

Client-side Technologies

Eng. Niveen Nasr El-Den
SD & Gaming CoE
iTi

Day 3

Basics of JavaScript

“

***Don't imitate..
Understand***

Anonymous

JavaScript

- JavaScript is a scripting language.
- Designed to add **interactivity** to HTML pages and create **dynamic** web sites.
 - ▷ i.e. Change contents of document, provide forms and controls, animation, control web browser window, etc.
- JavaScript statements **embedded** in an HTML page can recognize and **respond** to User Events.

JavaScript

- You can use JavaScript without buying a license.
- You only need a web browser & a text editor.
- Can be used as **object-oriented** language.
- JavaScript is very **simple** and **flexible**
- Powerful, beautiful and full-fledged dynamic Programming Language

Scripting vs. Programming vs. Markup Language

■ Scripting Language

- ▷ Interpreted command by command, and remain in their original form.
- ▷ Output isn't a standalone program or application
 - e.g. JavaScript, Action Script.

■ Programming Language

- ▷ Compiled, converted permanently into binary executable files (i.e., zeros and ones) before they are run.
- ▷ Produce a standalone program or application
 - e.g. C, C++..

■ Markup Language

- ▷ A text-formatting language designed to transform raw text into structured documents, by inserting procedural and descriptive markup into the raw text.
 - e.g. HTML, XHTML

JavaScript History

<http://www.ecma-international.org/ecma-262/5.1/>

- Developed by Brendan Eich at Netscape in **1995** first was called **Mocha** then renamed to **LiveScript**.

- In Navigator 2.0, name changed to **JavaScript** as a result of an agreement with Sun, the developer of Java.

<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

- Later, in **1997** ; **ECMAScript** was introduced by ECMA International as an attempt at standardization.

- Microsoft recognized the importance of JavaScript and entered the arena with two creations, JScript and VBscript.

JavaScript Characteristics

- Case sensitive.
- Object-based.
- Event-Driven.
- Browser-Dependent.
- Platform Independent.
- Interpreted language.
- Dynamic.



JavaScript Strength & Weakness

Strength

- Quick Development
- Easy to Learn
- Platform Independence
- Small Overhead

Weakness

- Limited Range of Built-in Methods
- No Code Hiding
- Altering the text on an HTML page will reload the entire document

What JavaScript Can do

- Giving the user more control over the browser.
 - ▷ i.e. open and create new browser window
- Detecting the user's browser, OS, screen size, etc.
- Performing **computations** on the client side.
- **Validating** the user's input data.
 - ▷ i.e. it validates the data on the user's machine before it is forwarded to the server.
- Can handle **events**, **exceptions**, etc..
- Can create **cookies**.
- Can **read** and **change** the content of an HTML element.
- Powerful to manipulate the **DOM**
 - ▷ DOM is a representation of the web page, which can be modified with JavaScript.

What JavaScript Can't do

- *Directly* access files on the user's system or the client-side LAN ; the only exception is the access to the browser's cookie files, because it is created within the script itself.
- *Directly* access files on the Web server.

Earlier, developers have to bear in mind ***the biggest JavaScript limitation:***
the user can always disable JavaScript!

How Does JavaScript Work?

- JavaScript statements are usually embedded directly in HTML code, using a `<script>` element.
- Scripts can go either in the head or body of the document.
- We can write JavaScript:
 1. Anywhere in the html file between `<script></script>` tags.
 2. As the value of the event handler attributes.
 3. In an external file and refer to it using the `src` attribute.

Embedding JavaScript in HTML

1. Anywhere in the html file between <script> tags.

```
<head>
  <title>A Simple Document</title>
  <script type="text/javascript">
    //JavaScript code goes here
    document.write ("Hello world")
  </script >
</head>
<body>
  <p>Page content</p>
  <script>
    document.write (" welcome to JavaScript world")
  </script>
</body>
```

Embedding JavaScript in HTML

2. As the value of the event handler attributes.

```
<head>
  <title>A Simple Document</title>
</head>
<body>
  We can write it at the event handlers
  <a href="try1.htm" onclick= "alert('Hello world') " >
    click here to run JavaScript code
  </a>
</body>
```

Embedding JavaScript in HTML

3. In an external file and refer to it using the **src** attribute.

```
<head>
  <title>A Simple Document</title>
  <script src= "MyJavascrIpFile.js"></script>
</head>
<body>
  We can refer to JavaScript statements in another file.
</body>
```


