js/js_objects.asp

JAVASCRIPT OBJECTS

- To ways to create an **empty object**:
  - **let** myCar = **new Object();** //not recommended … see https://www.w3schools.com/js/js_best_practices.asp
    - A more thorough discussion here: https://stackoverflow.com/questions/383402/is-javascripts-new-keyword-considered-harmful
  - **let** myCar = **{}**

- Let's add a few **properties.** What properties should we add?
  - For example: **myCar.miles = 12345;**

- Let's add a few **methods.** What methods should we add? Include **drive**().
  - For example: **myCar.drive = function(num_miles){**
    if(num_miles > 0)
    miles += num_miles;
    }

- **Important Note:** JavaScript does not support overloaded functions. (see source)
  - Defining multiple methods with the same name will overwrite/replace existing methods.
- Source: https://www.w3schools.com/js/js_objects.asp

JAVASCRIPT OBJECT CONSTRUCTOR FUNCTIONS

- **Object constructor functions** can be used (as a "blueprint") to create new objects
  - Use the **this** keyword to refer to the object being created
  - Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor1

- Example:

```
function Car(mileage, make, model, year) {
  this.mileage = mileage;
  this.make = make;
  this.model = model;
  this.year= year;
  this.version= 1.001; //what does this do???
  this.drive = function(num_miles){ if(num_miles > 0) miles += num_miles;} ;  //what about this???
}
```

- To create instances of such objects:

```
let mySportsCar = new Car(200, "Ford", "Mustang", 2021);
let myWorkCar = new Car(12000, "Toyota", "Camry", 2016);
mySportsCar.version …
mySportsCar.drive(10) …
```

- Source: https://www.w3schools.com/js/js_object_constructors.asp

JAVASCRIPT OBJECT CONSTRUCTOR FUNCTIONS

- How can you add another property (or method) to an existing constructor?

- Previous example:

```
function Car(mileage, make, model, year) {
  this.mileage = mileage;
  this.make = make;
  this.model = model;
  this.year= year;
}
```

- Wrong way (also see this example) – you cannot add this way to an existing constructor!
  - Car.color = "golden";

- Correct way: use the **prototype** property to add new properties and methods to **object constructors**
  - Car.**prototype.**color = "golden";

- Source: https://www.w3schools.com/js/js_object_prototypes.asp

JAVASCRIPT CLASSES

- A JavaScript **class** is a **template** for creating JavaScript objects.
  - Introduced in ECMAScript 2015 (ES6)
  - A **class** is a blueprint for creating objects, it is not an object.

- When a new object is created, the **constructor method** is automatically called.
  - The **constructor** is a special method for creating and initializing objects created within a class
  - An **empty constructor** is added automatically if you don't include a constructor definition in your class.
  - A class cannot have more than one **constructor** method

```
<p id="demo"></p>

<script>
  class Student {
    constructor(name, major, gpa) {
      this.name = name;
      this.major = major;
      this.gpa = gpa;
    }

    print()
    {
      return  this.name + " " + this.major;
    }
  }

  student1 = new Student("Alex", "CS", 3.56);

  document.getElementById("demo").innerHTML = student1.print();
</script>
```

- Example

- Sources:
  - https://www.w3schools.com/js/js_classes.asp
  - https://www.w3schools.com/jsref/jsref_classes.asp

JAVASCRIPT CLASS INHERITANCE

- What does **(class) inheritance** mean?

- To create a **class inheritance**, use the **extends** keyword.
  - Use the **super()** method in the constructor method to call the parent's constructor method
  - Use the **super.methodName()** to call the parent's method

- Example

- Our own example:
  - Create a class **User** (with a display method and a constructor),
  - Create a class **Student** that extends **User**, … make use of super …
  - Example: https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_class_extends

- Sources:
  - https://www.w3schools.com/js/js_class_inheritance.asp
  - https://www.w3schools.com/jsref/jsref_class_super.asp
  - https://www.w3schools.com/jsref/jsref_class_extends.asp

JAVASCRIPT STATIC METHOD

- Static methods are methods that are **called directly on the class**
  - There is no need to create an instance/object of that class.
- Example: https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_class_static2

- Source: https://www.w3schools.com/js/js_class_static.asp

JAVASCRIPT GETTERS AND SETTERS

- Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_classes_getters2

- In C# what is a getter? What is a setter?

- Let's see how we can a create getter/setter in JS.

```
<p id="demo"></p>

<script>
class Student {
  constructor(name, major) {
    this._name = name;
    this._major = major;
  }
  get name() {
    return this._name;
  }
  set name(x) {
    this._name = x;
  }

  get major() {
    return this._major;
  }
  set major(x) {
    this._major = x;
  }
}

let st1 = new Student("Alex", "Computer Science");

document.getElementById("demo").innerHTML = st1.name + " : " + st1.major;
</script>
```

- Source: https://www.w3schools.com/js/js_class_inheritance.asp

FOR YOUR LABS...

- Use the **import** & **export** statements to share code across multiple files.

- The **exports** keyword makes **properties** and **methods** available outside the file.
  - Only **exported** items are available to be reused elsewhere.

```
import { ScheduleItem } from "./ScheduleItem.js";

export class ScheduleList {
        //TODO: Add Constructor
}

//TODO: Add methods
```

The act of exporting one or multiple variables is called a named export:

```
const firstName = 'Robin';
const lastName = 'Wieruch';

export { firstName, lastName };
```

And import them in another file with a relative path to the first file.

```
import { firstName, lastName } from './file1.js';

console.log(firstName);
// Robin
```

```
import * as person from './file1.js';

console.log(person.firstName);
// Robin
```

Imports can have aliases, which are necessary when we import functionalities from multiple files that have the same named export.

```
import { firstName as username } from './file1.js';

console.log(username);
// Robin
```

- See also:
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
  - https://developer.mozilla.org/en-US/docs/web/javascript/reference/statements/export

- Sources:
  - https://www.w3schools.com/nodejs/nodejs_modules.asp
  - https://www.robinwieruch.de/javascript-import-export
  - https://github.com/MicrosoftLearning/20480-Programming-in-HTML5-with-JavaScript-and-CSS3/blob/master/Instructions/20480C_MOD07_LAK.md

IN-CLASS DEMO

Demonstration: Refining Code for Maintainability and Extensibility

- Source/Steps
- https://github.com/MicrosoftLearning/20480-Programming-in-HTML5-with-JavaScript-and-CSS3/blob/master/Instructions/20480C_MOD07_DEMO.md

LAB / HOMEWORK

- **Module 07**
  - Exercise 1: Refactoring JavaScript Code to Use Classes and Objects

- You will find the **high-level** steps on the following page:
  https://github.com/MicrosoftLearning/20480-Programming-in-HTML5-with-JavaScript-and-CSS3/blob/master/Instructions/20480C_MOD07_LAB_MANUAL.md
- You will find the **detailed** steps on the following page:
  https://github.com/MicrosoftLearning/20480-Programming-in-HTML5-with-JavaScript-and-CSS3/blob/master/Instructions/20480C_MOD07_LAK.md

- For your homework submit one zipped folder with your complete solution.