Advanced Programming 2
Recitation 8 – Web Applications Part II

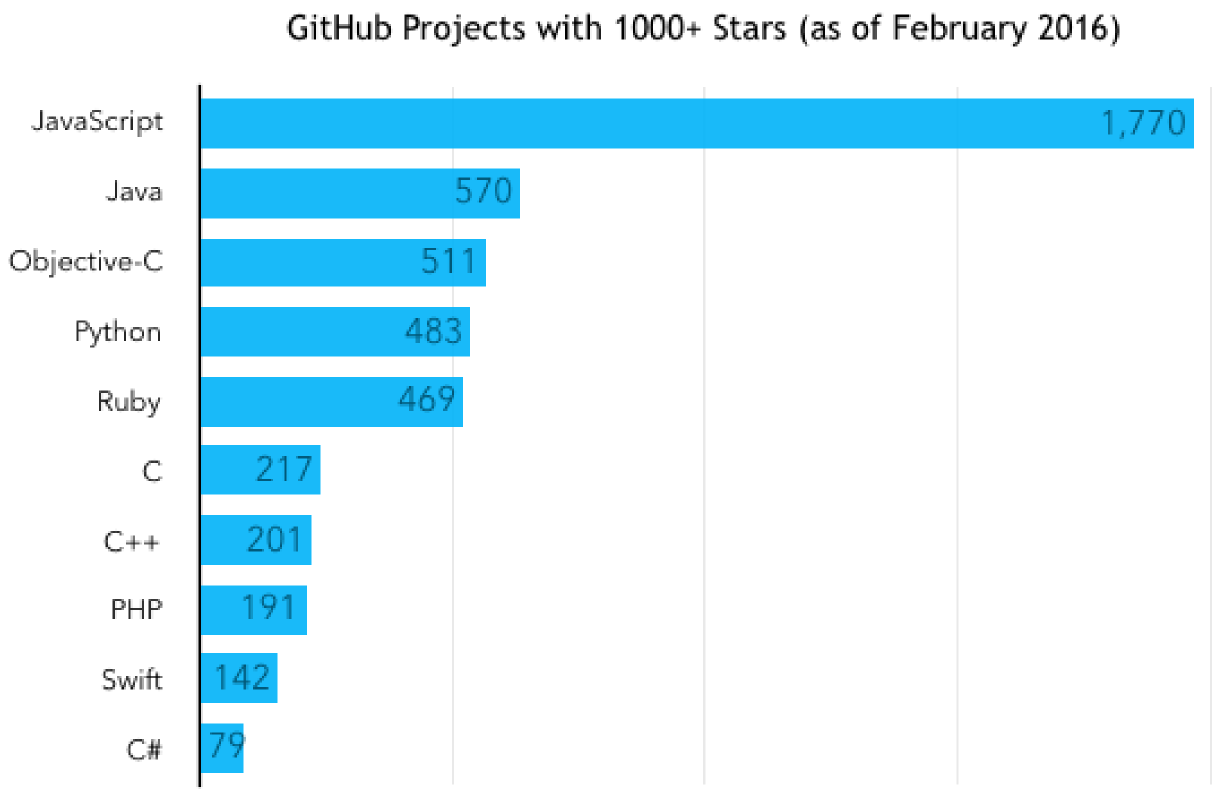
Roi Yehoshua

2017

JavaScript

JavaScript

- **The** programming language of the future?