JavaScript Variables Declarations

- Variables are containers that hold values.
- Variables are **loosly** typed, initial value is **undefined**.
 - ▷ `var num; //num = undefined`
- While it is not technically necessary, variable declarations should begin with the keyword **var** to keep tracking of a variable easily.
- Assignment:
`var myVar = value;`
 - `var month = "June";`
 - `month = "June";`

JavaScript Variables Naming

- **First** character must be a letter (a-z or A-Z) or an underscore (_), and the **rest** of the name can be (a-z or A-Z), (0-9), or underscores (_).
- Don't use spaces inside names.
 - ▷ **FirstName** NOT **First Name**.
- Avoid reserved words, words that are used for other purposes in JavaScript.
 - ▷ i.e. **you couldn't call a variable alert or goto.**
- Case-sensitive
 - ▷ **FirstName** differ from **firstName**
- Variables should have meaningful and descriptive names to describe what they are.
- The common naming convention in JavaScript is to use two words with no space between them, and capitalize the second word but not the first.

JavaScript Datatypes

- JavaScript is a *loosely* typed *dynamic* language.
 - ▷ No need to declare the type of a variable before using it.
 - ▷ Same variable can contain different types of data values.
- The latest ECMAScript standard defines *six* primitives data types and an Object

JavaScript Primitive Datatypes

Value	Example
Number	Any numeric value (e.g., 3, 5.3, 45e8, 055, 0x4A)
String	Any string of alphanumeric characters (e.g., "Hello, World!", "555-1212" or "KA12V2B334")
Boolean	true or false values only

JavaScript Special Primitive Values

Value	Example
null	A special keyword for the null value (no value or empty variable)
undefined	A special keyword means that a value hasn't even been assigned yet. Better to be used by JavaScript engine

undefined is the value of a variable with no value.

Variables can be emptied by setting the value to **null**;

JavaScript Operators

- Operators are functions
- JavaScript supports:
 - 1- Binary operators:
 - Require two operands in the expression such as `x+2`
 - 2- Unary operators:
 - Requires one operand such as `x++`
 - 3- Ternary operators:
 - Requires three operands

Arithmetic Operators

Operator	Type	Description
+	Addition	Adds the operands together.
-	Subtarction	Subtracts the right operand from the left operand
*	Multiplication	Multiplies together the operands.
/	Division	Divides the left operand by the right operand.
%	Modulus arithmetic	Divides the left operand by the right operand and calculates the remainder.
! -	unary	Negates the value of the operand.
++	Unary (Increment)	Increases the value of the supplied operand by one.
--	Unary (Decrement)	Decreases the value of the supplied operand by one.

Assignment Operators

(**x = 10** and **y =5**)

Operator	Example	Description
=	x = y Sets x to the value of y	Assigns the value of the right operand to the left operand
+=	x += y i.e. x = x + y (15)	Adds together the operands and assigns the result to the left operand.
-=	x -= y i.e. x = x - y (5)	Subtracts the right operand from the left operand and assigns the result to the left operand.
*=	x *= y i.e. x = x * y (50)	Multiplies together the operands and assigns the result to the left operand.
/=	x /= y i.e. x = x / y (2)	Divides the left operand by the right operand and assigns the result to the left operand.
%=	x %= y i.e. x = x % y (0)	Divides the left operand by the right operand and assigns the result to the left operand.

Comparison Operators

Operator	Definition
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Loose Equality (double equals)
!=	Inequality
===	Strict Equality (double equals or identity)
!==	Strict Inequality

Logical Operators

Operator	Description
A && B Logical "AND"	-Dealing with Boolean, it returns true when both operands are true; otherwise it returns false -otherwise, it returns A if it can be converted to false; otherwise, returns B
A B Logical "OR"	-Dealing with Boolean, returns true if either operand is true. It only returns false when both operands are false -otherwise, it returns A if it can be converted to true; otherwise, returns B
! Logical "NOT"	returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand

String Operators

- **+ operator**

Combines the operands into a single string.
i.e. used in sting concatenation.

