Client-side Technologies

Eng. Niveen Nasr El-Den
iTi

Day 4

Basics of JavaScript

JavaScript Functions

- A function is an organized block of reusable code (a set of statements) that handles and performs actions generated by user events
- Functions categorized into
  - built-in functions improve your program's efficiency and readability.
  - user defined functions , created by developer to make your programs scalable.
- Function executes when it is called.
  - from another function
  - from a user event, called by an event or
  - from a separate <script> block.

User-defined functions

- Function blocks begin with the keyword function followed by the function name and () then {} its building block declaration.

- Syntax:

```
function functionName(argument1, argument2, ...) {
    //statement(s) here
    //return statement;
}
```

- return can be used anywhere in the function to stop the function's work. Its better placed at the end.

Example!

User-defined functions

```
function doSomething(x) {
    //statements
}


doSomething("hello")
```

Function parameters

Function call

```
function sayHi() {
   //statements
   return "hi";
}

z = sayHi()
```

The value of z is "hi"

User-defined functions

- Function can be called before its declaration block.
- Functions are not required to return a value
  - Functions will return undefined implicitly if it is to set explicitly
- When calling function it is not obligatory to specify all of its arguments
  - The function has access to all of its passed parameters via arguments collection
  - We can specify default value using || operator or ES6 default function parameters

Example!

User-defined functions with default value

```
function dosomething (x) {
     x = x || “nothing was sent“;
    //x = x ?x: “nothing was sent”;
    //x = ( typeof x == “number”) ? x : 10;

     console.log (“value is :” + x);
}
```

```
dosomething(“hello”)
// value is : hello

dosomething()
// value is : nothing was sent

dosomething(0)
// value is : 0
```

ES6 Default value

```
function doSomething (x = “nothing was sent“) {
        console.log (“value is :” + x);
}
```

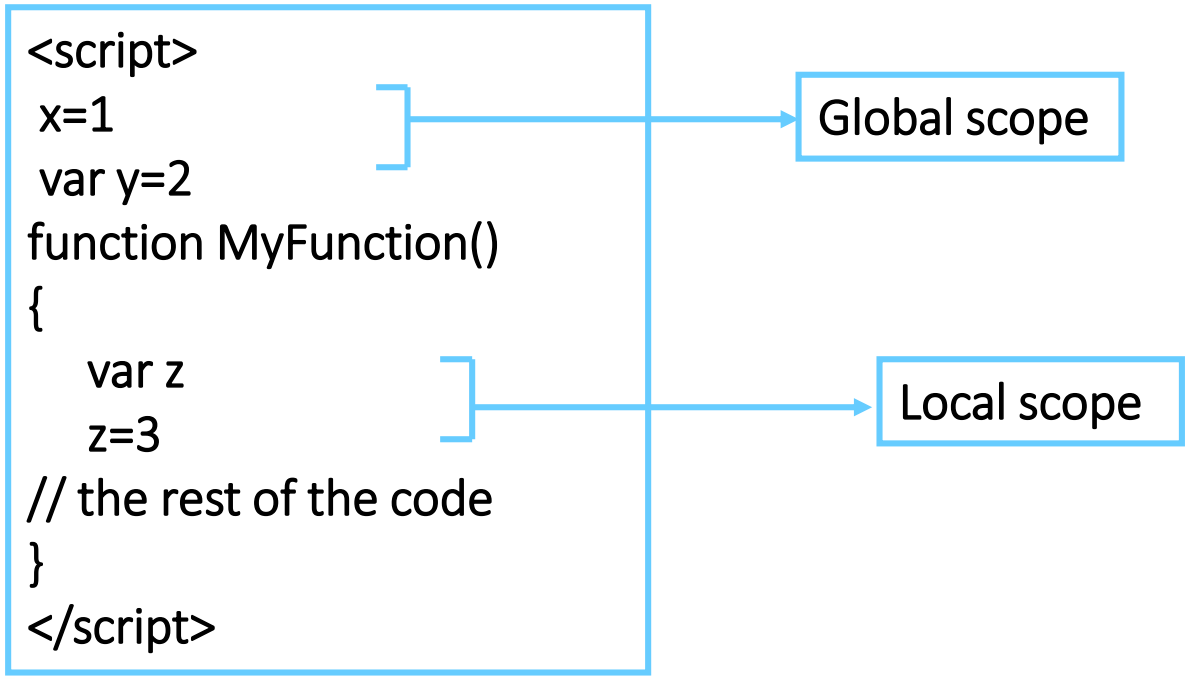
```
doSomething(“hello”)
// value is : hello

doSomething()
// value is : nothing was sent

doSomething(0)
// value is : 0
```

JavaScript Variables Lifetime

- Global Scope
  - A variable declared outside a function and is accessible in any part of your program
- Local Scope
  - A variable inside a function and stops existing when the function ends.



