

**For everyone who contributed to
the publication of this book**

Table of Contents

Java Programming Language	1
Java Script Programming Language	103
jQuery Programming Language	389
JSON Programming Language	457
Reference	488

Introduction to Book

Java is an object-oriented, cross platform, multi-purpose programming language produced by Sun Microsystems. First released in 1995, it was developed to be a machine independent web technology. It was based on C and C++ syntax to make it easy for programmers from those communities to learn. Since then, it has earned a prominent place in the world of computer programming.

Java has many characteristics that have contributed to its popularity:

- **Platform independence** - Many languages are compatible with only one platform. Java was specifically designed so that it would run on any computer, regardless if it was running Windows, Linux, Mac, Unix or any of the other operating systems.
- **Simple and easy to use** - Java's creators tried to design it so code could be written efficiently and easily.
- **Multi-functional** - Java can produce many applications from command-line programs to applets to Swing windows (basically, sophisticated graphical user interfaces).

Java does have some drawbacks. Since it has automated garbage collection, it can tend to use more memory than other similar languages. There are often implementation differences on different platforms, which have led to Java being described as a "write once, test everywhere" system. Lastly, since it uses an abstract "virtual machine", a generic Java program doesn't have access to the Native API's on a system directly. None of these issues are fatal, but it can mean that Java isn't an appropriate choice for a particular piece of software.

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage livelier and more interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.

jQuery is an open source JavaScript library that simplifies the interactions between an **HTML/CSS** document, or more precisely the Document Object Model (**DOM**), and JavaScript.

Introduction to Book

Elaborating the terms, **jQuery** simplifies **HTML** document traversing and manipulation, browser event handling, **DOM** animations, **Ajax** interactions, and cross-browser JavaScript development.

Some of the key points which supports the answer for why to use jQuery:

- It is incredibly popular, which is to say it has a large community of users and a healthy amount of contributors who participate as developers and evangelists.
- It normalizes the differences between web browsers so that you don't have to.
- It is intentionally a lightweight footprint with a simple yet clever plugin architecture.
- Its repository of plugins is vast and has seen steady growth since jQuery's release.
- Its API is fully documented, including inline code examples, which in the world of JavaScript libraries is a luxury. Heck, any documentation at all was a luxury for years.
- It is friendly, which is to say it provides helpful ways to avoid conflicts with other JavaScript libraries.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard **ECMA-262 3rd Edition - December 1999**. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

In this book, the basics for each of the three languages will be explained: **java, java Script, jQuery**, and **JSON**. This book will benefit everyone who wants to get to know and learn these languages, which can also benefit everyone who wants to enter the world of programming.

Java Programming Language

Java Programming Language

1

JAVA INTRODUCTION

What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than 3 billion devices run Java.

It is used for:

- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

Why Use Java?

- **Java** works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming languages in the world
- It is easy to learn and simple to use
- It is **open-source** and **free**
- It is secure, fast and powerful
- It has a huge community support (tens of millions of developers)
- Java is an object-oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to **C++** and **C#**, it makes it easy for programmers to switch to Java or vice versa

Java Programming Language

2

JAVA GET STARTED

• Java Install

Some PCs might have Java already installed.

To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

```
C:\Users\Your Name>java -version
```

If **Java** is installed, you will see something like this (depending on version):

```
java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)
```

If you do not have Java installed on your computer, you can download it for free from oracle.com.

Note: In this tutorial, we will write Java code in a text editor. However, it is possible to write Java in an Integrated Development Environment, such as **IntelliJ IDEA**, **Netbeans** or **Eclipse**, which are particularly useful when managing larger collections of Java files.

Setup for Windows

To install Java on Windows:

1. Go to "**System Properties**" (Can be found on Control Panel > System and Security > System > Advanced System Settings)
2. Click on the "**Environment variables**" button under the "**Advanced**" tab
3. Then, select the "**Path**" variable in System variables and click on the "**Edit**" button
4. Click on the "**New**" button and add the path where Java is installed, followed by **\bin**.

By default, Java is installed in **C:\Program Files\Java\jdk-11.0.1** (If nothing else was specified when you installed it). In that case, you will have to add a new path with:

C:\Program Files\Java\jdk-11.0.1\bin. Then, click "**OK**", and save the settings

Java Programming Language

3

- At last, open Command Prompt (**cmd.exe**) and type `java -version` to see if Java is running on your machine

Java Quickstart

In **Java**, every application begins with a class name, and that class must match the filename.

Let's create our first Java file, called **MyClass.java**, which can be done in any text editor (like **Notepad**).

The file should contain a "Hello World" message, which is written with the following code:

MyClass.java

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

```
C:\Users\Name>java MyClass  
Hello World
```

Don't worry if you don't understand the code above - we will discuss it in detail in later chapters. For now, focus on how to run the code above.

Save the code in Notepad as "**MyClass.java**". Open Command Prompt (cmd.exe), navigate to the directory where you saved your file, and type "**javac MyClass.java**":

```
C:\Users\Your Name>javac MyClass.java
```

This will compile your code. If there are no errors in the code, the command prompt will take you to the next line. Now, type "**java MyClass**" to run the file:

```
C:\Users\Your Name>java MyClass
```

The output should read:

Java Programming Language

4

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

```
C:\Users\Name\java MyClass  
Hello World
```

Example explained

Every line of code that runs in Java must be inside a class. In our example, we named the class **MyClass**. A class should always start with an uppercase first letter.

Note: Java is case-sensitive: "**MyClass**" and "**myclass**" has different meaning.

The name of the java file must match the class name. When saving the file, save it using the class name and add ".**java**" to the end of the filename. To run the example above on your computer, make sure that Java is properly installed

• Java Syntax

The main Method

The **main()** method is required and you will see it in every Java program:

```
public static void main(String[] args)
```

Any code inside the **main()** method will be executed. You don't have to understand the keywords before and after **main**.

For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the **main()** method.

System.out.println()

Inside the **main()** method, we can use the **println()** method to print a line of text to the screen:

Java Programming Language

5

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

```
C:\Users\Name\java MyClass  
Hello World
```

Note: The curly braces {} marks the beginning and the end of a block of code.

Note: Each code statement must end with a semicolon.

• Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        // This is a comment  
        System.out.println("Hello World");  
    }  
}
```

```
Hello World
```

This example uses a single-line comment at the end of a line of code:

Example

Java Programming Language

6

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello World"); // This is a comment  
    }  
}
```

Hello World

Java Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by Java.

This example uses a **multi-line comment** (a comment block) to explain the code:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        /* The code below will print the words Hello World  
         * to the screen, and it is amazing */  
        System.out.println("Hello World");  
    }  
}
```

Hello World

Single or multi-line comments?

It is up to you which you want to use. Normally, we use `//` for short comments, and `/*`
`*/` for longer.

• Java Variables

Variables are containers for storing data values.

In Java, there are different types of variables, for example:

- **String** - stores text, such as "Hello". String values are surrounded by double quotes
- **int** - stores integers (whole numbers), without decimals, such as 123 or -123
- **float** - stores floating point numbers, with decimals, such as 19.99 or -19.99

Java Programming Language

7

- **char** - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- **boolean** - stores values with two states: true or false

Declaring (Creating) Variables

To create a **variable**, you must specify the type and assign it a value:

Syntax

type variable = value;

Where type is one of Java's types (such as **int** or **String**), and variable is the name of the variable (such as **x** or **name**). The equal sign is used to assign values to the variable.

To create a variable that should store text, look at the following example:

Example

Create a variable called name of type String and assign it the value "John":

```
public class MyClass {  
    public static void main(String[] args) {  
        String name = "John";  
        System.out.println(name);  
    }  
}
```

John

To create a variable that should store a number, look at the following example:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int myNum = 15;  
        System.out.println(myNum);  
    }  
}
```

Java Programming Language

8

15

You can also declare a variable without assigning the value, and assign the value later:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int myNum;  
        myNum = 15;  
        System.out.println(myNum);  
    }  
}
```

15

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int myNum = 15;  
        myNum = 20; // myNum is now 20  
        System.out.println(myNum);  
    }  
}
```

20

Final Variables

However, you can add the final keyword if you don't want others (or yourself) to overwrite existing values (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

Example

Java Programming Language

9

```
public class MyClass {  
    public static void main(String[] args) {  
        final int myNum = 15;  
        myNum = 20; // will generate an error  
        System.out.println(myNum);  
    }  
}
```

```
MyClass.java:4: error: cannot assign a value to final variable myNum  
    myNum = 20;  
           ^  
1 error
```

Other Types

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;  
float myFloatNum = 5.99f;  
char myLetter = 'D';  
boolean myBool = true;  
String myText = "Hello";
```

Display Variables

The **println()** method is often used to display variables.

To combine both text and a variable, use the **+** character:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String name = "John";  
        System.out.println("Hello " + name);  
    }  
}
```

```
Hello John
```

You can also use the **+** character to add a variable to another variable:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String firstName = "John ";  
        String lastName = "Doe";  
        String fullName = firstName + lastName;  
        System.out.println(fullName);  
    }  
}
```

John Doe

For numeric values, the `+` character works as a mathematical operator (notice that we use `int` (integer) variables here):

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 5;  
        int y = 6;  
        System.out.println(x + y); // Print the value of x + y  
    }  
}
```

11

From the example above, you can expect:

- `x` stores the value 5
- `y` stores the value 6
- Then we use the `println()` method to display the value of `x + y`, which is 11

Declare Many Variables

To declare more than one variable of the same type, use a comma-separated list:

Example

Java Programming Language

11

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 5, y = 6, z = 50;  
        System.out.println(x + y + z);  
    }  
}
```

61

Java Identifiers

All **Java variables** must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        // Good  
        int minutesPerHour = 60;  
  
        // OK, but not so easy to understand what m actually is  
        int m = 60;  
  
        System.out.println(minutesPerHour);  
        System.out.println(m);  
    }  
}
```

60
60

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs

Java Programming Language

12

- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with \$ and _ (but we will not use it in this tutorial)
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as **int** or **boolean**) cannot be used as names

• Java Data Types

As explained in the previous chapter, a variable in Java must be a specified data type:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int myNum = 5;                      // integer (whole number)  
        float myFloatNum = 5.99f;           // floating point number  
        char myLetter = 'D';                // character  
        boolean myBool = true;              // boolean  
        String myText = "Hello";            // String  
        System.out.println(myNum);  
        System.out.println(myFloatNum);  
        System.out.println(myLetter);  
        System.out.println(myBool);  
        System.out.println(myText);  
    }  
}
```

```
5  
5.99  
D  
true  
Hello
```

Data types are divided into two groups:

- Primitive data types** - includes byte, short, int, long, float, double, boolean and char
- Non-primitive data types** - such as String, Arrays and Classes

Primitive Data Types

Java Programming Language

13

A **primitive data type** specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Numbers

Primitive number types are divided into two groups:

Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are byte, short, int and long. Which type you should use, depends on the numeric value.

Floating point types represents numbers with a fractional part, containing one or more decimals. There are two types: float and double.

Integer Types

Byte

The byte data type can store whole numbers from -128 to 127. This can be used instead of int or other integer types to save memory when you are certain that the value will be within -128 and 127:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        byte myNum = 100;  
        System.out.println(myNum);  
    }  
}
```

100

Short

The **short data type** can store whole numbers from -32768 to 32767:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        short myNum = 5000;  
        System.out.println(myNum);  
    }  
}
```

5000

Int

The **int data type** can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the int data type is the preferred data type when we create variables with a numeric value.

Example

Java Programming Language

15

```
public class MyClass {  
    public static void main(String[] args) {  
        int myNum = 100000;  
        System.out.println(myNum);  
    }  
}
```

```
100000
```

Long

The **long data type** can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        long myNum = 15000000000L;  
        System.out.println(myNum);  
    }  
}
```

```
15000000000
```

Floating Point Types

You should use a **floating-point** type whenever you need a number with a decimal, such as 9.99 or 3.14515.

