

Client-side Technologies

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

The background features a large, dark blue trapezoidal shape on the left side, which tapers towards the right. To the right of this shape is a white area. At the bottom, there is a horizontal orange bar that also tapers towards the right. The overall design is minimalist and geometric.

Day 3

The background features a large, dark blue trapezoidal shape on the left side, which tapers towards the right. To the right of this shape is a white area. At the bottom, there is a horizontal orange bar that also tapers towards the right. The overall design is modern and geometric.

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
}
```

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) scope
 - ▷ NOTE: **ES6** represents block scope via **let**, **const**.
- 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!

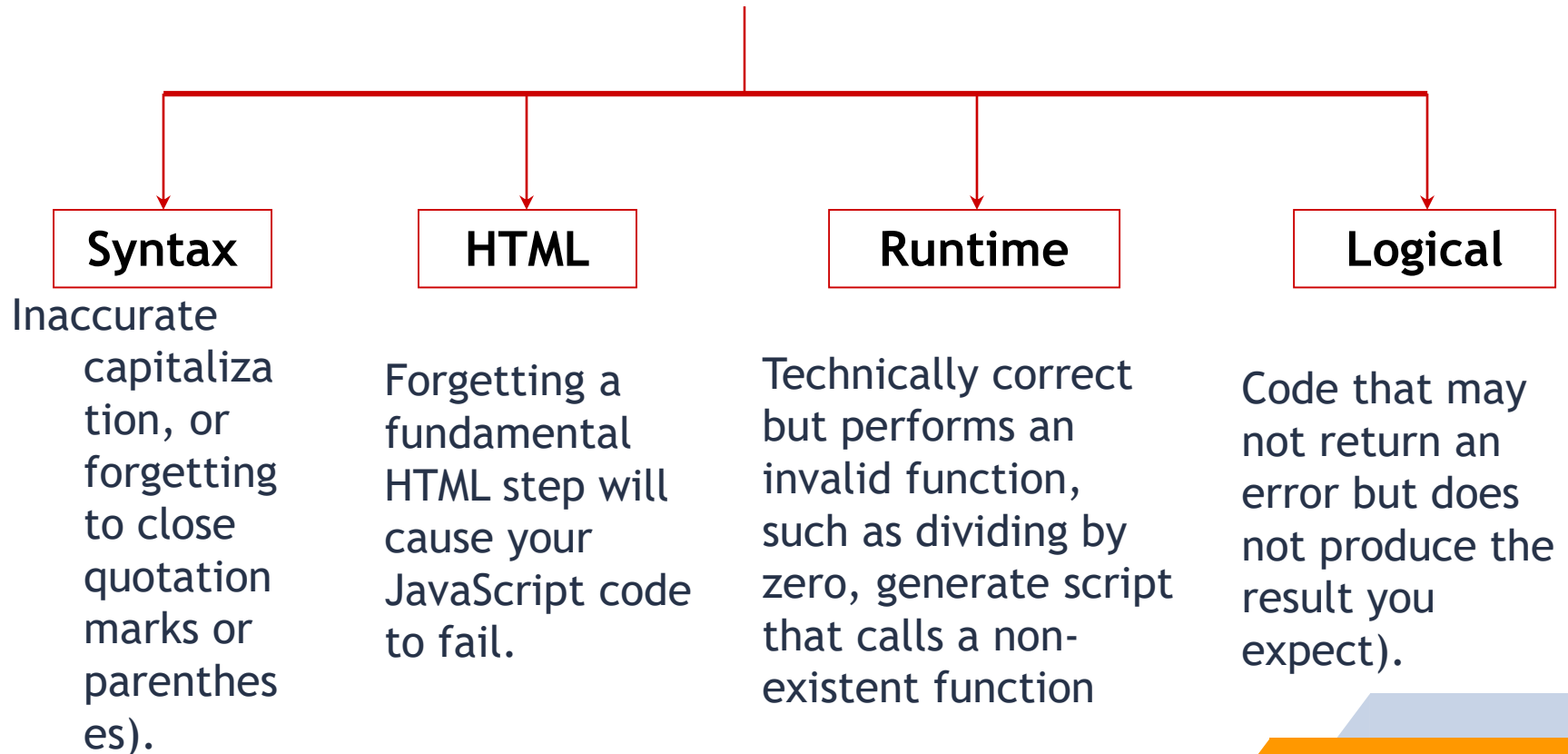
Hoisting

- 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 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)
- ▷ 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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_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>	<BIG>hi</BIG>
bold	<code>"hi".bold()</code>	hi
fontcolor	<code>"hi".fontcolor("green")</code>	hi
fontsize	<code>"hi".fontsize(1)</code>	hi
italics	<code>"hi".italics()</code>	<I>hi</I>
small	<code>"hi".small()</code>	<SMALL>hi</SMALL>
strike	<code>"hi".strike()</code>	<STRIKE>hi</STRIKE>
sup	<code>"hi".sup()</code>	<SUP>hi</SUP>

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
<code>Number.isInteger()</code>	<code>Number.isInteger(11.2)//false</code>
<code>Number.isFinite()</code>	<code>Number.isFinite(123)//true</code>
<code>Number.isNaN()</code>	<code>Number.isNaN("aa12")//true</code>
<code>Number.parseInt()</code>	<code>Number.parseInt("123")//123</code>
<code>Number.parseFloat ()</code>	<code>Number.parseFloat ("123.2")//123.2</code>

Number Object Methods