Variable Scope

- It is where var is available in code
- All properties and methods are in the public global scope
  - They are properties of the *global* object "window"
- In JavaScript, var scope is kept within functions, but not within blocks (such as while, if, and for statements) scope
  - NOTE: ES6 represents block scope via let, const.
- Variables declared with var
  - inside a function are local variable
  - outside any function are global variable
    - i.e. available to any other code in the current document

Example!

ES6 Variable Declarations

- **Constants**
  - Constants are declared with const keyword and must be assigned at the time of the declaration.

    const myConst = value;
  - Constants are read-only, can't be modified later .
  - Constants can't be declared with the same name as a function or variable in the same scope.
  - A constant can be global or local to a function where it is declared.
- **"let"**
  - Similar to variables but used for block declaration
- Naming a const/let in JavaScript follow the same rule of naming a var except that the const keyword is always required, even for global constants and let is required for block declaration.
- A script scope is created when let and /or const is declared in global scope

Hoisting

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

- Hoisting takes place before code execution
- Variables declared with var
  - Any variable declared with var is hoisted up to the top of its scope
  - Hoisted variables are of undefined value and can be accessed before being declared. This is called "**Declaration hoisting**"
  - We can refer to a var declared later without getting any exception or reference error.
  - Hoisting of let and const is different from hoisting of var. Both declarations are called "**lexical declarations**" that are considered as **non-hoisting** declarations
- Functions
  - Only function statements are hoisted too.
  - Function statements are available even before its declaration

Example!

Hoisting

- Hoisting is not a universally-agreed term nor a term defined in the ECMAScript specification.