Float

The **float data type** can store fractional numbers from 3.4e-038 to 3.4e+038. Note that you should end the value with an "f":

Example

Java Programming Language

16

```
public class MyClass {  
    public static void main(String[] args) {  
        float myNum = 5.75f;  
        System.out.println(myNum);  
    }  
}
```

5.75

Double

The **double data type** can store fractional numbers from $1.7e-308$ to $1.7e+308$. Note that you should end the value with a "d":

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        double myNum = 19.99d;  
        System.out.println(myNum);  
    }  
}
```

19.99

Use float or double?

The precision of a **floating-point** value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore, it is safer to use double for most calculations.

Scientific Numbers

A **floating-point** number can also be a scientific number with an "e" to indicate the power of 10:

Example

Java Programming Language

17

```
public class MyClass {  
    public static void main(String[] args) {  
        float f1 = 35e3f;  
        double d1 = 12E4d;  
        System.out.println(f1);  
        System.out.println(d1);  
    }  
}
```

```
35000.0  
120000.0
```

Booleans

A **Boolean data type** is declared with the Boolean keyword and can only take the values true or false:

```
public class MyClass {  
    public static void main(String[] args) {  
        boolean isJavaFun = true;  
        boolean isFishTasty = false;  
        System.out.println(isJavaFun);  
        System.out.println(isFishTasty);  
    }  
}
```

```
true  
false
```

Characters

The **char data type** is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        char myGrade = 'B';  
        System.out.println(myGrade);  
    }  
}
```

Java Programming Language

18

B

Alternatively, you can use **ASCII values** to display certain characters:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        char a = 65, b = 66, c = 67;  
        System.out.println(a);  
        System.out.println(b);  
        System.out.println(c);  
    }  
}
```

A
B
C

Strings

The **String data type** is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String greeting = "Hello World";  
        System.out.println(greeting);  
    }  
}
```

Hello World

The **String type** is so much used and integrated in Java, that some call it "the special **ninth type**".

A String in Java is actually a **non-primitive** data type, because it refers to an object. The String object has methods that are used to perform certain operations on strings. **Don't worry if you don't understand the term "object" just yet.**

Non-Primitive Data Types

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- **Non-primitive** types can be used to call methods to perform certain operations, while primitive types cannot.
- A **primitive** type has always a value, while non-primitive types can be null.
- A **primitive** type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a **primitive** type depends on the data type, while non-primitive types have all the same size.

• Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double
- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

Example

Java Programming Language

20

```
public class MyClass {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt; // Automatic casting: int to double  
  
        System.out.println(myInt);  
        System.out.println(myDouble);  
    }  
}
```

```
9  
9.0
```

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        double myDouble = 9.78;  
        int myInt = (int) myDouble; // Explicit casting: double to int  
  
        System.out.println(myDouble);  
        System.out.println(myInt);  
    }  
}
```

```
9.78  
9
```

• Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the **+** operator to add together two values:

Example

Java Programming Language

21

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 100 + 50;  
        System.out.println(x);  
    }  
}
```

150

Although the `+` operator is often used to add together two values, like in the example above, it can also be used to **add** together a variable and a value, or a variable and another variable:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int sum1 = 100 + 50;  
        int sum2 = sum1 + 250;  
        int sum3 = sum2 + sum1;  
        System.out.println(sum1);  
        System.out.println(sum2);  
        System.out.println(sum3);  
    }  
}
```

150

400

800

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Java Programming Language

22

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the assignment operator (`=`) to assign the value 10 to a variable called `x`:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 10;  
        System.out.println(x);  
    }  
}
```

10

The **addition assignment** operator (`+=`) adds a value to a variable:

Example

Java Programming Language

23

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 10;  
        x += 5;  
        System.out.println(x);  
    }  
}
```

15

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Comparison Operators

Comparison operators are used to compare two values:

Java Programming Language

24

Operator	Name	Example
<code>==</code>	Equal to	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>></code>	Greater than	<code>x > y</code>
<code><</code>	Less than	<code>x < y</code>
<code>>=</code>	Greater than or equal to	<code>x >= y</code>
<code><=</code>	Less than or equal to	<code>x <= y</code>

Java Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
<code>&&</code>	Logical and	Returns true if both statements are true	<code>x < 5 && x < 10</code>
<code> </code>	Logical or	Returns true if one of the statements is true	<code>x < 5 x < 4</code>
<code>!</code>	Logical not	Reverse the result, returns false if the result is true	<code>!(x < 5 && x < 10)</code>

• Java Strings

Strings are used for storing text.

A **String variable** contains a collection of characters surrounded by double quotes:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String greeting = "Hello";  
        System.out.println(greeting);  
    }  
}
```

Hello

String Length

Java Programming Language

25

A **String** in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the **length()** method:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
        System.out.println("The length of the txt string is: " + txt.length());  
    }  
}
```

The length of the txt string is: 26

More String Methods

There are many string methods available, for example **toUpperCase()** and **toLowerCase()**:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String txt = "Hello World";  
        System.out.println(txt.toUpperCase());  
        System.out.println(txt.toLowerCase());  
    }  
}
```

HELLO WORLD
hello world

Finding a Character in a String

The **indexOf()** method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace):

Example

Java Programming Language

26

```
public class MyClass {  
    public static void main(String[] args) {  
        String txt = "Please locate where 'locate' occurs!";  
        System.out.println(txt.indexOf("locate"));  
    }  
}
```

7

Java counts positions from zero.

0 is the first position in a string, 1 is the second, 2 is the third ...

String Concatenation

The `+` operator can be used between strings to combine them. This is called **concatenation**:

Example

```
public class MyClass {  
    public static void main(String args[]) {  
        String firstName = "John";  
        String lastName = "Doe";  
        System.out.println(firstName + " " + lastName);  
    }  
}
```

John Doe

Note that we have added an empty text (" ") to create a space between `firstName` and `lastName` on print.

You can also use the `concat()` method to concatenate two strings:

Example

Java Programming Language

27

```
public class MyClass {  
    public static void main(String[] args) {  
        String firstName = "John ";  
        String lastName = "Doe";  
        System.out.println(firstName.concat(lastName));  
    }  
}
```

John Doe

Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

```
String txt = "We are the so-called "Vikings" from the north.;"
```

The solution to avoid this problem, is to use the backslash escape character.

The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String txt = "We are the so-called \"Vikings\" from the north.;"  
        System.out.println(txt);  
    }  
}
```

We are the so-called "Vikings" from the north.

Java Programming Language

28

The sequence `\'` inserts a single quote in a string:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String txt = "It\'s alright."  
        System.out.println(txt);  
    }  
}
```

It's alright.

The sequence `\|` inserts a single backslash in a string:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String txt = "The character \\ is called backslash."  
        System.out.println(txt);  
    }  
}
```

The character \ is called backslash.

Six other escape sequences are valid in Java:

Code	Result
<code>\n</code>	New Line
<code>\r</code>	Carriage Return
<code>\t</code>	Tab
<code>\b</code>	Backspace
<code>\f</code>	Form Feed

Adding Numbers and Strings

WARNING!

Java uses the `+` operator for both addition and concatenation.

Java Programming Language

29

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 10;  
        int y = 20;  
        int z = x + y;  
        System.out.println(z);  
    }  
}
```

30

If you add two strings, the result will be a string concatenation:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String x = "10";  
        String y = "20";  
        String z = x + y;  
        System.out.println(z);  
    }  
}
```

1020

If you add a number and a string, the result will be a string concatenation:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String x = "10";  
        int y = 20;  
        String z = x + y;  
        System.out.println(z);  
    }  
}
```

1020

• Java Math

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

Math.max(x,y)

The **Math.max(x,y)** method can be used to find the highest value of x and y:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(Math.max(5, 10));  
    }  
}
```

10

Math.min(x,y)

The **Math.min(x,y)** method can be used to find the lowest value of of x and y:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(Math.min(5, 10));  
    }  
}
```

5

Math.sqrt(x)

The **Math.sqrt(x)** method returns the square root of x:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(Math.sqrt(64));  
    }  
}
```

8.0

Math.abs(x)

The **Math.abs(x)** method returns the absolute (positive) value of x:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(Math.abs(-4.7));  
    }  
}
```

4.7

Random Numbers

Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive):

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(Math.random());  
    }  
}
```

0.08722622676625247

To get more control over the random number, e.g. you only want a random number between 0 and 100, you can use the following formula:

Example

Java Programming Language

32

```
public class MyClass {  
    public static void main(String[] args) {  
        int randomNum = (int)(Math.random() * 101); // 0 to 100  
        System.out.println(randomNum);  
    }  
}
```

84

• Java Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, Java has a Boolean data type, which can take the values **true** or **false**

Boolean Values

A **Boolean** type is declared with the **Boolean** keyword and can only take the values **true** or **false**:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        boolean isJavaFun = true;  
        boolean isFishTasty = false;  
        System.out.println(isJavaFun);  
        System.out.println(isFishTasty);  
    }  
}
```

true
false

Boolean Expression

A **Boolean expression** is a Java expression that returns a Boolean value: true or false.