JavaScript

- Created In 1995 by Brendan Eich as a new language for Netscape Navigator
- Standardized as ECMAScript in 1997
  - Latest version is ECMAScript 7
- Originally used to enhance client side development in web applications
- Today used for many other purposes:
  - HTML5 mobile apps
  - Server side development (NodeJS)
  - JS on devices – the internet of things
    - Huge potential of running JavaScript on embedded devices

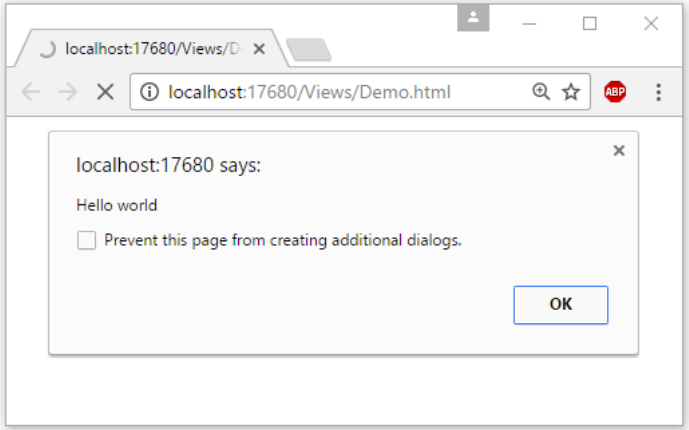
Language Main Features

- Interpreter based (no compilation) scripting language
- Loosely typed and dynamic language
- Uses syntax influenced by that of Java
  - However, has very different semantics than Java
- Main components
  - The Core (ECMAScript)
  - The DOM (Document Object Model)
  - The BOM (Browser Object Model)

Adding Script to HTML - Embedding

- You place javascript on a page using the <script> tag
  - From HTML5, browsers assume type="text/javascript" if not stated
- The script can be placed in either the head or body section
  - It is executed as soon as the browser renders the script block
  - Current practice often places it just before the closing body tag