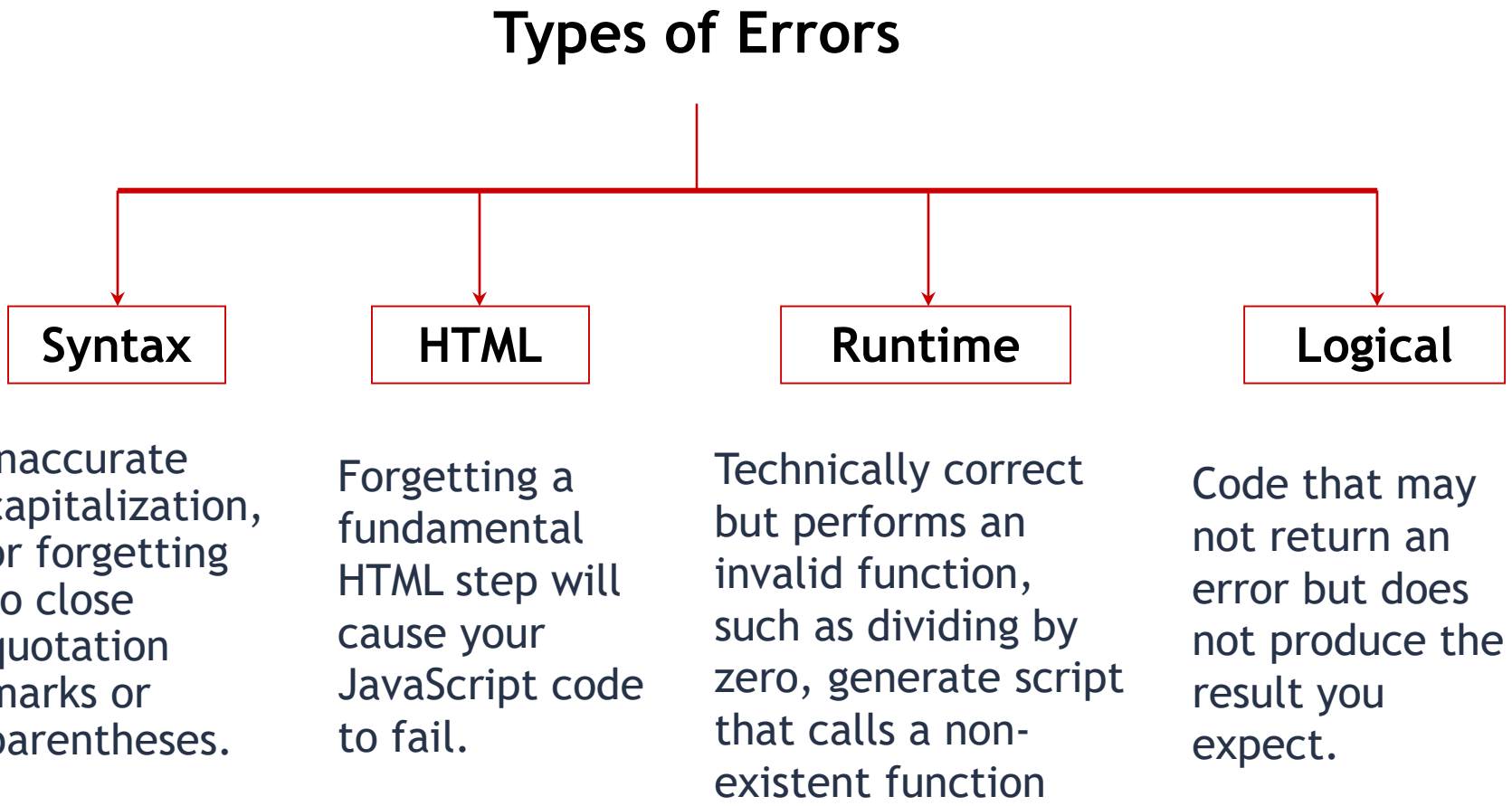
https://developer.mozilla.org/en-US/docs/Glossary/Hoisting

- Types of Hoisting
  - Type 1: Value hoisting
    - Being able to use a variable's value in its scope before the line it is declared.
    - e.g. function statement
  - Type 2: Declaration hoisting
    - Being able to reference a variable in its scope before the line it is declared, without throwing a ReferenceError, and its value is always undefined.
    - e.g. variable declared with var
  - Type 3: Behavior change hoisting
    - The declaration of the variable causes behavior changes in its scope before the line in which it is declared. This takes place due to TDZ
    - e.g. variable declared with let and const

var, let & const

- ES6 represents block scope via let, const.
  - Block starts by { and ends by }
- Variables defined by var & let can be re-assigned
- Variable defined by const cant be re-assigned
- In contrary to var, a variable defined by either const or let cant be re-declared or accessed before their declaration

JavaScript Debugging Errors



JavaScript Console Object

- Modern browsers have JavaScript console within developer tool (F12) where errors in scripts are reported
  - Errors may differ across browsers
- Console Object is a non-standard that provides access to the browser's debugging console
- The console object exists only if there is a debugging tool that supports it.
  - Used to write log messages at runtime
- Do not use it on production

JavaScript Console Object

- Methods of the `console` object:
  - debug(message)
  - log(message)
  - warn(message)
  - error(message)
  - clear()
  - etc..

https://developer.mozilla.org/en/docs/Web/API/console

JavaScript Objects

JavaScript Objects

JavaScript Objects fall into 4 categories:

1. **Custom Objects (User-defined)**
   - Objects that you, as a JavaScript developer, create and use.
2. **Built – in Objects (Native)**
   - Objects that are provided with JavaScript to make your life as a JavaScript developer easier.
3. **BOM Objects "Browser Object Model" (Host)**
   - It is a collection of objects that are accessible through the global objects window. The browser objects deal with the characteristic and properties of the web browser.
4. **DOM Objects "Document Object Model"**
   - Objects provide the foundation for creating dynamic web pages. The DOM provides the ability for a JavaScript script to access, manipulate, and extend the content of a web page dynamically.