You can use a comparison operator, such as the greater than ($>$) operator to find out if an expression (or a variable) is true:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 10;  
        int y = 9;  
        System.out.println(x > y); // returns true, because 10 is higher than 9  
    }  
}
```

true

Or even easier:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(10 > 9); // returns true, because 10 is higher than 9  
    }  
}
```

true

In the examples below, we use the equal to ($==$) operator to evaluate an expression:

Example

Java Programming Language

34

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 10;  
        System.out.println(x == 10); // returns true, because the value of x is equal to 10  
    }  
}
```

true

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println(15 == 10); // returns false, because 10 is not equal to 15  
    }  
}
```

false

- **Java If ... Else**

Java Conditions and If Statements

Java supports the usual logical conditions from mathematics:

- Less than: $a < b$
- Less than or equal to: $a \leq b$
- Greater than: $a > b$
- Greater than or equal to: $a \geq b$
- Equal to $a == b$
- Not Equal to: $a != b$

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The if Statement

Use the **if statement** to specify a block of Java code to be executed if a condition is true.

Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

Note that **if** is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        if (20 > 18) {  
            System.out.println("20 is greater than 18"); // obviously  
        }  
    }  
}
```

```
20 is greater than 18
```

We can also test variables:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int x = 20;  
        int y = 18;  
        if (x > y) {  
            System.out.println("x is greater than y");  
        }  
    }  
}
```

```
x is greater than y
```

Example explained

In the example above we use two variables, x and y, to test whether x is greater than y (using the `>` operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

The else Statement

Use the **else statement** to specify a block of code to be executed if the condition is false.

Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int time = 20;  
        if (time < 18) {  
            System.out.println("Good day.");  
        } else {  
            System.out.println("Good evening.");  
        }  
    }  
}
```

```
Good evening.
```

Example explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

The else if Statement

Use the **else if statement** to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false and condition2 is false  
}
```

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int time = 22;  
        if (time < 10) {  
            System.out.println("Good morning.");  
        } else if (time < 20) {  
            System.out.println("Good day.");  
        } else {  
            System.out.println("Good evening.");  
        }  
    }  
}
```

```
Good evening.
```

Example explained

In the example above, time (22) is greater than 10, so the first condition is false. The next condition, in the else if statement, is also false, so we move on to the else condition since condition1 and condition2 is both false - and print to the screen "Good evening". However, if the time was 14, our program would print "Good day."

Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the ternary operator because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax

variable = (condition) ? expressionTrue : expressionFalse;

Instead of writing:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int time = 20;  
        if (time < 18) {  
            System.out.println("Good day.");  
        } else {  
            System.out.println("Good evening.");  
        }  
    }  
}
```

Good evening.

You can simply write:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int time = 20;  
        String result;  
        result = (time < 18) ? "Good day." : "Good evening."  
        System.out.println(result);  
    }  
}
```

Good evening.

- **Java Switch**

Java Switch Statements

Use the **switch statement** to select one of many code blocks to be executed.

Syntax

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

This is how it works:

- The **switch expression** is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and default keywords are optional, and will be described later in this chapter

The example below uses the weekday number to calculate the weekday name:

Example

Java Programming Language

40

```
public class MyClass {  
    public static void main(String[] args) {  
        int day = 4;  
        switch (day) {  
            case 1:  
                System.out.println("Monday");  
                break;  
            case 2:  
                System.out.println("Tuesday");  
                break;  
            case 3:  
                System.out.println("Wednesday");  
                break;  
            case 4:  
                System.out.println("Thursday");  
                break;  
            case 5:  
                System.out.println("Friday");  
                break;  
            case 6:  
                System.out.println("Saturday");  
                break;  
            case 7:  
                System.out.println("Sunday");  
                break;  
        }  
    }  
}
```

Thursday

The break Keyword

When **Java** reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

The **default keyword** specifies some code to run if there is no case match:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int day = 4;  
        switch (day) {  
            case 6:  
                System.out.println("Today is Saturday");  
                break;  
            case 7:  
                System.out.println("Today is Sunday");  
                break;  
            default:  
                System.out.println("Looking forward to the Weekend");  
        }  
    }  
}
```

Looking forward to the Weekend

Note that if the **default statement** is used as the last statement in a switch block, it does not need a break.

- **Java While Loop**

Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

Java While Loop

The **while loop** loops through a block of code as long as a specified condition is true:

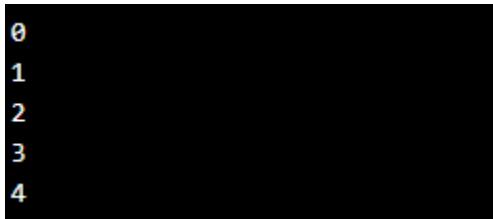
Syntax

```
while (condition) {  
    // code block to be executed  
}
```

In the example below, the code in the loop will run, over and over again, as long as a variable (**i**) is less than 5:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 5) {  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```



```
0  
1  
2  
3  
4
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

The Do/While Loop

The **do/while loop** is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {  
    // code block to be executed  
}  
while (condition);
```

Java Programming Language

43

The example below uses a **do/while loop**. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int i = 0;  
        do {  
            System.out.println(i);  
            i++;  
        }  
        while (i < 5);  
    }  
}
```

```
0  
1  
2  
3  
4
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

• Java For Loop

When you know exactly how many times you want to loop through a block of code, use the **for loop** instead of a while loop:

Syntax

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Java Programming Language

44

The example below will print the numbers 0 to 4:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

```
0  
1  
2  
3  
4
```

Example explained

Statement 1 sets a variable before the loop starts (**int i = 0**).

Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.

Statement 3 increases a value (**i++**) each time the code block in the loop has been executed.

Another Example

This example will only print even values between 0 and 10:

```
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i <= 10; i = i + 2) {  
            System.out.println(i);  
        }  
    }  
}
```

```
0  
2  
4  
6  
8  
10
```

For-Each Loop

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an array:

Syntax

```
for (type variableName : arrayName) {  
    // code block to be executed  
}
```

The following example outputs all elements in the cars array, using a "**for-each**" loop:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        for (String i : cars) {  
            System.out.println(i);  
        }  
    }  
}
```

```
Volvo  
BMW  
Ford  
Mazda
```

- **Java Break and Continue**

Java Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "**jump out**" of a switch statement.

The **break statement** can also be used to jump out of a loop.

This example jumps out of the loop when i is equal to 4:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i < 10; i++) {  
            if (i == 4) {  
                break;  
            }  
            System.out.println(i);  
        }  
    }  
}
```

```
0  
1  
2  
3
```

Java Continue

The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i < 10; i++) {  
            if (i == 4) {  
                continue;  
            }  
            System.out.println(i);  
        }  
    }  
}
```

```
0  
1  
2  
3  
5  
6  
7  
8  
9
```

Break and Continue in While Loop

You can also use **break** and **continue** in while loops:

Break Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 10) {  
            System.out.println(i);  
            i++;  
            if (i == 4) {  
                break;  
            }  
        }  
    }  
}
```

```
0  
1  
2  
3
```

Continue Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 10) {  
            if (i == 4) {  
                i++;  
                continue;  
            }  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

Java Programming Language

48

```
0  
1  
2  
3  
5  
6  
7  
8  
9
```

• Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        System.out.println(cars[0]);  
    }  
}
```

Java Programming Language

49

Volvo

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

```
cars[0] = "Opel";
```

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        cars[0] = "Opel";  
        System.out.println(cars[0]);  
    }  
}
```

Opel

Array Length

To find out how many elements an array has, use the length property:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        System.out.println(cars.length);  
    }  
}
```

4

Loop Through an Array

You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

The following example outputs all elements in the cars array:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        for (int i = 0; i < cars.length; i++) {  
            System.out.println(cars[i]);  
        }  
    }  
}
```

```
Volvo  
BMW  
Ford  
Mazda
```

Loop Through an Array with For-Each

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays:

Syntax

```
for (type variable : arrayname) {  
    ...  
}
```

The following example outputs all elements in the cars array, using a "**for-each**" loop:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        for (String i : cars) {  
            System.out.println(i);  
        }  
    }  
}
```

Java Programming Language

51

```
Volvo  
BMW  
Ford  
Mazda
```

The example above can be read like this: for each String element (called i - as in index) in cars, print out the value of i.

If you compare the for loop and **for-each loop**, you will see that the for-each method is easier to write, it does not require a counter (using the length property), and it is more readable.

Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

To create a **two-dimensional array**, add each array within its own set of curly braces:

Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of **myNumbers**:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };  
        int x = myNumbers[1][2];  
        System.out.println(x);  
    }  
}
```

7

Java Programming Language

52

We can also use a for loop inside another for loop to get the elements of a **two-dimensional array** (we still have to point to the two indexes):

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };  
        for (int i = 0; i < myNumbers.length; ++i) {  
            for(int j = 0; j < myNumbers[i].length; ++j) {  
                System.out.println(myNumbers[i][j]);  
            }  
        }  
    }  
}
```

```
1  
2  
3  
4  
5  
6  
7
```

JAVA METHODS

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as functions.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A **method** must be declared within a class. It is defined with the name of the method, followed by **parentheses ()**. Java provides some pre-defined methods, such as **System.out.println()**, but you can also create your own methods to perform certain actions:

Example

Create a method inside MyClass:

```
public class MyClass {  
    static void myMethod() {  
        // code to be executed  
    }  
}
```

Example Explained

- **myMethod()** is the name of the method
- **static** means that the method belongs to the **MyClass** class and not an object of the **MyClass** class.
- **void** means that this method does not have a return value.

Call a Method

To call a method in Java, write the method's name followed by two parentheses () and a **semicolon;**

In the following example, **myMethod()** is used to print a text (the action), when it is called:

Java Programming Language

54

Example

```
public class MyClass {  
    static void myMethod() {  
        System.out.println("I just got executed!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
    }  
}
```

I just got executed!

A method can also be called multiple times:

Example

```
public class MyClass {  
    static void myMethod() {  
        System.out.println("I just got executed!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
        myMethod();  
        myMethod();  
    }  
}
```

I just got executed!
I just got executed!
I just got executed!

- **Java Method Parameters**

Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called fname as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
public class MyClass {  
    static void myMethod(String fname) {  
        System.out.println(fname + " Refsnes");  
    }  
  
    public static void main(String[] args) {  
        myMethod("Liam");  
        myMethod("Jenny");  
        myMethod("Anja");  
    }  
}
```

```
Liam Refsnes  
Jenny Refsnes  
Anja Refsnes
```

When a parameter is passed to the method, it is called an argument. So, from the example above: **fname** is a parameter, while **Liam**, **Jenny** and **Anja** are arguments.

