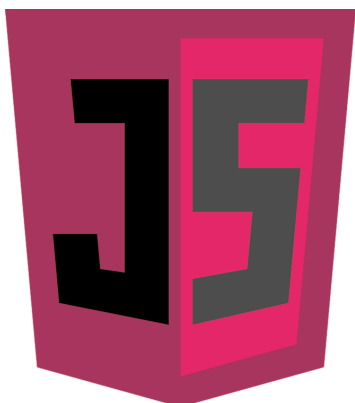


JavaScript Syntax Basics

In this lesson, we'll cover some JavaScript basics.
Let's begin!

WE'LL COVER THE FOLLOWING ^

- Identifiers
 - Examples:
- Strict mode
 - Examples:
- Reserved words
 - Examples:
- Comments
- Naming conventions
 - Examples:
 - Examples:
- Statements



Syntax Basics

à 1/4 1/4 1/4 1/4 1/4

Like every language, JavaScript has its own syntax rules.

It is time to dive into details.

As you have seen in the samples of this course, JavaScript is one of the “*curly-brace languages*”, its syntax is similar to C, C++, C#, and Java.

At the time of this writing, most browsers use the fifth edition of the **ECMAScript standard** (*Ecma-262, Edition 5.1/June, 2011*), so all syntax rules treated here are based on this standard. Any time JavaScript is mentioned in the context of syntax or semantic rules, the ECMAScript standard is used.

JavaScript is **case-sensitive**, so every token including keywords, function names, variables, object names, and operators **make a distinction between lowercase and uppercase letters**.

For example, the new operator (“`new`” is a keyword) is different from `New` which may be an identifier, just as `myNumber` is a totally different variable name than `Mynumber`.

When the JavaScript engine parses the script, it first **scans for smaller elements** like keywords, identifiers, operators, comments, literals, etc. To understand the rules of the languages, it is worth getting to know the syntax of these smaller units.

Identifiers

Identifiers are used all-around in scripts to name variables, functions, objects, properties, or function arguments.

An identifier may be one or more characters where the first character must be a `letter`, a dollar sign (`$`), or an underscore (`_`), and all other characters may be letters, numbers, dollar signs, or underscores.

Every letter character that can be used is considered in **Unicode** as letters, including Central European characters with accents such as “ü” or “é”, as well as other letters, including the Hebrew alef or the Greek gamma.


Examples: #

Examples

The following strings are all valid identifiers:

```
getMyObject  
$_this  
Car  
_transfer_24
```



 **NOTE:** The standard allows using other Unicode characters or even Unicode escape sequences to be used in identifiers, for example, `objü\u2345` is a valid identifier (`\u2345` is a Unicode escape sequence). However, whenever possible, use only the letters of the English alphabet.

The language defines reserved words that cannot be used as identifiers. The reason for this restriction is that allowing reserved words to be used as identifiers would make the language parsing complex and could lead to ambiguity.

 Show Useful Info

Strict mode

The fifth version of ECMAScript introduced a new concept, strict mode, which seems a bit weird at first sight.

This mode addresses some erratic behavior of the third ECMAScript edition, and it uses a different parsing and execution mode to avoid a few activities that are declared to be unsafe.

For example,

strict mode does not allow you to define new variables without the var keyword, and also defines additional reserved words that are not allowed to be used as identifiers.

You can enable strict mode for an entire script (a `.js` file or a `<script>` section

in HTML) by adding this line to the top:

```
"use strict";
```

To turn on strict mode for a single function, add the same line to the **top of the function body**:

Examples: #

```
function solveMyProblem(problem) {  
  "use strict";  
  // Put the function body here  
}
```

At first sight, this switch looks like a string that has not been assigned to any variable, but the JavaScript engine takes it into account as a *pragma to turn on strict mode*. This syntax was chosen to keep compatibility with the third edition of ECMAScript.

Reserved words

The standard defines a number of reserved words that include keywords, words reserved for future use, the null literal, true, and false (Boolean literals).

The **fifth ECMAScript edition** defines these keywords:

break	if
in	instanceof
new	return
switch	this
throw	try
typeof	var
void	while
with	case
catch	continue
debugger	default
delete	do
else	finally
for	function

A few other words are used as **keywords in proposed extensions**, and are therefore reserved to allow for the possibility of future adoption of those extensions:

```
class
const
enum
export
extends
import
super
```



Well, strict mode extends this list with the following **keywords**:

```
implements
interface
let
package
private
protected
public
static
yield
```



Although you cannot use reserved words as identifiers generally, the standard allows one **exception**. You can use them in property names.

Examples:

Let's see an example:

```
var myCar = new Object();
myCar.export = true;
```



The export word is in the list of reserved words however, it is still allowed as an object property name. But, you cannot do this:

```
var export = 23;
```



In this context, export is used as a variable name, and this scenario is declared **invalid** by the standard.

Comments

Just as in many curly-brace languages, you can use single line comments, or block comments in JavaScript. Single line comments start with two forward-slash characters (`//`), and all characters until the end of the line are the part of the comment:

```
var x = 1; // This is a single line comment
```



Block comments start with a forward-slash immediately followed by an asterisk (`/*`), end with the opposite (`*/`), and may contain multiple lines:

```
/*  
  This comment spreads  
  multiple lines  
*/  
var x = 1;
```



Block comments cannot be nested. The highlighted part of this snippet shows what is considered a block comment:

```
/*  
  This comment spreads  
  multiple lines  
/* and uses a wrong */  
  nested block comment  
*/  
var x = 1;
```



Naming conventions

APIs defined by the HTML and ECMAScript standards and most JavaScript programmers follow this convention:

◆ **Identifiers** in JavaScript use camel case. It means that words in the name are written with lowercase letters, the very first letter is lowercase, and each additional word is offset by a capital letter.

◆ **Constructor** functions which must be used with the new operator should start with a capital letter. JavaScript issues no warning if a

required new is omitted. Because odd things can happen if new is not used, this capitalization convention is indispensable.

Examples:

```
Car  
People  
Connection
```



Variables and object properties that represent constant values are written with all capital letters, words are separated by underscore.

Examples:

```
PI  
MY_CONSTANT
```



Statements

JavaScript is an imperative programming language, it describes the body of a program with function declarations and statements. Statements can be

- expressions
- variable declarations with optional initialization
- the debugger statement
- flow-control statements

Statements can be single statement, or organized into blocks that are wrapped with curly-braces ({ and } characters):

```
// This is a single statement  
car.doSomethingWithMe();
```



```
// This is a statement block  
if (car.wheels >= 4) {  
    var engine = new Object();  
}
```

```
var cargo = new Object();  
cargo.isLarge = true;  
}
```

As you can see, statements are terminated with semicolons. If you omit the semicolon, the parser determines the end of the statement:

```
var myExpr = 12 * (a + b)  
var myVal = myExpr * 2;
```



In this short code snippet, the first variable declaration statement is not terminated with a semicolon, but from context, the parser decided that the statement should be terminated at the end of the first line.

 Show Useful Info

By now, you possess all the things you need to know to learn more exciting topics. Let's plunge into declaring variables and using types in the *next lesson*.