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
<meta charset="utf-8" />
</head>
<body>
    <script>
        alert("Hello world");
    </script>
</body>
</html>
```



localhost:17680 says:

Hello world

□ Prevent this page from creating additional dialogs.

OK

Adding Script to HTML - Linking

- You can also place javascript in a separate file and link to it
  - Useful script is going to be used on multiple pages
  - Requires an additional request to the server
  - The requested file is cached by the browser
  - Preferred approach to working with script
- When linking to external script there are a few things to remember:
  - There must be a closing </script> tag
  - No javascript can occur within the script tag

MyPage.html

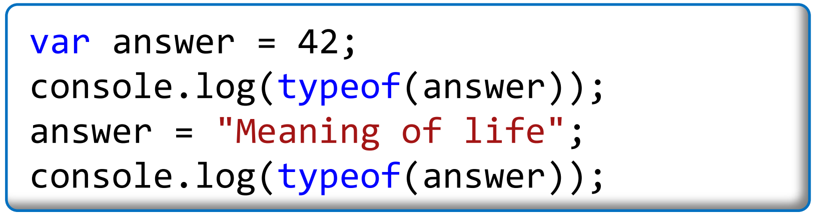
```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
</head>
<body>
    <script src="MyScript.js"></script>
</body>
</html>
```

MyScript.js

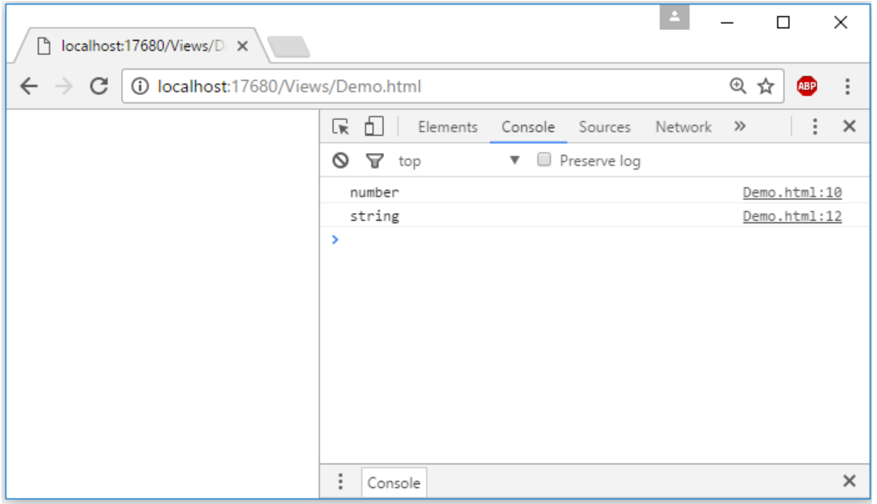
```
alert("Hello world");
```

Variables

- We use var to declare a variable
  - Best practice is to use camelCase for variable names
- You don't specify the data type of a variable when you declare it
- The same variable can point to different data types
  - You use the keyword typeof to read its runtime type
- A variable has a scope
  - Global variables should be avoided (like in any other object oriented language)



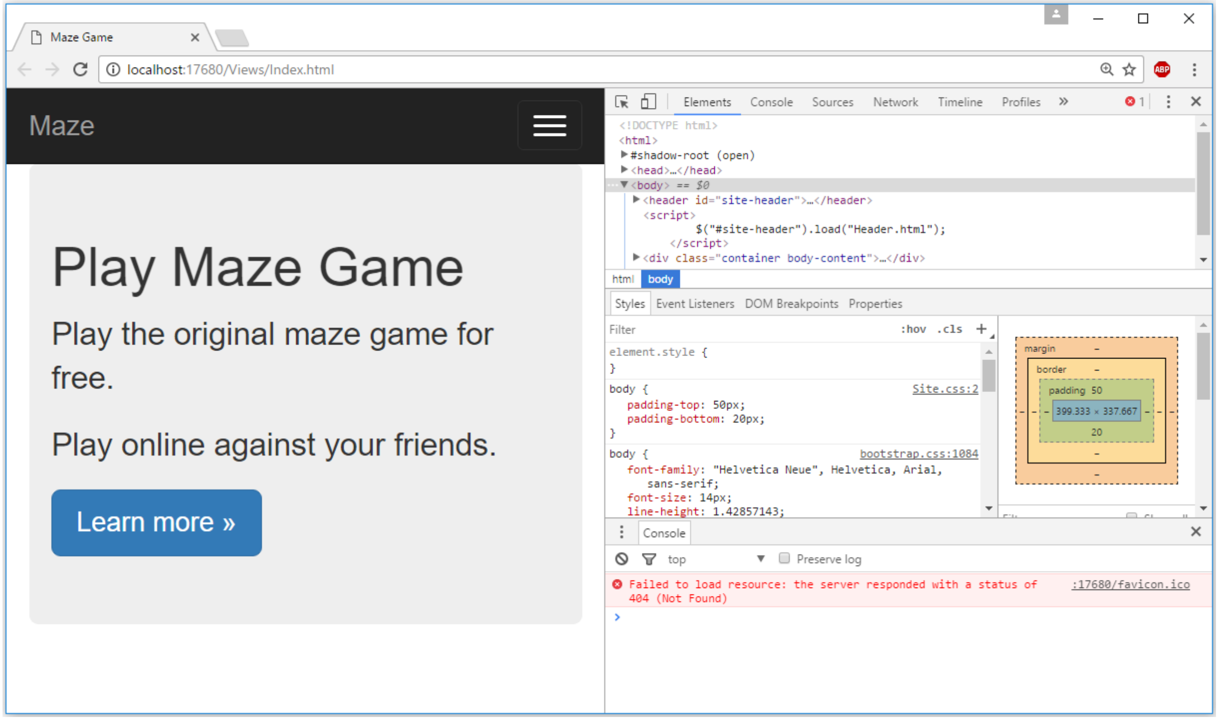
```
var answer = 42;
console.log(typeof(answer));
answer = "Meaning of life";
console.log(typeof(answer));
```