```
var n = new Number(10)
```

Methods	Description	Example
<code>toFixed(x)</code>	Fixed-point representation of a number object as a string. Rounds the returned value.	<code>n = 34.8896; n.toFixed(6); //34.889600</code>
<code>toExponential(x)</code>	Exponential notation of a number object as a string. Rounds the returned value.	<code>n = 56789; n.toExponential(2); // "5.68e+4"</code>
<code>toPrecision(x)</code>	Formats any number so it is of "x" length	<code>n = 34.8896; n.toPrecision (3); //34.9</code>

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	<pre>var x=n.toString(16); //a</pre>
	Returns a string representing the Number object.	<pre>var numStr = n.toString(); //" 10"</pre>
valueOf()	returns the primitive value of a Number object as a number data type.	<pre>var x = 5 + n.valueOf() ; //15</pre>
toLocaleString()	returns a string representing the number with the equivalent language sent as function parameter.	<pre>(123). toLocaleString('ar-EG'); //١٢٣</pre>

Math Object

- Allows you to perform common mathematical tasks.
- The Math object is a *static object*.
- Math is a little different from other built in objects because it **cannot** be used as a constructor to **create** objects.
- Its just a **collection** of **functions** and **constants**

Math Object

- Math object has:

- I- Properties (constant values)

- II- Methods

- Example:

- ```
var circleArea = Math.PI * radius * radius;
```

# Math Object Properties

| Name                | Returned value                      |
|---------------------|-------------------------------------|
| <b>Math.E</b>       | Returns Euler's constant            |
| <b>Math.PI</b>      | Return the value of $\pi$ (PI)      |
| <b>Math.SQRT2</b>   | Returns the square root of 2        |
| <b>Math.SQRT1_2</b> | Returns the square root of 0.5      |
| <b>Math.LN2</b>     | Returns the natural logarithm of 2  |
| <b>Math.LN10</b>    | Returns the natural logarithm of 10 |
| <b>Math.LOG2E</b>   | Returns the log base -2 of E        |
| <b>Math.LOG10E</b>  | Returns the log base -10 of E       |

# Math Object Methods

| Name   | Example                              | Returned value   |
|--------|--------------------------------------|------------------|
| max    | <code>Math.max(1 , 700)</code>       | 700              |
| min    | <code>Math.min(1 , 700)</code>       | 1                |
| sqrt   | <code>Math.sqrt(9801)</code>         | 99               |
| pow    | <code>Math.pow(6, 2)</code>          | 36               |
| random | <code>Math.random()</code>           | .7877896         |
| round  | <code>Math.round(0.567)</code>       | 1                |
| floor  | <code>Math.floor(0.567)</code>       | 0                |
| ceil   | <code>Math.ceil(0.567)</code>        | 1                |
| sin    | <code>Math.sin(Math.PI)</code>       | 0                |
| cos    | <code>Math.cos(Math.PI)</code>       | -1               |
| tan    | <code>Math.tan(1.5 * Math.PI)</code> | 5443746451065123 |



# Math Object Methods

| Name        | Example                | Returned value               |
|-------------|------------------------|------------------------------|
| <b>abs</b>  | <b>Math.abs(-6.5)</b>  | <b>6.5</b>                   |
| <b>acos</b> | <b>Math.acos(.5)</b>   | <b>1.047197551196597631</b>  |
| <b>asin</b> | <b>Math.asin(1)</b>    | <b>1.570796326794896558</b>  |
| <b>atan</b> | <b>Math.atan(.5)</b>   | <b>0.4636476090008060935</b> |
| <b>sqrt</b> | <b>Math.sqrt(9801)</b> | <b>99</b>                    |
| <b>exp</b>  | <b>Math.exp(8)</b>     | <b>2980.957987041728302</b>  |
| <b>log</b>  | <b>Math.log(5)</b>     | <b>1.609437912434100282</b>  |

# Array Object

- Array is actually a special type of object
- It has **length** property:
  - ▷ gives the length of the array
  - ▷ It is one more than the highest index in the array
- To declare an array use
  - ▷ new keyword
  - ▷ array literal notation

# Array Object

## ■ Using new operator:

→ `var colorArray = new Array();`  
`colorArray [0]="red";`  
`colorArray [1]="blue";`  
`colorArray [2]="green";`

OR

→ `var colorArray = new Array(3);`  
`colorArray [0]="red";`  
`colorArray [1]="blue";`  
`colorArray [2]="green";`

OR

→ `var colorArray = new Array("red","blue","green");`

//this is called **dense** array where array is populated at the time it is declared

## ■ Use array literal notation

→ `var arr = ["apple", "banana", "grapes"];`  
→ `var arr = [ , 1, , , "a"];`

# Array Object Methods