- **Example:**

```
<script>
```

```
A="Welcome "
```

```
B="Ali"
```

```
C=A+B
```

```
document.write (C)
```

```
// the result will be "Welcome Ali"
```

```
</script>
```

Special Operators

- **?:** Conditional Ternary Operator
- **,** Comma Operator
- **new** Operator
- **this** Operator
- Unary Operators
 - ▷ **delete** Operator
 - ▷ **typeof** Operator
 - ▷ **void** Operator
- Relational Operators
 - ▷ **instanceof** Operator
 - ▷ **in** Operator

Ternary Operator

- (test_Condition) ? if true : if false

Evaluates to one of two different values based on a condition.

- Example :

```
<script>
```

```
temp=120
```

```
newvar=(temp>100) ? "red" : "blue"
```

```
// the value of newvar will be "red"
```

```
temp=20
```

```
newvar=(temp>100) ? "red" : "blue"
```

```
// the value of newvar will be "blue"
```

```
</script>
```

Comma Operator

- The (, **operator**) cause two expressions to be executed sequentially.
- It is commonly used when
 - ▷ naming variables,
 - ▷ in the increment expression of a **for** loop,
 - ▷ in function calls, arrays and object declarations.
- The (, operator) causes the expressions on either side of it to be executed in left-to-right order, and obtains the value of the expression on the right.
 - ▷ Example:
`var k=0, i, j=0;`

typeof Operator

- **typeof** Operator

- ▷ A unary operator returns a string that represents the data type.
- ▷ The return values of using **typeof** can be one of the following:
 - "number", "string", "boolean", "undefined", "object", or "function".

- Example:

```
var myName = "javascript";  
typeof myName;    //string
```

JavaScript Expression

- An expression is a part of a statement that is evaluated as a value.
- Main types of expressions:
 - ▷ Left-hand-side “Assignment”
 - `a = 25;` → assign RHS to variable of LHS
 - ▷ Arithmetic
 - `10 + 12;` → evaluates to sum of 10 and 12
 - ▷ String
 - `“Hello” + ” All !!”;` → evaluates to new string
 - ▷ Logical
 - `25<27` → evaluates to the Boolean value

Coercion

- Coercion is **forcing conversion** from one data type to another when expression is executed giving a result without causing any error.
- Sometimes gives **surprising** results from human perspective
- JavaScript engine coerce
 - ▷ Number to string
 - `1+"2" → 12`
 - ▷ Boolean to number
 - `3<2<1 → true`
 - ▷ Both undefined and null coerce to false

Avoid it by using

===
()

Precedence & Association

- Operator precedence
 - ▷ Determines the order in which operators are evaluated. Operators with higher precedence are evaluated first
- Operator Associativity
 - ▷ Determines the **order** in which operators of the **same** precedence are processed

Operator Precedence

- The operators that you have learned are evaluated in the following order (from highest precedence to lowest):

1. Parentheses(`()`)
2. Multiply/divide/modulus (`*`, `/`, `%`)
3. Addition/Subtraction (`+`, `-`)
4. Relational (`<`, `<=`, `>=`, `>`)
5. Equality (`==`, `!=`)
6. Logical and (`&&`)
7. Logical or (`||`)
8. Conditional (`?:`)
9. Assignment operators (`=`, `+=`, `-=`, `*=`, `/=`, `%=`)

Example:

$5 + 3 * 2 = 11 \rightarrow 3 * 2 = 6$, then $6 + 5 = 11$.

BUT

$(5 + 3) * 2 = 16 \rightarrow 5 + 3 = 8$, then $8 * 2 = 16$.

Controlling Program Flow

- Program flow is normally linear, i.e. each statement is processed in its turn.
- One of the more common approaches to changing the program flow in JavaScript is through *Control Statements*.
- Control Statements that can be used are:
 1. Conditional Statements
 - a. if ...else
 - b. switch/case
 2. Loop Statements
 - a. for
 - b. for..in
 - c. while
 - d. do...while

Control Statements

Conditional Statements

a) **if....else**

```
if (condition)
{
    statements if condition is
true;
}
else
{
    statements if condition is
false;
}
```

b) **switch / case**

```
switch (expression)
{
case label1:statements
break
case label2:statements
break
default :
}
```

Control Statements

■ Example:

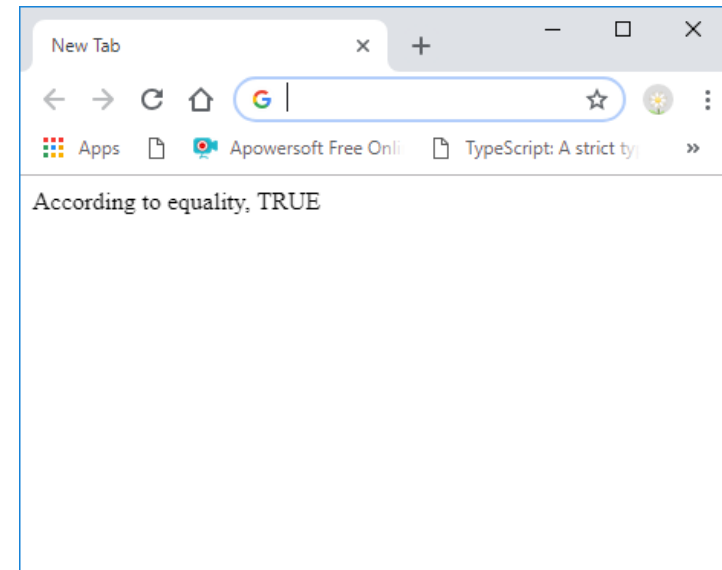
```
var nValue = 3.0;  
var sValue = "3.0";
```

```
if (nValue == sValue)
```

```
    document.write("According to equality, TRUE");
```

```
if (nValue === sValue)
```

```
    document.write("According to identity, FALSE");
```



Control Statements

Looping Statements

a) **for**

```
for ( initExp ; condition ; updateExp )
{
    statements;
}
```

b) **for..in**

```
for (variablename in object)
{
    statement;
}
```

c) **while**

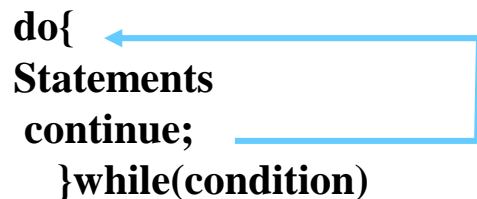
```
while (condition)
{
    statements
}
```

d) **do...while**

```
do
{
    statements
}while(condition)
```


Continue

```
do{
    Statements
    continue;
}while(condition)
```



Break

```
do{
    Statements
    break;
}while(condition)
```



Communicating with the User

- Four ways of communication:
 - ▷ one that displays a text message in a pop-up window,
 - ▷ one that asks for information in a pop-up window,
 - ▷ one that asks a question in a pop-up window,
 - ▷ and one that displays a text message in the browser window.

Outputting text with JavaScript *(on the current window)*

- You can write out **plain text** or you can mix **HTML tags** in with the text being written using **document.write()** to return text to the browser screen.

**document.write(" ") Or
document.writeln(" ") Methods**

- Example:
 document.write("Hello There!")
 document.writeln("Hello There!")

alert() : Giving the user a pop up message

- pop up when it is called

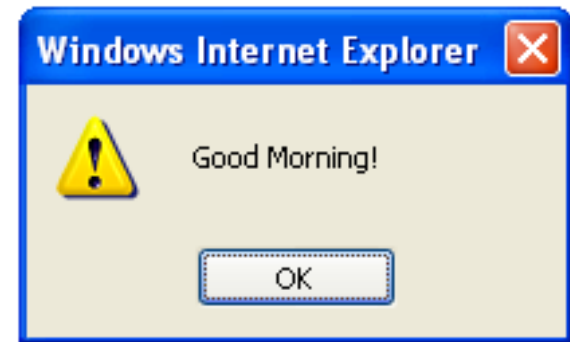
- **Syntax**

`alert("message");`

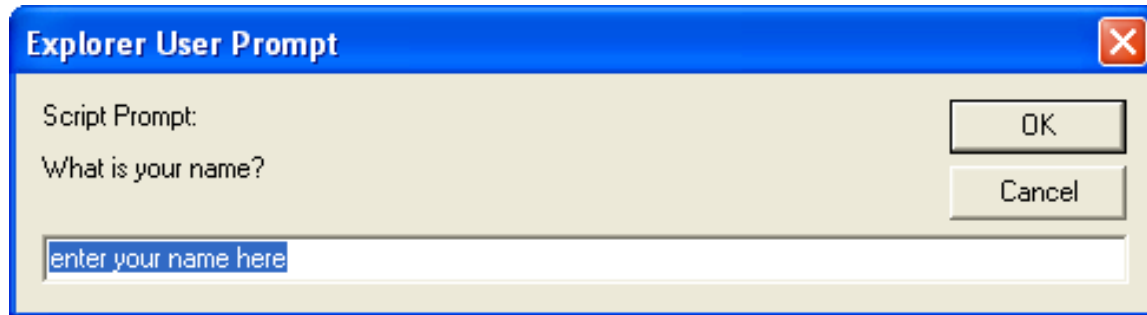
- The script

`alert("Good Morning!");`

HTML holding the script will not continue or execute until the user clicks the OK button.