```
number                                    Demo.html:10
string                                    Demo.html:12
>
```

Chrome Developer Tools

- Incredibly useful for testing and debugging
- To display press F12 or Ctrl-Shift-I
- Console panel shows debugging/error messages



Play Maze Game

Play the original maze game for free.

Play online against your friends.

Learn more »

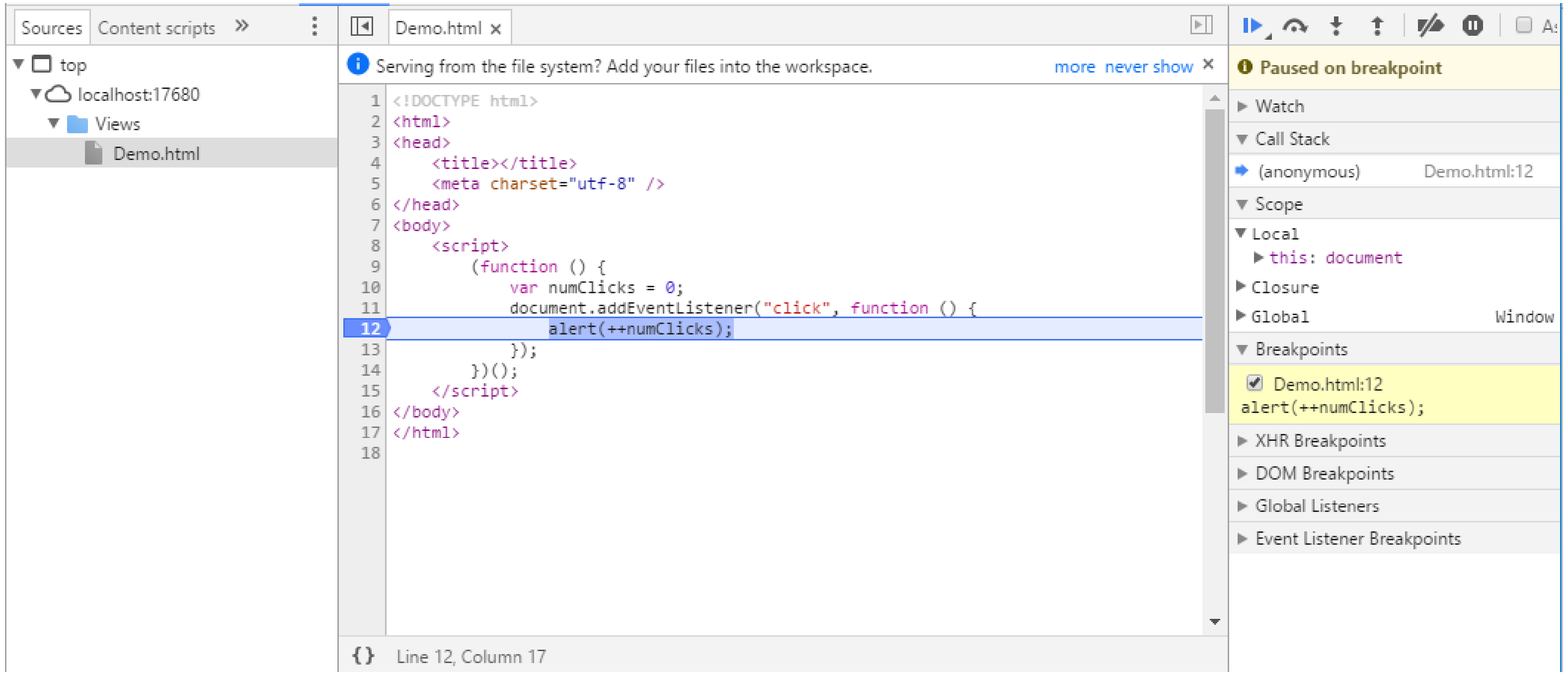
```
<!DOCTYPE html>
<html>
 ▶#shadow-root (open)
 ▶<head>…</head>
 ▼<body> == $0
   ▶<header id="site-header">…</header>
    <script>
            $("#site-header").load("Header.html");
        </script>
   ▶<div class="container body-content">…</div>
```