JavaScript built-in Objects

JavaScript Built-in Objects

- String
- Number
- Array
- Date
- Math
- Boolean
- RegExp
- Error
- Function
- Object

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

String Object

- Enables us to work with and manipulate strings of text.
- String Objects have:
  - Property
    - length : gives the length of the String.
  - Methods that fall into three categories:
    - Manipulate the contents of the String
    - Manipulate the appearance of the String
    - Convert the String into an HTML element
- To create a String Object
  - var str = new String('hello');

Methods Manipulating the contents of the String Object

```
var myStr = "Let's see what happens!";
```

Method	Example	Returned value
charAt	myStr.charAt(0)	L
charCodeAt	myStr.charCodeAt(12)	97// unicode of a=97
split	myStr.split(“ “,3)	["Let's", "see", "what"]
indexOf	myStr.indexOf("at")	12
lastIndexOf	myStr.lastIndexOf("a")	16
substring	myStr.substring(0, 7)	Let's s
concat	myStr.concat(“ now”);	Let's see what happens! now
replace	myStr.replace(/e/,”?”)	L?t's see what happens!
	myStr.replace(/e/g,”?”);	L?t's s?? what happ?ns!

Methods Manipulating the appearance of the String Object

Name	Example	Returned value
big	“hi".big()	<BIG>hi</BIG>
bold	"hi".bold()	<B>hi</B>
fontcolor	"hi".fontcolor("green")	<FONT COLOR="green“>hi</FONT>
fontsize	"hi".fontsize(1)	<FONT SIZE="1">hi</FONT>
italics	"hi".italics()	<I>hi</I>
small	"hi".small()	<SMALL>hi</SMALL>
strike	"hi".strike()	<STRIKE>hi</STRIKE>
sup	"hi".sup()	^{hi}

Other Useful Methods

Method name
toLowerCase()
toUpperCase()
endsWith()
startsWith()
includes()
repeat()
search()
trim()
trimRight()
trimLeft()

Number Object

- **Number** objects are not primitive objects, but if you use a number method on a primitive number, the primitive will be converted to a Number object behind the scenes and the code will work.
  - It is an object wrapper for primitive numeric values.
- Example:
  - var n = 123;
  - typeof n;
    → "number“
  - n.toString()
    → "123“
  - n.toString(16)
    → “7b"

Number Object

- To create a Number Object

  → var n = new Number(101);

  OR

  → n = new Number();

  // if not assigned a value initially n = 0

  → n=10;

  // value changed to n=10

- Number class has a set of *Constant values* & object methods.

Number Object Constants

1. Class Constants

Properties	Description
Number.MAX_VALUE	A constant property (cannot be changed) that contains the maximum allowed number. →1.7976931348623157e+308
Number.MIN_VALUE	The smallest number you can work with in JavaScript. →5e-324
Number.NaN	Contains the Not A Number number.
Number.POSITIVE_INFINITY	Contains the Infinity number. It is read-only.
Number.NEGATIVE_INFINITY	Has the value -Infinity.

Number Object Constants

- Class Constant Methods

Methods	Example
Number.isInteger()	Number.isInteger(11.2)//false
Number.isFinite()	Number.isFinite(123)//true
Number.isNaN()	Number.isNaN(“aa12”)//true
Number.parseInt()	Number.parseInt(“123”)//123
Number.parseFloat ()	Number.parseFloat (“123.2”)//123.2

Number Object Methods

var n = new Number(10)

Methods	Description	Example

toFixed(x)	Fixed-point representation of a number object as a string. Rounds the returned value.	n = 34.8896;
		n.toFixed(6);
		//34.889600

toExponential(x)	Exponential notation of a number object as a string. Rounds the returned value.	n = 56789;
		n.toExponential(2);
		// "5.68e+4"

toPrecision(x)	Formats any number so it is of "x" length	n = 34.8896; n.toPrecision (3); //34.9

Other Methods

var n = new Number(10)

Methods	Description	Example
toString()	Converts from decimal system to any other system when passing its base as parameter	var x=n.toString(16); //a
	Returns a string representing the Number object.	var numStr = n.toString(); //"10"
valueOf()	returns the primitive value of a Number object as a number data type.	var x = 5 + n.valueOf() ; //15
toLocaleString()	returns a string representing the number with the equivalent language sent as function parameter.	(123). toLocaleString('ar-EG'); //١٢٣

Assignments