```
var arr1=new Array("A","B","C");
```

```
var arr2 = new Array(1,2,0);
```

| Name    | Example                       | Result                                                           |
|---------|-------------------------------|------------------------------------------------------------------|
| concat  | arr1.concat(arr2);            | A,B,C,1,2,0<br>//neither arr1 nor arr2 changed                   |
| join    | arr1.join()<br>arr1.join("*") | A,B,C<br>A*B*C<br>//arr1 not changed                             |
| reverse | arr1.reverse()                | C,B,A                                                            |
| pop     | arr1.pop()                    | C<br>// and arr1.length becomes 2                                |
| push    | arr1.push("D");               | 4<br>// 4 → Length of the array<br>// resulting in : arr1[3]="D" |

# Array Object Methods

```
var arr1=new Array("A","B","C");
```

```
var arr2 = new Array(4,2,3,0);
```

| Name                              | Example                          | Result                                   |
|-----------------------------------|----------------------------------|------------------------------------------|
| shift                             | arr1.shift();                    | Returns: A<br>arr1[0] = "B" & arr[1]="C" |
| unshift                           | arr1.unshift("D");               | arr1[0]="D"<br>//length become 4         |
| slice                             | arr1.slice(1);<br>arr1.slice(2); | B,C<br>C<br>//arr1 not changed           |
| sort<br>(according to<br>Unicode) | arr2.sort()                      | 0,2,3,4                                  |

# Associative Array

## ■ The Arrays That Aren't

- Associative array is just like an ordinary array, except that instead of the indices being numbers, they're **strings**, which can be a lot easier to remember and reference.
- The key idea is that every JavaScript object is an associative array
- Can't be access the array using numeric indexes.
- Associative arrays let you specify key-value pairs.
- Although the keys for an associative array have to be strings, the values can be of any data type, including other arrays or associative arrays.

## ■ Syntax:

```
var assocArray = new Array();
assocArray["one"] = "one";
assocArray["1"] = "two";
assocArray["Next Value"] = "Three";
assocArray["new"] = 2;
```

# Date Object

- To obtain and manipulate the day and time in a script.
- The information either takes the value from the user's computer or from a specified date and time

- To create date object:

`var varName = new Date([parameters])`

▷ Parameters are

- Year, month, date of the month, hour, minute, second, and milliseconds

▷ Example:

`var varName = new Date()`

`var varName = new Date(milliseconds)`

`var varName = new Date(datestring)`

`var varName = new Date(yr, month, date [, hrs, min, sec, msec])`

# Date Object Number Conventions

| Date Attribute   | Numeric Range                                    |
|------------------|--------------------------------------------------|
| seconds, minutes | 0 - 59                                           |
| hours            | 0 - 23                                           |
| day              | 0 - 6<br>(0 = Sunday, 1 = Monday, and so on)     |
| date             | 1 - 31                                           |
| month            | 0 - 11<br>(0 = January, 1 = February, and so on) |
| year             | 0 + number of years since 1900                   |



# Date Object

- The Date object methods fall into these broad categories:

1. *"get"* methods

→ for getting date and time values from date objects

2. *"set"* methods

→ for setting date and time values in date objects

3. *"to"* methods

→ for returning string values from date objects.

# Date Object “get” Methods