prompt() : Getting data from the user



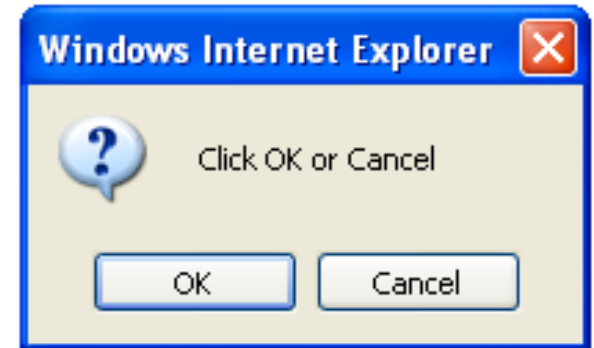
- The user needs to fill in a field and then press *OK* or *Cancel* button.
- **Syntax**
`prompt("Message to user", "default response text");`
- When you press *OK* the value you typed in the *field* is returned.
- When you press *Cancel* the value *null* is returned.

confirm() : ask the user a simple "yes or no" type of question

- Confirm displays a dialog box with two buttons: OK and Cancel
- It is similar to the alert() method with one significant exception: confirm() returns a value of either true or false.

- **Syntax**

`confirm("Question to the user");`



- It is a message box that provides both OK and Cancel buttons
 - ▷ If the user clicks on **OK** it will return **true**.
 - ▷ If the user clicks on the **Cancel** it will return **false**.

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.

JavaScript Built-in functions

Name	Example
parseInt(s,r)	<code>parseInt("3") //returns 3</code> <code>parseInt("3a") //returns 3</code> <code>parseInt("a3") //returns NaN</code> <code>parseInt("110", 2)// returns 6</code> <code>parseInt("0xD9", 16)// returns 217</code>
parseFloat(s)	<code>parseFloat("3.55") //returns 3.55</code> <code>parseFloat("3.55a") //returns 3.55</code> <code>parseFloat("a3.55") //returns NaN</code>
Number(objArg)	converts the object argument to a number representing the object's value.
String(objArg)	converts the object argument to a string representing the object's value.

JavaScript Built-in functions

Name	Description	Example
isFinite(num) (used to test number)	returns true if the number is finite, else false	<pre>document.write(isFinite("2.2345")) //returns true document.write(isFinite("Hello")) //returns false</pre>
isNaN(val) (used to test value)	validate the argument for a number and returns true if the given value is not a number else returns false.	<pre>document.write(isNaN("hello")) //returns true document.write(isNaN("348")) //returns false</pre>

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
}
```

Function parameters

```
dosomething("hello")
```

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
```

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.

```
<script>  
x=1  
var y=2  
function MyFunction()  
{  
    var z  
    z=3  
    // the rest of the code  
}  
</script>
```

Global scope

Local scope

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)
- Variables declared
 - ▷ **inside** a function are **local** variable
 - ▷ **outside** any function are **global** variable
 - i.e. available to any other code in the current document

Example!