```
element.style {
}
body {                                  Site.css:2
   padding-top: 50px;
   padding-bottom: 20px;
}
body {                            bootstrap.css:1084
   font-family: "Helvetica Neue", Helvetica, Arial,
      sans-serif;
   font-size: 14px;
   line-height: 1.42857143;
```

```
Failed to load resource: the server responded with a status of     :17680/favicon.ico
404 (Not Found)
>
```

Debugging JS in Google Chrome

- Chrome Developer Tools contain a JavaScript Console
  - Opened via the Sources Tab
  - Can set breakpoints
  - Step Over – F10, Step Into – F11, Step Out – Shift+F11



Serving from the file system? Add your files into the workspace.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
</head>
<body>
    <script>
        (function () {
            var numClicks = 0;
            document.addEventListener("click", function () {
                alert(++numClicks);
            });
        })();
    </script>
</body>
</html>
```

Paused on breakpoint

☑ Demo.html:12
alert(++numClicks);

Line 12, Column 17

JavaScript Core Types

- Primitive data types:
  - Numbers
  - Strings
    - can be expressed using " or '
  - Booleans (true/false)
  - Undefined
    - indicates an uninitialized variable
- Reference data types:
  - Object
  - Function
- A reference is implemented as a pointer
  - Points to an object that resides inside the heap

```
console.log(typeof 1); // number
console.log(typeof 1.2); // number
console.log(typeof "abc"); // string
console.log(typeof "abc"[0]); // string
console.log(typeof true); // boolean
console.log(typeof function () {}); // function
console.log(typeof {}); // object
console.log(typeof null); // object
console.log(typeof new Date()); // object
console.log(typeof window); // object
console.log(typeof undefined); // undefined
console.log(typeof blabla); // undefined
```

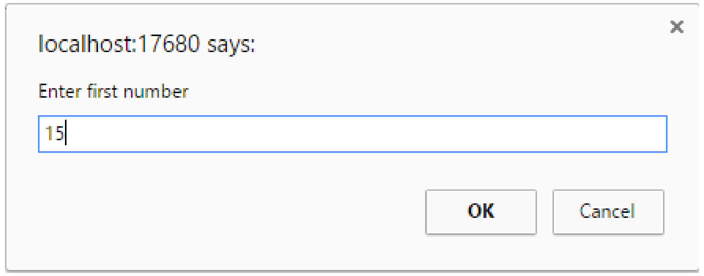
Data Type Conversion

- Data types are converted automatically as needed during script execution
- parseInt() can be used to parse a string into an integer number
- parseFloat() can be used to parse a string into a floating-point number
- In case of failure NaN is returned
- e.g., a program that gets two numbers from the user and prints their sum:

```
var num1, num2;
do {
    num1 = parseInt(prompt("Enter first number"));
} while (isNaN(num1));

do {
    num2 = parseInt(prompt("Enter second number"));
} while (isNaN(num2));

var sum = num1 + num2;
alert("sum is: " + sum);
```



localhost:17680 says:

Enter first number

15

Comparison Operators

- JavaScript has both abstract and strict comparisons
- Abstract comparison (==) converts the operands to the same type before making the comparison
- A strict comparison (===) is only true if the operands are the same type

```
console.log(3 == "3"); // true
console.log(3 === "3"); // false

console.log(true == "1"); // true (boolean true and the
string '1' are converted to 1)
console.log(true === "1"); // false

console.log(undefined == null); // true
console.log(undefined === null); // false (undefined and
null are distinct types)
```

Arrays

- An array is created using the following syntax:
  - []

```
var arr = [];
var arr = [1, 2, 3];
```

  - new Array()

```
var arr = new Array();       // an empty array
var arr = new Array(10);     // length is 10
var arr = new Array(10, 2); // length is 2
```

