Introduction to Java Script

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the `new` keyword)
- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

3.14 is a primitive value

A **primitive data type** is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

Examples

- `string`
- `number`
- `boolean`
- `null`
- `undefined`
- `symbol`
- `bigint`

Immutable

Primitive values are immutable (they are hardcoded and cannot be changed).

if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

Objects are Variables

JavaScript variables can contain single values:

Example

```
let person = "John Doe";
```

JavaScript variables can also contain many values.

Objects are variables too. But objects can contain many values.

Object values are written as **name : value** pairs (name and value separated by a colon).

Example

```
let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

A JavaScript object is a collection of **named values**

It is a common practice to declare objects with the `const` keyword.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

Object Properties

The named values, in JavaScript objects, are called **properties**.

Property	Value
firstName	John

lastName	Doe
age	50
eyeColor	blue

Objects written as name value pairs are similar to:

- Associative arrays in PHP
- Dictionaries in Python
- Hash tables in C
- Hash maps in Java
- Hashes in Ruby and Perl

Object Methods

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

Property	**Value**
firstName	John
lastName	Doe

age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript objects are containers for named values, called properties and methods.

You will learn more about methods in the next chapters.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword `new`.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using `Object.create()`.

Using an Object Literal

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

Spaces and line breaks are not important. An object definition can span multiple lines:

This example creates an empty JavaScript object, and then adds 4 properties:

Example

```
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Using the JavaScript Keyword new

The following example create a new JavaScript object using `new Object()`, and then adds 4 properties:

Example

```
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

The examples above do exactly the same.

But there is no need to use `new Object()`.

For readability, simplicity and execution speed, use the object literal method.

JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

```
const x = person;  // Will not create a copy of person.
```

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Any changes to x will also change person, because x and person are the same object.

Example

```
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}

const x = person;
x.age = 10;      // Will change both x.age and person.age
```

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element

change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads i
resize	onresize	When the visitor resizes the window of the browser

Let's discuss some examples over events and their handlers.

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
    function clickevent()
    {
        document.write("This is JavaTpoint");
    }
    //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
    function mouseoverevent()
    {
        alert("This is JavaTpoint");
    }
    //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

Focus Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
    function focusevent()
    {
        document.getElementById("input1").style.background=" aqua";
    }
//-->
</script>
```

```
</body>
</html>
```

Keydown Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
    function keydownevent()
    {
        document.getElementById("input1");
        alert("Pressed a key");
    }
//-->
</script>
</body>
</html>
```

Load event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
document.write("The page is loaded successfully");
//-->
</script>
</body>
```

11. **</html>**

JavaScript Strings

Previous Page

Next Page

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

Syntax

Use the following syntax to create a String object −

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded.

String Properties

Here is a list of the properties of String object and their description.

Sr.No.	Property & Description
1	constructor Returns a reference to the String function that created the object.
2	length Returns the length of the string.
3	prototype The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to demonstrate the usage of String properties.

String Methods

Here is a list of the methods available in String object along with their description.

Sr.No.	Method & Description
1	charAt() Returns the character at the specified index.
2	charCodeAt() Returns a number indicating the Unicode value of the character at the given index.
3	concat() Combines the text of two strings and returns a new string.
4	indexOf() Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	lastIndexOf() Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	localeCompare() Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	match() Used to match a regular expression against a string.
8	replace()

	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	search() Executes the search for a match between a regular expression and a specified string.
10	slice() Extracts a section of a string and returns a new string.
11	split() Splits a String object into an array of strings by separating the string into substrings.
12	substr() Returns the characters in a string beginning at the specified location through the specified number of characters.
13	substring() Returns the characters in a string between two indexes into the string.
14	toLocaleLowerCase() The characters within a string are converted to lower case while respecting the current locale.
15	toLocaleUpperCase() The characters within a string are converted to upper case while respecting the current locale.
16	toLowerCase() Returns the calling string value converted to lower case.

17	toString() Returns a string representing the specified object.
18	toUpperCase() Returns the calling string value converted to uppercase.
19	valueOf() Returns the primitive value of the specified object.

String HTML Wrappers

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

Sr.No.	Method & Description
1	anchor() Creates an HTML anchor that is used as a hypertext target.
2	big() Creates a string to be displayed in a big font as if it were in a <big> tag.
3	blink() Creates a string to blink as if it were in a <blink> tag.
4	bold() Creates a string to be displayed as bold as if it were in a <b> tag.
5	fixed() Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag

6	fontcolor() Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag.
7	fontsize() Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.
8	italics() Causes a string to be italic, as if it were in an <i> tag.
9	link() Creates an HTML hypertext link that requests another URL.
10	small() Causes a string to be displayed in a small font, as if it were in a <small> tag.
11	strike() Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
12	sub() Causes a string to be displayed as a subscript, as if it were in a <sub> tag
13	sup() Causes a string to be displayed as a superscript, as if it were in a <sup> tag

In the following sections, we will have a few examples to demonstrate the usage of String methods.

JavaScript Conditions

Example

Execute a block of code based on user input:

```
var text;
var fruits = document.getElementById("myInput").value;

switch(fruits) {
  case "Banana":
    text = "Banana is good!";
    break;
  case "Orange":
    text = "I am not a fan of orange.";
    break;
  case "Apple":
    text = "How you like them apples?";
    break;
  default:
    text = "I have never heard of that fruit...";
}
```

More "Try it Yourself" examples below.

Definition

The switch statement executes a block of code depending on different cases.

The switch statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions. Use switch to select one of many blocks of code to be executed. This is the perfect solution for long, nested if/else statements.

The switch statement evaluates an expression. The value of the expression is then compared with the values of each case in the structure. If there is a match, the associated block of code is executed.

The switch statement is often used together with a break or a default keyword (or both). These are both optional:

The **break** keyword breaks out of the switch block. This will stop the execution of more execution of code and/or case testing inside the block. If break is omitted, the next code block in the switch statement is executed.

The **default** keyword specifies some code to run if there is no case match. There can only be one default keyword in a switch. Although this is optional, it is recommended that you use it, as it takes care of unexpected cases.

Syntax

```
switch(expression) {
  case n:
    code block
    break;
  case n:
    code block
    break;
  default:
    default code block
}
```

Parameter Values

Parameter	Description
expression	Required. Specifies an expression to be evaluated. The expression is evaluated once. The value of the expression is compared with the values of each case labels in the structure. If there is a match, the associated block of code is executed

More Examples

Example

Use today's weekday number to calculate the weekday name (Sunday=0, Monday=1, Tuesday=2, ...):

```
var day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
    break;
  default:
    day = "Unknown Day";
}
```