Multiple Parameters

You can have as many parameters as you like:

Example

```
public class MyClass {  
    static void myMethod(String fname, int age) {  
        System.out.println(fname + " is " + age);  
    }  
  
    public static void main(String[] args) {  
        myMethod("Liam", 5);  
        myMethod("Jenny", 8);  
        myMethod("Anja", 31);  
    }  
}
```

```
Liam is 5  
Jenny is 8  
Anja is 31
```

Return Values

The **void** keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

Example

```
public class MyClass {  
    static int myMethod(int x) {  
        return 5 + x;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(3));  
    }  
}
```

```
8
```

This example returns the sum of a method's two parameters:

Example

```
public class MyClass {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(5, 3));  
    }  
}
```

```
8
```

Java Programming Language

57

You can also store the result in a variable (recommended, as it is easier to read and maintain):

Example

```
public class MyClass {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        int z = myMethod(5, 3);  
        System.out.println(z);  
    }  
}
```

8

A Method with If...Else

It is common to use **if...else** statements inside methods:

Example

```
public class MyClass {  
  
    // Create a checkAge() method with an integer parameter called age  
    static void checkAge(int age) {  
  
        // If age is less than 18, print "access denied"  
        if (age < 18) {  
            System.out.println("Access denied - You are not old enough!");  
  
        // If age is greater than 18, print "access granted"  
        } else {  
            System.out.println("Access granted - You are old enough!");  
        }  
    }  
  
    public static void main(String[] args) {  
        checkAge(20); // Call the checkAge method and pass along an age of 20  
    }  
}
```

```
Access granted - You are old enough!
```

• Java Method Overloading

With method overloading, multiple methods can have the same name with different parameters.

Example

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

Consider the following example, which have two methods that add numbers of different type:

Example

```
public class MyClass {
    static int plusMethodInt(int x, int y) {
        return x + y;
    }

    static double plusMethodDouble(double x, double y) {
        return x + y;
    }

    public static void main(String[] args) {
        int myNum1 = plusMethodInt(8, 5);
        double myNum2 = plusMethodDouble(4.3, 6.26);
        System.out.println("int: " + myNum1);
        System.out.println("double: " + myNum2);
    }
}
```

```
int: 13
double: 10.559999999999999
```

Instead of defining two methods that should do the same thing, it is better to overload one.

Java Programming Language

59

In the example below, we overload the **plusMethod** method to work for both int and double:

Example

```
public class MyClass {  
    static int plusMethod(int x, int y) {  
        return x + y;  
    }  
  
    static double plusMethod(double x, double y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        int myNum1 = plusMethod(8, 5);  
        double myNum2 = plusMethod(4.3, 6.26);  
        System.out.println("int: " + myNum1);  
        System.out.println("double: " + myNum2);  
    }  
}
```

Note: Multiple methods can have the same name as long as the number and/or type of parameters are different.

JAVA CLASSES

- **Java OOP**

Java - What is OOP?

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

Java - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

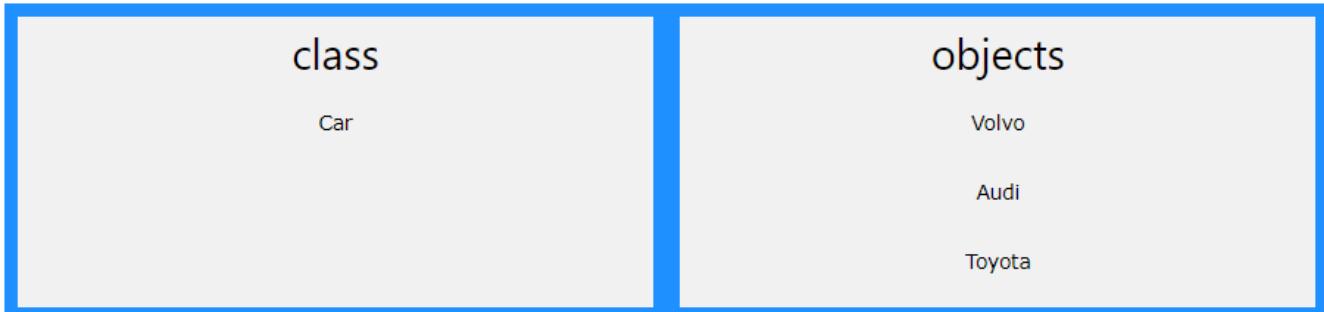
Look at the following illustration to see the difference between class and objects:



Java Programming Language

61

Another example:



So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

• Java Classes and Objects

Java is an **object-oriented programming language**.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

A **Class** is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword **class**:

MyClass.java

Create a class named "MyClass" with a variable x:

```
public class MyClass {  
    int x = 5;  
}
```

Create an Object

In Java, an object is created from a class. We have already created the class named **MyClass**, so now we can use this to create objects.

Java Programming Language

62

To create an object of **MyClass**, specify the class name, followed by the object name, and use the keyword new:

Example

Create an object called "myObj" and print the value of x:

```
public class MyClass {  
    int x = 5;  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        System.out.println(myObj.x);  
    }  
}
```

5

Multiple Objects

You can create multiple objects of one class:

Example

```
public class MyClass {  
    int x = 5;  
  
    public static void main(String[] args) {  
        MyClass myObj1 = new MyClass();  
        MyClass myObj2 = new MyClass();  
        System.out.println(myObj1.x);  
        System.out.println(myObj2.x);  
    }  
}
```

5

5

Using Multiple Classes

Java Programming Language

63

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the **main()** method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory/folder:

- MyClass.java
- OtherClass.java

MyClass.java

```
public class MyClass {  
    int x = 5;  
}
```

OtherClass.java

```
class OtherClass {  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        System.out.println(myObj.x);  
    }  
}
```

When both files have been compiled:

```
C:\Users\Your Name>javac MyClass.java  
C:\Users\Your Name>javac OtherClass.java
```

Run the OtherClass.java file:

```
C:\Users\Your Name>java OtherClass
```

And the output will be:

```
class OtherClass {  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        System.out.println(myObj.x);  
    }  
}
```

• Java Class Attributes

In the previous chapter, we used the term "**variable**" for x in the example (as shown below). It is actually an attribute of the class. Or you could say that class attributes are variables within a class:

Example

Create a class called "**MyClass**" with two attributes: x and y:

```
public class MyClass {  
    int x = 5;  
    int y = 3;  
}
```

Another term for class attributes is **fields**.

Accessing Attributes

You can access attributes by creating an object of the class, and by using the dot syntax (.):

The following example will create an object of the **MyClass** class, with the name **myObj**.

We use the x attribute on the object to print its value:

Example

```
public class MyClass {  
    int x = 5;  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        System.out.println(myObj.x);  
    }  
}
```

Modify Attributes

You can also modify attribute values:

Example

```
public class MyClass {  
    int x;  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        myObj.x = 40;  
        System.out.println(myObj.x);  
    }  
}
```

40

Or override existing values:

Example

```
public class MyClass {  
    int x = 10;  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        myObj.x = 25; // x is now 25  
        System.out.println(myObj.x);  
    }  
}
```

25

If you don't want the ability to override existing values, declare the attribute as **final**:

Example

Java Programming Language

66

```
public class MyClass {  
    final int x = 10;  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        myObj.x = 25; // will generate an error  
        System.out.println(myObj.x);  
    }  
}
```

```
MyClass.java:6: error: cannot assign a value to final variable x  
    myObj.x = 25;  
           ^  
1 error
```

The `final` keyword is useful when you want a variable to always store the same value, like PI (3.14159...).

The `final` keyword is called a "modifier".

Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

Example

```
public class MyClass {  
    int x = 5;  
  
    public static void main(String[] args) {  
        MyClass myObj1 = new MyClass();  
        MyClass myObj2 = new MyClass();  
        myObj2.x = 25;  
        System.out.println(myObj1.x);  
        System.out.println(myObj2.x);  
    }  
}
```

```
5  
25
```

Multiple Attributes

You can specify as many attributes as you want:

Example

```
public class Person {  
    String fname = "John";  
    String lname = "Doe";  
    int age = 24;  
  
    public static void main(String[] args) {  
        Person myObj = new Person();  
        System.out.println("Name: " + myObj.fname + " " + myObj.lname);  
        System.out.println("Age: " + myObj.age);  
    }  
}
```

```
Name: John Doe  
Age: 24
```

- ### Java Class Methods

myMethod() prints a text (the action), when it is called. To call a method, write the method's name followed by two **parentheses ()** and a **semicolon;**

Example

```
public class MyClass {  
    static void myMethod() {  
        System.out.println("Hello World!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
    }  
}
```

```
Hello World!
```

Static vs. Non-Static

You will often see Java programs that have either static or public attributes and methods.

In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

Example

```
public class MyClass {  
    // Static method  
    static void myStaticMethod() {  
        System.out.println("Static methods can be called without creating objects");  
    }  
  
    // Public method  
    public void myPublicMethod() {  
        System.out.println("Public methods must be called by creating objects");  
    }  
  
    // Main method  
    public static void main(String[] args) {  
        myStaticMethod(); // Call the static method  
  
        MyClass myObj = new MyClass(); // Create an object of MyClass  
        myObj.myPublicMethod(); // Call the public method  
    }  
}
```

Static methods can be called without creating objects
Public methods must be called by creating objects

Access Methods with an Object

Example

Java Programming Language

69

```
// Create a Car class
public class Car {

    // Create a fullThrottle() method
    public void fullThrottle() {
        System.out.println("The car is going as fast as it can!");
    }

    // Create a speed() method and add a parameter
    public void speed(int maxSpeed) {
        System.out.println("Max speed is: " + maxSpeed);
    }

    // Inside main, call the methods on the myCar object
    public static void main(String[] args) {
        Car myCar = new Car();          // Create a myCar object
        myCar.fullThrottle();          // Call the fullThrottle() method
        myCar.speed(200);              // Call the speed() method
    }
}
```

```
The car is going as fast as it can!
Max speed is: 200
```

Example explained

- 1) We created a custom Car class with the **class** keyword.
- 2) We created the **fullThrottle()** and **speed()** methods in the Car class.
- 3) The **fullThrottle()** method and the **speed()** method will print out some text, when they are called.
- 4) The **speed()** method accepts an int parameter called **maxSpeed** - we will use this in 8).
- 5) In order to use the Car class and its methods, we need to create an object of the Car Class.
- 6) Then, go to the **main()** method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).
- 7) By using the new **keyword** we created a Car object with the name **myCar**.