- You can have variables of different types in the same array

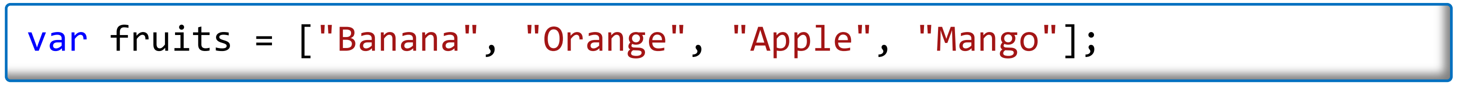
```
arr[0] = new Date();
arr[1] = 5;
arr[2] = new Array("Saab", "Volvo", "BMW");
```

- You refer to an array element by referring to the index number

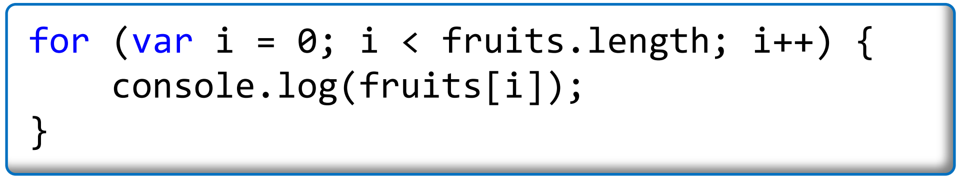
```
var num = arr[0];
arr[0] = 3;
```

Iterating an Array

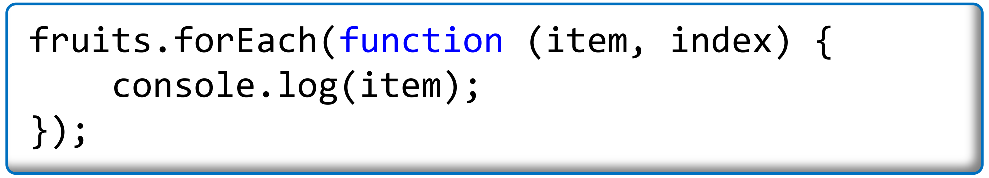
- Two ways to iterate an array
  - Using a for loop
    - Use a running index and the length property
  - Using a for each loop
    - The forEach() method calls a provided function once for each element in an array, in order.



```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```



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



```
fruits.forEach(function (item, index) {
    console.log(item);
});
```

Array is dynamic

- New elements can be added/deleted at runtime
  - The unassigned parts of an array are undefined
- The property length is automatically being updated

```
var arr = [1, 2, 3, 4];
arr.push(10);     // add last
arr.pop();        // remove last
arr.splice(1, 1); // remove second element
alert(arr);       // prints 1,3,4
arr[10] = 10;     // never throws exception (array resizes)
arr.length = 5;   // resize
arr.shift();      // remove first
arr.splice(0, 1); // remove first
alert(arr);       // prints 4,,
```

Functions

- Declaring a function:

```
function add(num1, num2) {
    return num1 + num2;
}
```

- Calling a function is also straightforward:

```
var result = add(5, 7);
```

- JS only supports “pass by value” mechanism
  - The parameter being sent to a function is copied
    - Whether it is a reference or a value
- All parameters are optional
  - Parameters of functions default to undefined
  - Thus, overloading is not supported
    - But can be simulated

```
function add(num1, num2, num3) {
    num3 = num3 || 0;
    return num1 + num2 + num3;
}
```

Anonymous Functions

- An anonymous function is a function that was declared without any named identifier to refer to it
- The most common use for anonymous functions are as arguments to other functions
  - e.g., the method window.setInterval() calls a function at specified intervals (in milliseconds)

```
var func = function () {
    alert('I am anonymous');
};
func();
```

```
setTimeout(function () {
    alert('hello');
}, 1000);
```

Closures

- A closure is an inner function that refers to the outer (enclosing) function’s variables
- These functions 'remember' the environment in which they were created
  - i.e., they have access to the outer variables even after the outer function returns