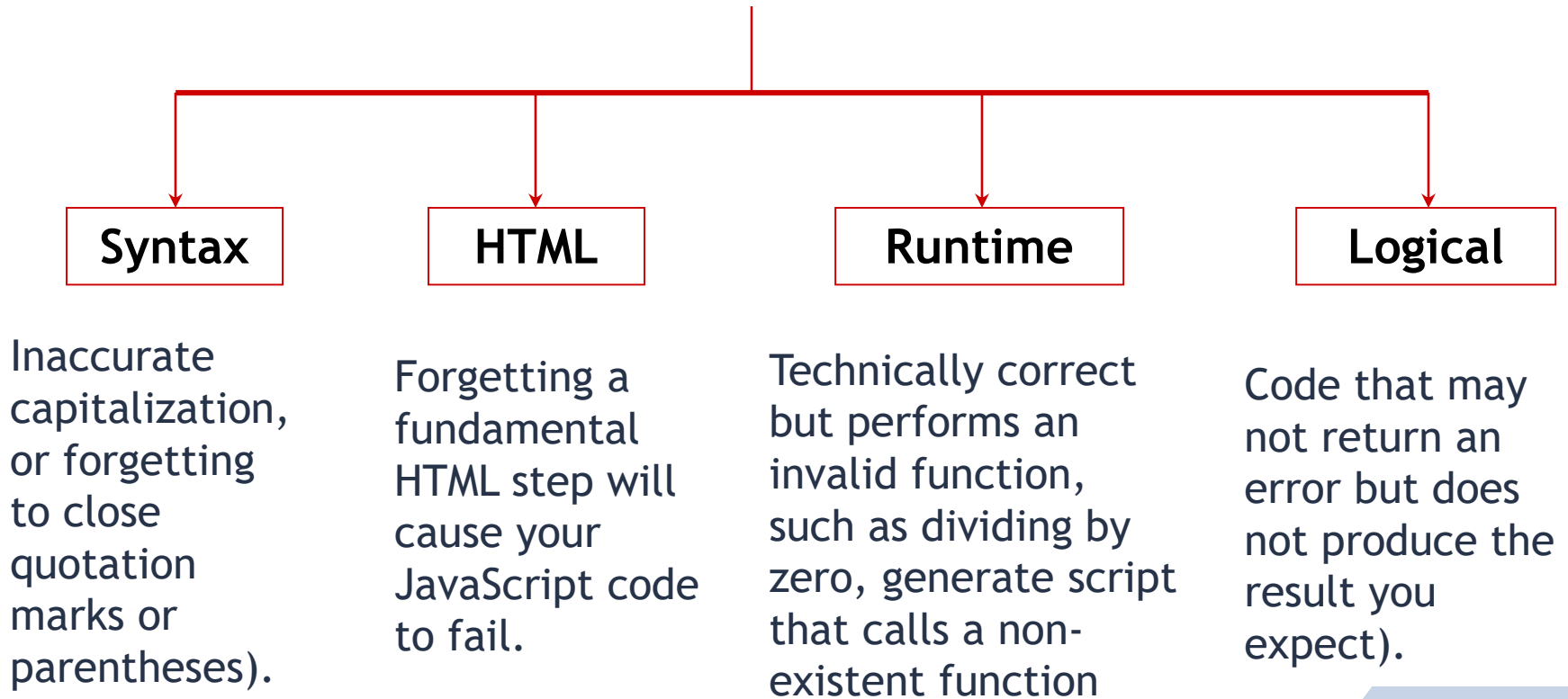
Hoisting

- Hoisting takes place before code execution
- Variables
 - ▷ Any variable declared with **var** is **hoisted** up to the top of its scope
 - ▷ Hoisted variables are of **undefined** value.
 - ▷ We can refer to a variable declared later without getting any exception or error
- Functions
 - ▷ Function **statements** are **hoisted** too.
 - ▷ Functions are available even before its declaration

Example!

JavaScript Debugging Errors

Types of Errors



JavaScript Debugging

- Modern browsers have JavaScript console within developer tool (F12) where errors in scripts are reported
 - ▷ Errors may differ across browsers
- Several tools to debug JavaScript
 - ▷ Debugging in MSIE:
 - Microsoft Script Editor (installed with MS Office)
 - Microsoft Developer Tool
 - ▷ Debugging in Firefox:
 - Download and use firefox add-on *Firebug*

JavaScript Console Object

- 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)
- ▷ etc..

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

JavaScript Objects

The background features abstract geometric shapes. A large light blue triangle points downwards from the top left. A dark blue trapezoidal shape points to the right, containing the text. In the bottom right corner, there are overlapping light blue and orange trapezoidal shapes.

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

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	<code>"hi".big()</code>	<code><BIG>hi</BIG></code>
bold	<code>"hi".bold()</code>	<code>hi</code>
fontcolor	<code>"hi".fontcolor("green")</code>	<code>hi</code>
fontsize	<code>"hi".fontsize(1)</code>	<code>hi</code>
italics	<code>"hi".italics()</code>	<code><i>hi</i></code>
small	<code>"hi".small()</code>	<code><SMALL>hi</SMALL></code>
strike	<code>"hi".strike()</code>	<code><STRIKE>hi</STRIKE></code>
sup	<code>"hi".sup()</code>	<code><SUP>hi</SUP></code>

Other Useful Methods

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

RegExp Object

- Regular expressions provide a powerful way to search and manipulate text.
- A Regular Expression is a way of representing a pattern you are looking for in a string.
- A Regular Expression lets you build *patterns* using a set of special characters. Depending on whether or not there's a match, appropriate action can be taken.
- People often use regular expressions for *validation* purposes.
 - ▷ In the validation process; you don't know what exact values the user will enter, but you do know the format they need to use.