Java Programming Language

70

8) Then, we call the **fullThrottle()** and **speed()** methods on the **myCar** object, and run the program using the name of the object (**myCar**), followed by a dot (.), followed by the name of the method (**fullThrottle()**; and **speed(200);**). Notice that we add an int parameter of 200 inside the **speed()** method.

Remember that.

The dot (.) is used to access the object's attributes and methods.

To call a method in Java, write the method name followed by a set of parentheses (), followed by a semicolon (;).

A class must have a matching filename (**Car** and **Car.java**).

Using Multiple Classes

Like we specified in the Classes chapter, it is a good practice to create an object of a class and access it in another class.

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory:

- Car.java
- OtherClass.java

Car.java

```
public class Car {  
    public void fullThrottle() {  
        System.out.println("The car is going as fast as it can!");  
    }  
  
    public void speed(int maxSpeed) {  
        System.out.println("Max speed is: " + maxSpeed);  
    }  
}
```

OtherClass.java

Java Programming Language

71

```
class OtherClass {  
    public static void main(String[] args) {  
        Car myCar = new Car();      // Create a myCar object  
        myCar.fullThrottle();      // Call the fullThrottle() method  
        myCar.speed(200);         // Call the speed() method  
    }  
}
```

When both files have been compiled:

```
C:\Users\Your Name>javac Car.java  
C:\Users\Your Name>javac OtherClass.java
```

Run the OtherClass.java file:

```
C:\Users\Your Name>java OtherClass
```

And the output will be:

```
class OtherClass {  
    public static void main(String[] args) {  
        Car myCar = new Car();      // Create a myCar object  
        myCar.fullThrottle();      // Call the fullThrottle() method  
        myCar.speed(200);         // Call the speed() method  
    }  
}
```

```
The car is going as fast as it can!  
Max speed is: 200
```

• Java Constructors

A **constructor** in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

Example

Java Programming Language

72

```
// Create a MyClass class
public class MyClass {
    int x;

    // Create a class constructor for the MyClass class
    public MyClass() {
        x = 5;
    }

    public static void main(String[] args) {
        MyClass myObj = new MyClass();
        System.out.println(myObj.x);
    }
}
```

5

Note that the constructor name must **match the class name**, and it cannot have a **return type** (like **void**).

Also note that the constructor is called when the object is created.

All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

Constructor Parameters

Constructors can also take parameters, which is used to initialize attributes.

The following example adds an int y parameter to the constructor. Inside the constructor we set x to y ($x=y$). When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:

Example

Java Programming Language

73

```
public class MyClass {  
    int x;  
  
    public MyClass(int y) {  
        x = y;  
    }  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass(5);  
        System.out.println(myObj.x);  
    }  
}
```

5

You can have as many parameters as you want:

Example

```
public class Car {  
    int modelYear;  
    String modelName;  
  
    public Car(int year, String name) {  
        modelYear = year;  
        modelName = name;  
    }  
  
    public static void main(String[] args) {  
        Car myCar = new Car(1969, "Mustang");  
        System.out.println(myCar.modelYear + " " + myCar.modelName);  
    }  
}
```

1969 Mustang

- **Java Modifiers**

Modifiers

Java Programming Language

74

By now, you are quite familiar with the public keyword that appears in almost all of our examples:

```
public class MyClass
```

The public keyword is an access modifier, meaning that it is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

- **Access Modifiers** - controls the access level
- **Non-Access Modifiers** - do not control access level, but provides other functionality.

Access Modifiers

For classes, you can use either **public** or **default**:

Modifier	Description
public	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier.

Final

If you don't want the ability to override existing attribute values, declare attributes as final:

Example

```
public class MyClass {  
    final int x = 10;  
    final double PI = 3.14;  
  
    public static void main(String[] args) {  
        MyClass myObj = new MyClass();  
        myObj.x = 50; // will generate an error  
        myObj.PI = 25; // will generate an error  
        System.out.println(myObj.x);  
    }  
}
```

Java Programming Language

75

```
MyClass.java:7: error: cannot assign a value to final variable x
    myObj.x = 50;
          ^
MyClass.java:8: error: cannot assign a value to final variable PI
    myObj.PI = 25;
          ^ 2 errors
```

Static

A **static method** means that it can be accessed without creating an object of the class, unlike public:

Example

```
public class MyClass {
    // Static method
    static void myStaticMethod() {
        System.out.println("Static methods can be called without creating objects");
    }

    // Public method
    public void myPublicMethod() {
        System.out.println("Public methods must be called by creating objects");
    }

    // Main method
    public static void main(String[] args) {
        myStaticMethod(); // Call the static method

        MyClass myObj = new MyClass(); // Create an object of MyClass
        myObj.myPublicMethod(); // Call the public method
    }
}
```

```
Static methods can be called without creating objects
Public methods must be called by creating objects
```

Abstract

An **abstract** method belongs to an abstract class, and it does not have a body. The body is provided by the subclass:

Java Programming Language

76

Example

```
// Code from filename: Person.java

// abstract class
abstract class Person {

    public String fname = "John";
    public int age = 24;
    public abstract void study(); // abstract method
}

// Subclass (inherit from Person)

class Student extends Person {

    public int graduationYear = 2018;

    public void study() { // the body of the abstract method is provided here
        System.out.println("Studying all day long");
    }
}

// End code from filename: Person.java

// Code from filename: MyClass.java

class MyClass {

    public static void main(String[] args) {
        // create an object of the Student class (which inherits attributes and methods from
        Person)
        Student myObj = new Student();
    }
}
```

```
System.out.println("Name: " + myObj.fname);
System.out.println("Age: " + myObj.age);
System.out.println("Graduation Year: " + myObj.graduationYear);
myObj.study(); // call abstract method
}
}
```

```
Name: John
Age: 24
Graduation Year: 2018
Studying all day long
```

- **Java Encapsulation**

Encapsulation

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users.

To achieve this, you must:

- declare class variables/attributes as private
- provide public get and set methods to access and update the value of a private variable

Get and Set

You learned that private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public get and set methods.

The get method returns the variable value, and the set method sets the value.

Syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter in upper case:

Example

Java Programming Language

78

```
public class Person {  
    private String name; // private = restricted access  
  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```

Example explained

The get method returns the value of the variable name.

The set method takes a parameter (**newName**) and assigns it to the name variable. This keyword is used to refer to the current object.

However, as the name variable is declared as private, we cannot access it from outside this class:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        Person myObj = new Person();  
        myObj.name = "John"; // error  
        System.out.println(myObj.name); // error  
    }  
}
```

If the variable was declared as public, we would expect the following output:

```
John
```

However, as we try to access a private variable, we get an error:

Java Programming Language

79

```
MyClass.java:4: error: name has private access in Person
    myObj.name = "John";
          ^
MyClass.java:5: error: name has private access in Person
    System.out.println(myObj.name);
          ^
2 errors
```

Instead, we use the `getName()` and `setName()` methods to **access** and update the variable:

Example

```
public class MyClass {
    public static void main(String[] args) {
        Person myObj = new Person();
        myObj.setName("John"); // Set the value of the name variable to "John"
        System.out.println(myObj.getName());
    }
}

// Outputs "John"
```

```
John
```

Why Encapsulation?

- Better control of class attributes and methods
- Class attributes can be made read-only (if you only use the get method), or write-only (if you only use the set method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

• Java Inheritance

Java Inheritance (Subclass and Superclass)

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from

To inherit from a class, use the **extends** keyword.

In the example below, the **Car** class (subclass) inherits the attributes and methods from the Vehicle class (superclass):

Example

```
class Vehicle {  
    protected String brand = "Ford";  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}  
  
class Car extends Vehicle {  
    private String modelName = "Mustang";  
    public static void main(String[] args) {  
        Car myFastCar = new Car();  
        myFastCar.honk();  
        System.out.println(myFastCar.brand + " " + myFastCar.modelName);  
    }  
}
```

```
Tuut, tuut!  
Ford Mustang
```

The final Keyword

If you don't want other classes to inherit from a class, use the **final** keyword:

If you try to access a final class, Java will generate an error:

```
final class Vehicle {  
    ...  
}  
  
class Car extends Vehicle {  
    ...  
}
```

Example

```
final class Vehicle {  
    protected String brand = "Ford";  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}  
  
class Car extends Vehicle {  
    private String modelName = "Mustang";  
    public static void main(String[] args) {  
        Car myFastCar = new Car();  
        myFastCar.honk();  
        System.out.println(myFastCar.brand + " " + myFastCar.modelName);  
    }  
}
```

```
Car.java:8: error: cannot inherit from final Vehicle  
class Car extends Vehicle {  
    ^  
1 error
```

• Java Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a superclass called Animal that has a method called **animalSound()**. Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Example

Java Programming Language

82

```
class Animal {  
    public void animalSound() {  
        System.out.println("The animal makes a sound");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: wee wee");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: bow wow");  
    }  
}
```

Now we can create Pig and Dog objects and call the **animalSound()** method on both of them:

Example

```
class Animal {  
    public void animalSound() {  
        System.out.println("The animal makes a sound");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: wee wee");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: bow wow");  
    }  
}
```

Java Programming Language

83

```
class MyMainClass {  
    public static void main(String[] args) {  
        Animal myAnimal = new Animal();  
        Animal myPig = new Pig();  
        Animal myDog = new Dog();  
  
        myAnimal.animalSound();  
        myPig.animalSound();  
        myDog.animalSound();  
    }  
}
```

```
The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow
```

Why and When to Use "Inheritance" and "Polymorphism"?