```
var now = new Date ("November 25,2009");
```

| Name           | Example          | Returned Value                                                                     |
|----------------|------------------|------------------------------------------------------------------------------------|
| getDate        | now.getDate()    | 25                                                                                 |
| getMonth       | now.getMonth()   | 10                                                                                 |
| <b>getYear</b> | now.getYear()    | 109                                                                                |
| getDay         | now.getDay()     | 6                                                                                  |
| getHours       | now.getHours()   | 0                                                                                  |
| getMinutes     | now.getMinutes() | 0                                                                                  |
| getSeconds     | now.getSeconds() | 0                                                                                  |
| getTime        | now.getTime()    | The internal, millisecond representation of a Date object similar to now.valueOf() |

# Date Object “set” Methods

```
var someDate= new Date ();
```

| Name       | Example                               |
|------------|---------------------------------------|
| setDate    | someDate.setDate(6)                   |
| setHours   | someDate.setHours(14)                 |
| setMinutes | someDate.setMinutes(50)               |
| setMonth   | someDate.setMonth(7)                  |
| setSeconds | someDate.setSeconds(7)                |
| setTime    | someDate.setTime(yesterday.getTime()) |
| setYear    | someDate.setYear(88)                  |

# Date Object “to” Methods

```
var now = new Date ("November 25,2009");
```

| Name           | Example              | Returned value                                                                    |
|----------------|----------------------|-----------------------------------------------------------------------------------|
| toUTCString    | now.toUTCString()    | Tue, 24 Nov 2009 22:00:00 GMT<br>//used instead of the deprecated<br>toGMTSting() |
| toLocaleString | now.toLocaleString() | 11/25/2009, 12:00:00 AM                                                           |
| toString       | now.toString()       | Wed Nov 25 2009 00:00:00<br>GMT+0200 (Egypt Standard Time)                        |

# Date Object

- Hours should be specified using a **24-hour** clock.
- The **month** is always indexed from **zero**, so that November is month 10.
- The year can also be offset by 1900, so that you can use either of these two forms

```
var NovDate = new Date(90, 10, 23);
var NovDate = new Date(1990, 10, 23);
```

- For the year 2000 and beyond you must use the second form

```
var NovDate = new Date(2006, 10, 23);
```

- This form may optionally take an additional three integer arguments for the time, so that 1:05 PM on November 23, 1990 is

```
var NovDate2 = new Date(90, 10, 23, 13, 5, 0);
```

# Boolean Object

- The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).
- Everything in the language is either “truthy” or “falsy”
- The rules for truthiness:
  - ▷ 0 , "" , NaN , null , and undefined → falsy
  - ▷ Everything else → truthy
- You can convert any value to its boolean equivalent by applying “!!” preceding the value
  - Example:
    - !!"" → false
    - !!123 → true
- To create Boolean Object
  - ▷ var b = new Boolean(); → false // typeof is Object
  - ▷ B = false → false // typeof “boolean”

# Boolean Object

- All the following lines of code create Boolean objects with an initial value of **false**:

```
var myBoolean=new Boolean()
var myBoolean=new Boolean(0)
var myBoolean=new Boolean(null)
var myBoolean=new Boolean(undefined)
var myBoolean=new Boolean("")
var myBoolean=new Boolean(false)
var myBoolean=new Boolean(NaN)
```

- And all the following lines of code create Boolean objects with an initial value of **true**:

```
var myBoolean=new Boolean(true)
var myBoolean=new Boolean(1)
var myBoolean=new Boolean("false")
var myBoolean=new Boolean("anyThing")
```

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





# *Assignment*