RegExp Object

- Specified literally as a sequence of characters with forward slashes (/) or as a JavaScript string passed to the RegExp() constructor
- A regular expression consists of:
 - ▷ A **pattern** used to match text, Mandatory parameter.
 - ▷ Zero or more **modifiers** (also called **flags**) that provide more instructions on how the pattern should be applied, Optional parameter.

RegExp Object

- to create regular expression objects

- ▷ Explicitly using the RegExp object

- `var searchPattern = new RegExp("pattern" [, "flag"]);`
 - `var re = new RegExp("j.*t")`

- ▷ Using literal RegExp

- `var myRegExp = / pattern / [flag] ;`
 - `var re = /j.*t/;`

- In the example above,

- ▷ `j.*t` is the regular expression pattern. It means, "Match any string that starts with `j`, ends with `t` and has zero or more characters in between".
 - ▷ The asterisk `*` means "zero or more of the preceding";
 - ▷ the dot `(.)` means "any character"

RegExp Object

- **Modifiers** can be passed as a second parameter in any combination of the following characters and in any order

- ▷ "g" for global
- ▷ "i" for ignoreCase
- ▷ "m" for multiline
- ▷ etc.

<https://javascript.info/regexp-introduction>

- Example:

- ▷ `var re = new RegExp('j.*t', 'gmi');`
- ▷ `var re = /j.*t/ig;`

RegExp Object Properties

- **global:**
 - ▷ If this property is false, which is the default, the search stops when the first match is found. Set this to true if you want all matches.
- **ignoreCase:**
 - ▷ Case sensitive match or not, defaults to false.
- **multiline:**
 - ▷ Search matches that may span over more than one line, defaults to false.
- **lastIndex:**
 - ▷ The position at which to start the search, defaults to 0.
- **source:**
 - ▷ Contains the regexp pattern.

Once set, the modifier cannot be changed

RegExp Methods

- **test()**

- ▷ returns a boolean (true when there's a match, false otherwise)
- ▷ Example:

`/j.*t/.test("Javascript")`

→ false

case sensitive

- **exec()**

- ▷ returns an array of matched strings.
- ▷ Example:

`/j.*t/i.exec("Javascript")[0]`

→ "Javascript"

String Methods that Accept Regular Expressions as Parameters

- **.match(regex)**
 - ▷ returns an array of matches
- **.search(regex)**
 - ▷ returns the position of the first match
- **.replace(regex, txt)**
 - ▷ allows you to substitute matched text with another string
- **.split(delimiter [, limit])**
 - ▷ also accepts a RegExp when splitting a string into array elements

RegExp Syntax

Character	Description	Example
.	Any character	<code>/a.*a/</code> matches "aa", "aba", "a9qa", "a!?!_a",
^	Start	<code>/^a/</code> matches "apple", "abcde"..
\$	End	<code>/z\$/</code> matches "abcz", "az"..
	Or	<code>/abc def g/</code> matches lines with "abc", "def", or "g"
[]	Match any one character between the brackets	<code>/[a-z]/</code> matches any lowercase letter
[^]	Match any one character not between the brackets	<code>/[^abcd]/</code> matches any character but not a, b, c, or d

RegExp Syntax

Character	Description	Example	
*	0 or more	/Goo <u>o</u> gle/ → "Gogle", "Go <u>o</u> gle", "Go <u>oo</u> gle", "Go <u>ooo</u> gle	
+	1 or more	/Goo <u>o</u> gle/ → "Go <u>o</u> gle", "Go <u>oo</u> gle", "Go <u>ooo</u> gle	
?	0 or 1	/Goo <u>o</u> gle/ → "Gogle", "Go <u>o</u> gle",	
{min, max}	{min,} → min or more	{2,} 2 or more	/a(bc){2,4}/ → "a <u>bc</u> bc", "a <u>bc</u> bc <u>bc</u> ", or "a <u>bc</u> bc <u>bc</u> bc"
	{,max} → up to max	{,6} up to 6	
	{val} → exact value	{3} exactly 3	

Assignment