```
function makeFunc() {
    var name = "Roi"; // a local variable created by makeFunc
    function displayName() { // displayName() is the inner function, a closure
        alert(name); // use variable declared in the parent function
    }
    return displayName;
}

var myFunc = makeFunc(); // myFunc is a reference to the instance of the function
displayName created when makeFunc is run
myFunc(); // when myFunc is invoked, the variable name remains available for use
```

Self Executing Function

- Create, execute and discard a function in one block
- Used to create a private scope and prevent global namespace resolution

```
(function () {
    //  External code has no access to these variables
    var url = "http://www.google.com";
    var productKey = "ABC";
})();
```

- Only inner functions (closures) inside the self-executing function can access the variables declared in this function

```
(function () {
    var numClicks = 0;
    document.addEventListener("click", function () {
        alert(++numClicks);
    });
})();
```

Handling Errors – Try, Catch and Finally

A try catch finally statement

```
try
{
  var x = parseInt(prompt("Enter a number number", ""));
  if (isNaN(x)){
    var e = new Error();
    e.message = "That wasn't a number";
    throw e;
  }
}
catch (e)
{
  alert("Something went wrong:" + e.message);
}
finally
{
  . . .
}
```

Try to execute the code

Create a new error object

Create the thrown error

Finally executes on success or failure

Check the input

Throw the error

Handle the error

JS Object

- A container of keys and values
- The key must be of type string
- An object can be initialized at declaration
- Two ways to create an object:

Using the new operator

```
var obj = new Object();
obj.id = 123;
obj.name = "Roi Yehoshua";
obj.email = "roiyeho@gmail.com";
```

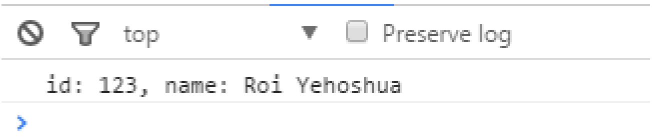
object literal syntax

```
var obj = {
    id: 123,
    name: "Roi Yehoshua",
    email: "roiyeho@gmail.com"
};
```

Function inside an Object

- An object can contain functions (methods)
- Typically these methods are defined using an anonymous function declaration
- The keyword this is used for accessing the object's properties

```
var obj = {
    id: 123,
    name: "Roi Yehoshua",
    email: "roiyeho@gmail.com",
    print: function() {
        console.log("id: " + this.id +
", name: " + this.name);
    }
};
obj.print();
```



```
id: 123, name: Roi Yehoshua
>
```

Modules

- Arrange your JavaScript code into modules
- Each module is surrounded with self-executing function thus hiding all local variables and functions
- Peek the ones that should be public (sparsely)

```
var counter = (function () {
    // Keep this variable private inside this closure
scope
    var privateCounter = 0;
    function increment() {
        privateCounter++;
    }
    function decrement() {
        privateCounter--;
    }
    function value() {
        return privateCounter;
    }
    // Explicitly reveal public pointers to the private
    // functions that we want to reveal publicly
    return {
        increment: increment,
        decrement: decrement,
        value: value
    };
})();

console.log(counter.value()); // logs 0
counter.increment();
counter.increment();
console.log(counter.value()); // logs 2
```

Classes (ECMAScript2015)

- You use the **class** keyword to declare a class
- The data members of the class are defined in the constructor
- Getters and setters behave like C# properties
- The **extends** clause lets you create a subclass of an existing class

```
class Rectangle {
    constructor(width, height) {
        this._width = width;
        this._height = height;
    }
    get width() {
        return this._width;
    }
    get height() {
        return this._height;
    }
    calcArea() {
        return this._width * this._height;
    }
}

const square = new Rectangle(10, 10);
console.log("width: " + square.width + ", height: "
+ square.height);
console.log(square.calcArea());
```