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

• Java Inner Classes

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class:

Example

```
class OuterClass {  
    int x = 10;  
  
    class InnerClass {  
        int y = 5;  
    }  
}  
  
public class MyMainClass {  
    public static void main(String[] args) {  
        OuterClass myOuter = new OuterClass();  
        OuterClass.InnerClass myInner = myOuter.new InnerClass();  
        System.out.println(myInner.y + myOuter.x);  
    }  
}
```

Private Inner Class

Unlike a "regular" class, an inner class can be private or protected. If you don't want outside objects to access the inner class, declare the class as private:

Example

```
class OuterClass {  
    int x = 10;  
  
    private class InnerClass {  
        int y = 5;  
    }  
}  
  
public class MyMainClass {  
    public static void main(String[] args) {  
        OuterClass myOuter = new OuterClass();  
        OuterClass.InnerClass myInner = myOuter.new InnerClass();  
        System.out.println(myInner.y + myOuter.x);  
    }  
}
```

```
MyMainClass.java:12: error: OuterClass.InnerClass has private access in OuterClass  
    OuterClass.InnerClass myInner = myOuter.new InnerClass();  
                           ^  
  
MyMainClass.java:12: error: OuterClass.InnerClass has private access in OuterClass  
    OuterClass.InnerClass myInner = myOuter.new InnerClass();  
                           ^  
2 errors
```

Static Inner Class

An **inner class** can also be static, which means that you can access it without creating an object of the outer class:

Example

```
class OuterClass {  
    int x = 10;  
  
    static class InnerClass {  
        int y = 5;  
    }  
}  
  
public class MyMainClass {  
    public static void main(String[] args) {  
        OuterClass.InnerClass myInner = new OuterClass.InnerClass();  
        System.out.println(myInner.y);  
    }  
}
```

5

Note: just like **static** attributes and methods, a **static** inner class does not have access to members of the outer class.

Access Outer Class from Inner Class

One advantage of inner classes, is that they can access attributes and methods of the outer class:

Example

```
class OuterClass {  
    int x = 10;  
  
    class InnerClass {  
        public int myInnerMethod() {  
            return x;  
        }  
    }  
}  
  
public class MyMainClass {  
    public static void main(String args[]) {  
        OuterClass myOuter = new OuterClass();  
        OuterClass.InnerClass myInner = myOuter.new InnerClass();  
        System.out.println(myInner.myInnerMethod());  
    }  
}
```

- **Java Abstraction**

Java Abstract Classes and Methods

Data abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class**: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal {  
    public abstract void animalSound();  
    public void sleep() {  
        System.out.println("Zzz");  
    }  
}
```

From the example above, it is not possible to create an object of the Animal class:

```
Animal myObj = new Animal(); // will generate an error
```

To access the abstract class, it must be inherited from another class.

Example

Java Programming Language

87

```
// Abstract class
abstract class Animal {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep() {
        System.out.println("Zzz");
    }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
}

class MyMainClass {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();

        myPig.sleep();
    }
}
```

```
The pig says: wee wee
Zzz
```

Why and When to Use Abstract Classes and Methods?

To achieve security - hide certain details and only show the important details of an object.

• Java User Input (Scanner)

The Scanner class is used to get user input, and it is found in the **java.util** package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the **nextLine()** method, which is used to read Strings:

Example

Java Programming Language

88

```
import java.util.Scanner; // import the Scanner class

class MyClass {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);
        String userName;

        // Enter username and press Enter
        System.out.println("Enter username");
        userName = myObj.nextLine();

        System.out.println("Username is: " + userName);
    }
}
```

```
Enter username
elaf
Username is: elaf
```

Input Types

In the example above, we used the **nextLine()** method, which is used to read Strings. To read other types, look at the table below:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

In the example below, we use different methods to read data of various types:

Example

```
import java.util.Scanner;

class MyClass {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter name, age and salary:");

        // String input
        String name = myObj.nextLine();

        // Numerical input
        int age = myObj.nextInt();
        double salary = myObj.nextDouble();

        // Output input by user
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
    }
}
```

```
Enter name, age and salary:
elaf
20
200
Name: elaf
Age: 20
Salary: 200
```

Note: If you enter wrong input (e.g. text in a numerical input), you will get an exception/error message (like "InputMismatchException").

• Java Date and Time

Java Dates

Java Programming Language

90

Java does not have a built-in Date class, but we can import the `java.time` package to work with the date and time API. The package includes many date and time classes. For example:

Class	Description
<code>LocalDate</code>	Represents a date (year, month, day (yyyy-MM-dd))
<code>LocalTime</code>	Represents a time (hour, minute, second and milliseconds (HH-mm-ss-zzz))
<code>LocalDateTime</code>	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss.zzz)
<code>DateTimeFormatter</code>	Formatter for displaying and parsing date-time objects

Display Current Date

To display the current date, import the `java.time.LocalDate` class, and use its `now()` method:

Example

```
import java.time.LocalDate; // import the LocalDate class

public class MyClass {
    public static void main(String[] args) {
        LocalDate myObj = LocalDate.now(); // Create a date object
        System.out.println(myObj); // Display the current date
    }
}
```

2020-05-15

Display Current Time

Java Programming Language

91

To display the current time (hour, minute, second, and milliseconds), import the **java.time.LocalTime** class, and use its **now()** method:

Example

```
import java.time.LocalTime; // import the LocalTime class

public class MyClass {
    public static void main(String[] args) {
        LocalTime myObj = LocalTime.now();
        System.out.println(myObj);
    }
}
```

20:36:45.173001

Display Current Date and Time

To display the current date and time, import the **java.time.LocalDateTime** class, and use its **now()** method:

Example

```
import java.time.LocalDateTime; // import the LocalDateTime class

public class MyClass {
    public static void main(String[] args) {
        LocalDateTime myObj = LocalDateTime.now();
        System.out.println(myObj);
    }
}
```

2020-05-15T20:37:42.326941

Formatting Date and Time

The "T" in the example above is used to separate the date from the time. You can use the **DateTimeFormatter** class with the **ofPattern()** method in the same package to format or

Java Programming Language

92

parse date-time objects. The following example will remove both the "T" and milliseconds from the date-time:

Example

```
import java.time.LocalDateTime; // Import the LocalDateTime class
import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class MyClass {
    public static void main(String[] args) {
        LocalDateTime myDateObj = LocalDateTime.now();
        System.out.println("Before formatting: " + myDateObj);
        DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

        String formattedDate = myDateObj.format(myFormatObj);
        System.out.println("After formatting: " + formattedDate);
    }
}
```

```
Before Formatting: 2020-05-15T20:38:49.376260
After Formatting: 15-05-2020 20:38:49
```

The **ofPattern()** method accepts all sorts of values, if you want to display the date and time in a different format. For example:

Value	Example
yyyy-MM-dd	"1988-09-29"
dd/MM/yyyy	"29/09/1988"
dd-MMM-yyyy	"29-Sep-1988"
E, MMM dd yyyy	"Thu, Sep 29 1988"

• Java ArrayList

The **ArrayList** class is a resizable array, which can be found in the **java.util** package.

The difference between a built-in array and an **ArrayList** in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you

have to create a new one). While elements can be added and removed from an **ArrayList** whenever you want. The syntax is also slightly different:

Example

Create an **ArrayList** object called cars that will store strings:

```
import java.util.ArrayList; // import the ArrayList class

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

Add Items

The **ArrayList** class has many useful methods. For example, to add elements to the **ArrayList**, use the **add()** method:

Example

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars);
    }
}
```

```
[Volvo, BMW, Ford, Mazda]
```

Access an Item

To access an element in the **ArrayList**, use the **get()** method and refer to the index number:

Example

Java Programming Language

94

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars.get(0));
    }
}
```

Volvo

Remember: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Item

To modify an element, use the **set()** method and refer to the index number:

Example

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        cars.set(0, "Opel");
        System.out.println(cars);
    }
}
```

[Opel, BMW, Ford, Mazda]

Remove an Item

Java Programming Language

95

To remove an element, use the **remove()** method and refer to the index number:

Example

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        cars.remove(0);
        System.out.println(cars);
    }
}
```

```
[BMW, Ford, Mazda]
```

To remove all the elements in the **ArrayList**, use the **clear()** method:

Example

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        cars.clear();
        System.out.println(cars);
    }
}
```

```
[]
```

ArrayList Size

To find out how many elements an **ArrayList** have, use the **size** method:

Example

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars.size());
    }
}
```

4

Loop Through an ArrayList

Loop through the elements of an **ArrayList** with a for loop, and use the **size()** method to specify how many times the loop should run:

Example

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        for (int i = 0; i < cars.size(); i++) {
            System.out.println(cars.get(i));
        }
    }
}
```

Java Programming Language

97

```
Volvo  
BMW  
Ford  
Mazda
```

You can also loop through an **ArrayList** with the for-each loop:

Example

```
import java.util.ArrayList;  
  
public class MyClass {  
    public static void main(String[] args) {  
        ArrayList<String> cars = new ArrayList<String>();  
        cars.add("Volvo");  
        cars.add("BMW");  
        cars.add("Ford");  
        cars.add("Mazda");  
        for (String i : cars) {  
            System.out.println(i);  
        }  
    }  
}
```

```
Volvo  
BMW  
Ford  
Mazda
```

Other Types

Elements in an **ArrayList** are actually objects. In the examples above, we created elements (objects) of type "**String**". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent wrapper class: Integer. For other primitive types, use: Boolean for **Boolean**, Character for char, Double for double, etc:

Example

Java Programming Language

98

```
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<Integer> myNumbers = new ArrayList<Integer>();
        myNumbers.add(10);
        myNumbers.add(15);
        myNumbers.add(20);
        myNumbers.add(25);
        for (int i : myNumbers) {
            System.out.println(i);
        }
    }
}
```

```
10
15
20
25
```

Sort an ArrayList

Another useful class in the **java.util** package is the Collections class, which include the **sort()** method for sorting lists alphabetically or numerically:

Example

```
import java.util.ArrayList;
import java.util.Collections;

public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");

        Collections.sort(cars);

        for (String i : cars) {
            System.out.println(i);
        }
    }
}
```

Java Programming Language

99

```
BMW  
Ford  
Mazda  
Volvo
```

Example

```
import java.util.ArrayList;  
import java.util.Collections;  
  
public class MyClass {  
    public static void main(String[] args) {  
        ArrayList<Integer> myNumbers = new ArrayList<Integer>();  
        myNumbers.add(33);  
        myNumbers.add(15);  
        myNumbers.add(20);  
        myNumbers.add(34);  
        myNumbers.add(8);  
        myNumbers.add(12);  
  
        Collections.sort(myNumbers);  
  
        for (int i : myNumbers) {  
            System.out.println(i);  
        }  
    }  
}
```

```
8  
12  
15  
20  
33  
34
```

- **Java Exceptions - Try...Catch**

Java Exceptions

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

Java Programming Language

100

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

Java try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

Syntax

```
try {  
    // Block of code to try  
}  
catch(Exception e) {  
    // Block of code to handle errors  
}
```

Consider the following example:

This will generate an error, because **myNumbers[10]** does not exist.

