More on Javascript

Declaration using var, let, const

- Re-declare
- Access before declare
- Global Variable scope
- Block or function scope (later in the slides)

Declaration using var

EXAMPLE

```
var x=5;
console.log(x);
```

OUTPUT

```
5
```

Variables are containers for storing data (values).

Re-declaration using var

EXAMPLE

```
var a=1;  //declaration
var a=2;  //redeclaration
console.log(a);
```

OUTPUT

```
2
```

Using the keyword var you can declare and redeclare..

Redeclaration using let and const

EXAMPLE

```
let a=1;  //declaration
let a=2;  //redeclaration ERROR
console.log(a);
```

OUTPUT

```
Error
```

Using the keyword let or const you are NOT ALLOWED to redeclare..

Accessing variables before declaration

EXAMPLE

```
console.log(x);
var x=5;
console.log(x);
```

OUTPUT

```
Undefined
5
```

The undefined output is not clear, especially when you are dealing with a complex program

Accessing variables before declaration using let and const

EXAMPLE

```
console.log(a);  //uncaught ReferenceError:
let a=1;
```

OUTPUT

```
//uncaught ReferenceError
```

When declaring a variable using let or const, you cannot access a before declaration

Variable scope drama

EXAMPLE

```
var abc=1;
console.log(window.abc); // will output 1
```

OUTPUT

```
1
```

When declaring a variable using let or const, you cannot access a before declaration

Global Variable scope

EXAMPLE

```
let abc=1;
console.log(window.abc); // will output 1
```

OUTPUT

```
undefined
```

When declaring a variable using let or const, it is not placed in the global scope under the window object

Template literals

- **Template Literals** use back-ticks (``) rather than the quotes ("") to define a string:

```
let text = `Hello World!`;
```

- With **template literals**, you can use both single and double quotes inside a string:

```
let text = `He's often called "Johnny"`;
```

- **Template literals** allows multiline strings:

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Template literals - interpolation

EXAMPLE

```
let a = “We love”;
let b =“Javascript”;
let c =“and”;
let d=“programming”;
console.log(a+ ”  ” +b+  ”\n”+c+” “+d);  //old way

console.log(a+” \”\” “+b+”\n”+c+” “+d);  //old way
console.log(`${a} ${b}
${c} ${d}` );
```

Use `${…}`

Template literals - Expression substitution

EXAMPLE

```
…..
let price = 10;
let VAT = 0.25;
let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;

document.getElementById("demo").innerHTML = total;
….
```

The toFixed() method formats a number using fixed-point notation. toFixed() returns a string representation of numObj that has exactly digits digits after the decimal place

Template literals - HTML templates

EXAMPLE

```
Let markup=`
        <div class=“card”>
                <div class=“child”>
                        <h2>Title</h2>
                        <p> paragraph</p>
                             </div>
                 </div>
                `;

document.write(markup);
```

Template literals are literals delimited with backticks (`), allowing embedded expressions called substitutions.

```
`;
```

Conditional using the ternary operator

Syntax

```
Condition ? If True : If False
```

EXAMPLE

```
Let name=“John”;
Let gender=“male”;
Let age=20;
If (gender==“male”){
console.log(“mr”);
}else{
console.log(“”Mrs”);}
```

```
`;
```

```
Let name=“John”;
Let gender=“male”;
Let age=20;
Gender==“male” ? console.log(“Mr”):console.log(“female”)
```

```
Let name=“John”;
Let gender=“male”;
Let age=20;
Let result=Gender==“male” ?“Mr” : “Mrs”
document.write(result);
console.log(Gender==“male”? “Mr” : “Mrs”);
console.log(`hello ${Gender==“male”? “Mr” : “Mrs”}
${name}`));
```

Chained ternary operator

Syntax

```
condition1
          ? statement
          : condition2
          ? statement
          : condition3
          ? statement
          : statement;
```

Example

```
theAge < 20
  ? console.log(20)
  : theAge > 20 && theAge < 60
  ? console.log("20 To 60")
  : theAge > 60
  ? console.log("Larger Than 60")
  : console.log("Unknown");
```

Additional Array methods (slice)

- The slice() method a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array.
- The original array will not be modified.

Example

```
let friends=[“”,””,””,””,””,””];
Console.log(friends);
Console.log(friends.slice());
Console.log(friends.slice(1));
Console.log(friends.slice(1,3));
Console.log(friends.slice(-3));
Console.log(friends.slice(1,-2));
Console.log(friends.slice(-4,-2));
Console.log(friends);
```

Additional Array methods (splice)

- The splice() method adds or removes (start deletion or insertion index, deletecount, add elements);
- It overwrites the original array.