```
public class MyClass {  
    public static void main(String[] args) {  
        int[] myNumbers = {1, 2, 3};  
        System.out.println(myNumbers[10]);  
    }  
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
at MyClass.main(MyClass.java:4)
```

If an error occurs, we can use **try...catch** to catch the error and execute some code to handle it:

Example

Java Programming Language

101

```
public class MyClass {  
    public static void main(String[] args) {  
        try {  
            int[] myNumbers = {1, 2, 3};  
            System.out.println(myNumbers[10]);  
        } catch (Exception e) {  
            System.out.println("Something went wrong.");  
        }  
    }  
}
```

```
Something went wrong.
```

Finally

The **finally** statement lets you execute code, after **try...catch**, regardless of the result:

Example

```
public class MyClass {  
    public static void main(String[] args) {  
        try {  
            int[] myNumbers = {1, 2, 3};  
            System.out.println(myNumbers[10]);  
        } catch (Exception e) {  
            System.out.println("Something went wrong.");  
        } finally {  
            System.out.println("The 'try catch' is finished.");  
        }  
    }  
}
```

```
Something went wrong.  
The 'try catch' is finished.
```

The throw keyword

The **throw** statement allows you to create a custom error.

Java Programming Language

102

The throw statement is used together with an exception type. There are many exception types available in Java: **ArithmaticException**, **FileNotFoundException**, **ArrayIndexOutOfBoundsException**, **SecurityException**, etc:

Example

```
public class MyClass {  
    static void checkAge(int age) {  
        if (age < 18) {  
            throw new ArithmaticException("Access denied - You must be at least 18 years old.");  
        } else {  
            System.out.println("Access granted - You are old enough!");  
        }  
    }  
  
    public static void main(String[] args) {  
        checkAge(15);  
    }  
}
```

```
Exception in thread "main" java.lang.ArithmaticException: Access denied - You must be at  
least 18 years old.  
    at MyClass.checkAge(MyClass.java:4)  
    at MyClass.main(MyClass.java:12)
```

If age was 20, you would not get an exception:

Example

```
public class MyClass {  
    static void checkAge(int age) {  
        if (age < 18) {  
            throw new ArithmaticException("Access denied - You must be at least 18 years old.");  
        } else {  
            System.out.println("Access granted - You are old enough!");  
        }  
    }  
  
    public static void main(String[] args) {  
        checkAge(20);  
    }  
}
```

```
Access granted - You are old enough!
```

Java Script

Programming Language

Java Script Programming Language

103

JAVA SCRIPT INTRODUCTION

- **JavaScript Where To**

The <script> Tag

In HTML, JavaScript code is inserted between **<script>** and **</script>** tags.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>

</body>
</html>
```

JavaScript in Body

My First JavaScript

Old JavaScript examples may use a type attribute: **<script type="text/javascript">**.

The type attribute is not required. JavaScript is the default scripting language in HTML.

JavaScript Functions and Events

A JavaScript function is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an event occurs, like when the user clicks a button.

JavaScript in <head> or <body>

Java Script Programming Language

104

You can place any number of scripts in an HTML document.

Scripts can be placed in the **<body>**, or in the **<head>** section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript function is placed in the **<head>** section of an HTML page. The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>JavaScript in Head</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in Head

A Paragraph.

Try it

Java Script Programming Language

105

JavaScript in Head

Paragraph changed.

Try it

JavaScript in <body>

In this example, a JavaScript function is placed in the <boby> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

JavaScript in Body

A Paragraph.

Try it

Java Script Programming Language

106

JavaScript in Body

Paragraph changed.

Try it

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

External JavaScript

Scripts can also be placed in external files:

External file: myScript.js

```
function myFunction() {  
    document.getElementById("demo").innerHTML = "Paragraph changed.";  
}
```

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the **src** (source) attribute of a **<script>** tag:

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>External JavaScript</h2>  
  
<p id="demo">A Paragraph.</p>  
  
<button type="button" onclick="myFunction()">Try it</button>  
  
<p>(myFunction is stored in an external file called "myScript.js")</p>  
  
<script src="myScript.js"></script>  
  
</body>  
</html>
```

Java Script Programming Language

107

External JavaScript

A Paragraph.

Try it

(myFunction is stored in an external file called "myScript.js")

External JavaScript

Paragraph changed.

Try it

(myFunction is stored in an external file called "myScript.js")

You can place an external script reference in **<head>** or **<body>** as you like.

The script will behave as if it was located exactly where the **<script>** tag is located.

External scripts cannot contain **<script>** tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

External scripts can be referenced with a **full URL** or with a path relative to the current web page.

Java Script Programming Language

108

This example uses a **full URL** to link to a script:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>(myFunction is stored in an external file called "myScript.js")</p>

<script src="https://www.w3schools.com/js/myScript.js"></script>

</body>
</html>
```

External JavaScript

A Paragraph.

Try it

(myFunction is stored in an external file called "myScript.js")

External JavaScript

Paragraph changed.

Try it

(myFunction is stored in an external file called "myScript.js")

This example uses a script located in a specified folder on the current web site:

Example

Java Script Programming Language

109

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>(myFunction is stored in an external file called "myScript.js")</p>

<script src="/js/myScript.js"></script>

</body>
</html>
```

External JavaScript

A Paragraph.

Try it

(myFunction is stored in an external file called "myScript.js")

External JavaScript

Paragraph changed.

Try it

(myFunction is stored in an external file called "myScript.js")

This example links to a script located in the same folder as the current page:

Example

Java Script Programming Language

110

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>(myFunction is stored in an external file called "myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```

External JavaScript

A Paragraph.

Try it

(myFunction is stored in an external file called "myScript.js")

External JavaScript

Paragraph changed.

Try it

(myFunction is stored in an external file called "myScript.js")

JAVA SCRIPT TUTORIAL

• JavaScript Output

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

Java Script Programming Language

111

- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

Using innerHTML

To access an HTML element, JavaScript can use the `document.getElementById(id)` method.

The id attribute defines the HTML element. The `innerHTML` property defines the HTML content:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My First Paragraph.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

My First Web Page

My First Paragraph.

11

Changing the **innerHTML** property of an HTML element is a common way to display data in HTML.

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

Java Script Programming

Language

112

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.
It will overwrite the whole document.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

My First Web Page

My first paragraph.

Never call document.write after the document has finished loading. It will overwrite the whole document.

11

Using document.write() after an HTML document is loaded, will **delete all existing**

HTML:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Java Script Programming Language

113

My First Web Page

My first paragraph.

Try it

11

The **document.write()** method should only be used for testing.

Using window.alert()

You can use an **alert box** to display data:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

An embedded page on this page says

11

OK

Java Script Programming Language

114

My First Web Page

My first paragraph.

You can skip the **window** keyword.

In JavaScript, the window object is the global scope object, that means that variables, properties, and methods by default belongs to the window object. This also means that specifying the window keyword is optional:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>

</body>
</html>
```

An embedded page on this page says

11

OK

My First Web Page

My first paragraph.

Using console.log()

Java Script Programming Language

115

For debugging purposes, you can call the **console.log()** method in the browser to display data.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Activate Debugging</h2>

<p>F12 on your keyboard will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

Activate Debugging

F12 on your keyboard will activate debugging.

Then select "Console" in the debugger menu.

Then click Run again.

JavaScript Print

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the **window.print()** method in the browser to print the content of the current window.

Example

Java Script Programming Language

116

```
<!DOCTYPE html>
<html>
<body>

<h2>The window.print() Method</h2>
<p>Click the button to print the current page.</p>
<button onclick="window.print()">Print this page</button>

</body>
</html>
```

The window.print() Method

Click the button to print the current page.

[Print this page](#)

• JavaScript Statements

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>
<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a computer.</p>

<p id="demo"></p>

<script>
var x, y, z; // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;   // Statement 4

document.getElementById("demo").innerHTML =
"The value of z is " + z + ".";
</script>

</body>
</html>
```

Java Script Programming Language

117

JavaScript Statements

A **JavaScript program** is a list of **statements** to be executed by a computer.

The value of z is 11.

JavaScript Programs

A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called statements.

A **JavaScript** program is a list of programming statements.

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello Dolly.";
</script>

</body>
</html>
```

Java Script Programming Language

118

JavaScript Statements

In HTML, JavaScript statements are executed by the browser.

Hello Dolly.

Most **JavaScript** programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

Semicolons ;

Semicolons separate JavaScript statements.

Add a **semicolon** at the end of each executable statement:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>JavaScript statements are separated by semicolons.</p>
<p id="demo1"></p>

<script>
var a, b, c;
a = 5;
b = 6;
c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>

</body>
</html>
```

JavaScript Statements

JavaScript statements are separated by semicolons.

11

When separated by **semicolons**, multiple statements on one line are allowed:

Java Script Programming Language

119

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>Multiple statements on one line is allowed.</p>

<p id="demo1"></p>

<script>
var a, b, c;
a = 5; b = 6; c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>

</body>
</html>
```

JavaScript Statements

Multiple statements on one line is allowed.

11

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

A good practice is to put spaces around operators (`= + - * /`):

```
var x = y + z;
```

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a **JavaScript** statement does not fit on one line, the best place to break it is after an operator:

Java Script Programming Language

120

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>
The best place to break a code line is after an operator or a comma.
</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Hello Dolly!";
</script>

</body>
</html>
```

JavaScript Statements

The best place to break a code line is after an operator or a comma.

Hello Dolly!

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets `{ ... }`.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in **JavaScript** functions:

Example

Java Script Programming Language

121

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>JavaScript code blocks are written between { and }</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
</script>

</body>
</html>
```

JavaScript Statements

JavaScript code blocks are written between { and }

Click Me!

JavaScript Statements

JavaScript code blocks are written between { and }

Click Me!

Hello Dolly!

How are you?

• JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

Java Script Programming Language

122

```
var x, y, z;           // How to declare variables
x = 5; y = 6;          // How to assign values
z = x + y;            // How to compute values
```

JavaScript Values

The **JavaScript** syntax defines two types of values: Fixed values and variable values. Fixed values are called literals. Variable values are called **variables**.

JavaScript Literals

The most important rules for writing fixed values are:

Numbers are written with or without decimals:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Number can be written with or without decimals.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 10.50;
</script>

</body>
</html>
```

JavaScript Numbers

Number can be written with or without decimals.

10.5

Strings are text, written within double or single quotes:

Java Script Programming Language

123

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings can be written with double or single quotes.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 'John Doe';
</script>

</body>
</html>
```

JavaScript Strings

Strings can be written with double or single quotes.