Syntax

array.splice(*index*, *howmanytodelete(optional)*, *item1(optional)*, *.....*, *itemX(optional)*)

Example

```
myCourses.splice(1, 2, "Art", "Sports");
console.log(myCourses);
myCourses.splice(0,1,"Art", "Sports"); //deletes then adds
myCourses.splice(0,2,"Art", "Sports"); //deletes then adds
```

Additional Array methods (concat)

- The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

Syntax

```
array1.concat(array2, array3, ..., arrayX)
```

Example

```
Let myfrielnds=["Ahmed", "Sayed", "Ali", "Osama“]
Let mynewfriends=…
Let schoolFriends=..
Let allFriends=myfriends.conact(myNewFriend);

Let allFriends=myfriends.conact(myNewFriend,schholfriends);
Let allFriends=myfriends.conact(myNewFriend,schholfriends, “sara”);
```

Additional array methods (join)

- The join() method creates and returns a **new string** by concatenating all of the elements in an array, separated by commas or a specified separator string.
- If the array has only one item, then that item will be returned without using the separator.

Example

```
….
Console.log(allFriends.join());  //returns a string separated by comma
…..
```

Functions rest parameters

- When you don’t know the number of arguments (Ex: skills)
- Using rest parameters, you can allow the function to receive an unknown number of parameters

```
Let result=0;
function calc(…numbers){   /*numbers is an array of arguments */
For (let i=0; i<=numbers.length;i++){
Result+=numbers[i]
}
return `final result is ${result}`;

Console.log(calc(10,33,44,55,33));
```

Anonymous Function

- Anonymous Function is **a function that does not have any name associated with it**. Normally we use the function keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the function keyword without the function name.

Syntax

```
function() {
    // Function Body
 }
```

Nested functions

- A nested function is **a function which is defined within another function, the enclosing function**.

EXAMPLE

```
function sayMessage(fName, lName) {
  let message = `Hello`;
  // Nested Function
  function concatMsg() {
    message = `${message} ${fName} ${lName}`;
  }
  concatMsg();
  return message;
}

console.log(sayMessage("Sara", "Tedmori"));
```

Global Scope vs Local Scope

EXAMPLE

```
var a=1;
var b=2;
Function add(){


}
```

```
var a=1;
var b=2;
Function add(){
var c=2;
var d=3;
}
```

When decalared outside a function, a and b are both global variables that can be accessed from anywhere!

When decalared inside a function, c and d are both local variables that can be accessed only from inside the function.

Block scope vs function scope

EXAMPLE

```
var x = 10;

if (10 === 10) {
  var x = 50;

  console.log(`From If Block ${x}`);
}

console.log(`From Global ${x}`);
```

```
var x = 10;

if (10 === 10) {
  let x = 50;

  console.log(`From If Block ${x}`);
}

console.log(`From Global ${x}`);
```

OUTPUT

```
From if Block 50
From Global 50
```

```
From if Block 50
From Global 10
```

EXAMPLE

```
function run() {
  var foo = "Foo";
  let bar = "Bar";

  console.log(foo, bar); // Foo Bar

  {
    var moo = "Mooo"
    let baz = "Bazz";
    console.log(moo, baz); // Mooo Bazz
  }

  console.log(moo); // Mooo
  console.log(baz); // ReferenceError
}

run();
```

EXAMPLE

```
// i IS NOT known here
// j IS NOT known here
// k IS known here, but undefined
// l IS NOT known here
function loop(arr) {

    // i IS known here, but undefined
    // j IS NOT known here
    // k IS known here, but has a value only the
         second time loop is called
    // l IS NOT known here

    for( var i = 0; i < arr.length; i++ ) {
        // i IS known here, and has a value
        // j IS NOT known here
        // k IS known here, but has a value only
          the second time loop is called
        // l IS NOT known here
    };

    // i IS known here, and has a value
    // j IS NOT known here
    // k IS known here, but has a value only the
         second time loop is called
    // l IS NOT known here
    for( let j = 0; j < arr.length; j++ ) {
        // i IS known here, and has a value
        // j IS known here, and has a value
        // k IS known here, but has a value only
           the second time loop is called
        // l IS NOT known here
    };
    // i IS known here, and has a value
    // j IS NOT known here
    // k IS known here, but has a value only the
         second time loop is called
    // l IS NOT known here
}

loop([1,2,3,4]);

for( var k = 0; k < arr.length; k++ ) {
    // i IS NOT known here
    // j IS NOT known here
    // k IS known here, and has a value
    // l IS NOT known here
};

for( let l = 0; l < arr.length; l++ ) {
    // i IS NOT known here
    // j IS NOT known here
    // k IS known here, and has a value
    // l IS known here, and has a value
};

loop([1,2,3,4]);

// i IS NOT known here
// j IS NOT known here
// k IS known here, and has a value
// l IS NOT known here
```

Lexical Scope

```
function parent() {
  let a = 10;
  function child() {
    console.log(a);
    console.log(`From Child ${b}`); //uncaught reference error..
    function grand() {
      let b = 100;
      console.log(`From Grand ${a}`);
      console.log(`From Grand ${b}`);
    }
    grand();
  }
  child();
}
parent();
```