John Doe

JavaScript Variables

In a programming language, **variables** are used to store data values.

JavaScript uses the var keyword to declare variables.

An equal sign is used to assign values to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

Java Script Programming Language

124

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, x is defined as a variable.  
Then, x is assigned the value of 6:</p>

<p id="demo"></p>

<script>
var x;
x = 6;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Variables

In this example, x is defined as a variable. Then, x is assigned the value of 6:

6

JavaScript Operators

JavaScript uses arithmetic operators (+ - * /) to compute values:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Operators</h2>

<p>JavaScript uses arithmetic operators to compute values (just like algebra).</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = (5 + 6) * 10;
</script>

</body>
</html>
```

Java Script Programming Language

125

JavaScript Operators

JavaScript uses arithmetic operators to compute values (just like algebra).

110

JavaScript uses an assignment operator (`=`) to assign values to variables:

```
<!DOCTYPE html>
<html>
<body>

<h2>Assigning JavaScript Values</h2>

<p>In JavaScript the = operator is used to assign values to variables.</p>

<p id="demo"></p>

<script>
var x, y;
x = 5;
y = 6;
document.getElementById("demo").innerHTML = x + y;
</script>

</body>
</html>
```

Assigning JavaScript Values

In JavaScript the `=` operator is used to assign values to variables.

11

JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, $5 * 10$ evaluates to 50:

Java Script Programming Language

126

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 * 10;
</script>

</body>
</html>
```

JavaScript Expressions

Expressions compute to values.

50

Expressions can also contain variable values:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>
var x;
x = 5;
document.getElementById("demo").innerHTML = x * 10;
</script>

</body>
</html>
```

JavaScript Expressions

Expressions compute to values.

50

Java Script Programming Language

127

The values can be of various types, such as numbers and strings.

For example, "**John**" + " " + "**Doe**", evaluates to "**John Doe**":

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "John" + " " + "Doe";
</script>

</body>
</html>
```

JavaScript Expressions

Expressions compute to values.

John Doe

JavaScript Keywords

JavaScript keywords are used to identify actions to be performed.

The var keyword tells the browser to create variables:

Java Script Programming Language

128

```
<!DOCTYPE html>
<html>
<body>

<h2>The var Keyword Creates Variables</h2>

<p id="demo"></p>

<script>
var x, y;
x = 5 + 6;
y = x * 10;
document.getElementById("demo").innerHTML = y;
</script>

</body>
</html>
```

The var Keyword Creates Variables

110

JavaScript Comments

Not all **JavaScript statements** are "executed".

Code after double slashes // or between /* and */ is treated as a comment.

Comments are ignored, and will not be executed:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Comments are NOT Executed</h2>

<p id="demo"></p>

<script>
var x;
x = 5;
// x = 6; I will not be executed
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

129

JavaScript Comments are NOT Executed

5

JavaScript Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign (\$).

Subsequent characters may be letters, digits, underscores, or dollar signs.

JavaScript is Case Sensitive

All **JavaScript identifiers** are case sensitive.

The variables **lastName** and **lastname**, are two different variables:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript is Case Sensitive</h2>

<p>Try change lastName to lastname.</p>

<p id="demo"></p>

<script>
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
document.getElementById("demo").innerHTML = lastName;
</script>

</body>
</html>
```

Java Script Programming Language

130

JavaScript is Case Sensitive

Try change lastName to lastname.

Doe

JavaScript does not interpret **VAR** or **Var** as the keyword **var**.

JavaScript and Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name:

Hyphens:

first-name, last-name, master-card, inter-city.

Underscore:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

Lower Camel Case:

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

• JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with **//**.

Java Script Programming Language

131

Any text between `//` and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="myH"></h1>
<p id="myP"></p>

<script>
// Change heading:
document.getElementById("myH").innerHTML = "JavaScript Comments";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>

</body>
</html>
```

JavaScript Comments

My first paragraph.

This example uses a single line comment at the end of each line to explain the code:

Example

Java Script Programming Language

132

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Comments</h2>

<p id="demo"></p>

<script>
var x = 5;      // Declare x, give it the value of 5
var y = x + 2; // Declare y, give it the value of x + 2
// Write y to demo:
document.getElementById("demo").innerHTML = y;
</script>

</body>
</html>
```

JavaScript Comments

7

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.

Any text between `/*` and `*/` will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

Example

Java Script Programming Language

133

```
<!DOCTYPE html>
<html>
<body>

<h1 id="myH"></h1>
<p id="myP"></p>

<script>
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
*/
document.getElementById("myH").innerHTML = "JavaScript Comments";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>

</body>
</html>
```

JavaScript Comments

My first paragraph.

Using Comments to Prevent Execution

Using **comments** to prevent execution of code is suitable for code testing.

Adding **//** in front of a code line changes the code lines from an executable line to a comment.

This example uses **//** to prevent execution of one of the code lines:

Example

Java Script Programming Language

134

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Comments</h2>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>

<p>The line starting with // is not executed.</p>

</body>
</html>
```

JavaScript Comments

My first paragraph.

The line starting with // is not executed.

This example uses a comment block to prevent execution of multiple lines:

Example

Java Script Programming Language

135

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Comments</h2>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>
/*
document.getElementById("myH").innerHTML = "Welcome to my Homepage";
document.getElementById("myP").innerHTML = "This is my first paragraph.";
*/
document.getElementById("myP").innerHTML = "The comment-block is not executed.";
</script>

</body>
</html>
```

JavaScript Comments

The comment-block is not executed.

• JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

Example

Java Script Programming Language

136

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, x, y, and z are variables.</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

JavaScript Variables

In this example, x, y, and z are variables.

The value of z is: 11

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- z stores the value 11

Much Like Algebra

In this example, price1, price2, and total, are variables:

Example

Java Script Programming Language

137

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p id="demo"></p>

<script>
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
document.getElementById("demo").innerHTML =
"The total is: " + total;
</script>

</body>
</html>
```

JavaScript Variables

The total is: 11

In programming, just like in algebra, we use variables (like price1) to hold values.

In programming, just like in algebra, we use variables in expressions (total = price1 + price2).

From the example above, you can calculate the total to be 11.

JavaScript Identifiers

All **JavaScript variables** must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with \$ and _ (but we will not use it in this tutorial)

Java Script Programming Language

138

- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

The Assignment Operator

In JavaScript, the equal sign (`=`) is an "**assignment**" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

`x = x + 5`

In JavaScript, however, it makes perfect sense: it assigns the value of $x + 5$ to x .

(It calculates the value of $x + 5$ and puts the result into x . The value of x is incremented by 5.)

The "equal to" operator is written like `==` in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

Java Script Programming Language

139

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>Strings are written with quotes.</p>
<p>Numbers are written without quotes.</p>

<p id="demo"></p>

<script>
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';

document.getElementById("demo").innerHTML =
pi + "<br>" + person + "<br>" + answer;
</script>

</body>
</html>
```

JavaScript Variables

Strings are written with quotes.

Numbers are written without quotes.

3.14
John Doe
Yes I am!

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the var keyword:

```
var carName;
```

After the declaration, the variable has no value (technically it has the value of undefined).

To assign a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

Java Script Programming Language

140

```
var carName = "Volvo";
```

In the example below, we create a variable called **carName** and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with **id="demo"**:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>Create a variable, assign a value to it, and display it:</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

JavaScript Variables

Create a variable, assign a value to it, and display it:

Volvo

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with var and separate the variables by comma:

Java Script Programming

Language

141

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>You can declare many variables in one statement.</p>

<p id="demo"></p>

<script>
var person = "John Doe", carName = "Volvo", price = 200;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

JavaScript Variables

You can declare many variables in one statement.

Volvo

A declaration can span multiple lines:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>You can declare many variables in one statement.</p>

<p id="demo"></p>

<script>
var person = "John Doe",
carName = "Volvo",
price = 200;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

Java Script Programming Language

142

JavaScript Variables

You can declare many variables in one statement.

Volvo

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A **variable** declared without a value will have the value **undefined**.

The variable **carName** will have the value **undefined** after the execution of this statement:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>A variable declared without a value will have the value undefined.</p>

<p id="demo"></p>

<script>
var carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

JavaScript Variables

A variable declared without a value will have the value **undefined**.

undefined

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

Java Script Programming Language

143

The variable **carName** will still have the value "Volvo" after the execution of these statements:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>If you re-declare a JavaScript variable, it will not lose its value.</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
var carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

Volvo

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

Example

Java Script Programming Language

144

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>The result of adding 5 + 2 + 3:</p>

<p id="demo"></p>

<script>
var x = 5 + 2 + 3;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Variables

The result of adding 5 + 2 + 3:

10

You can also add strings, but strings will be concatenated:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>The result of adding "John" + " " + "Doe":</p>

<p id="demo"></p>

<script>
var x = "John" + " " + "Doe";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

145

JavaScript Variables

The result of adding "John" + " " + "Doe":

John Doe

Also try this:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>The result of adding "5" + 2 + 3:</p>

<p id="demo"></p>

<script>
x = "5" + 2 + 3;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Variables

The result of adding "5" + 2 + 3:

523

Now try this:

Example

Java Script Programming Language

146

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>The result of adding 2 + 3 + "5":</p>

<p id="demo"></p>

<script>
var x = 2 + 3 + "5"
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Variables

The result of adding 2 + 3 + "5":

55

JavaScript Dollar Sign \$

Remember that JavaScript identifiers (names) must begin with:

- A letter (A-Z or a-z)
- A dollar sign (\$)
- Or an underscore (_)

Since JavaScript treats a dollar sign as a letter, identifiers containing \$ are valid variable names:

Example

Java Script Programming Language

147

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript $</h2>

<p>The dollar sign is treated as a letter in JavaScript names.</p>

<p id="demo"></p>

<script>
var $ = 2;
var $myMoney = 5;
document.getElementById("demo").innerHTML = $ + $myMoney;
</script>

</body>
</html>
```

JavaScript S

The dollar sign is treated as a letter in JavaScript names.

7

Using the **dollar** sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function **\$** is used to select HTML elements. In jQuery **\$(“p”);** means "select all p elements".

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing **_** are valid variable names:

Example

Java Script Programming Language

148

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript $</h2>

<p>The underscore is treated as a letter in JavaScript names.</p>

<p id="demo"></p>

<script>
var _x = 2;
var _100 = 5;
document.getElementById("demo").innerHTML = _x + _100;
</script>

</body>
</html>
```

JavaScript S

The underscore is treated as a letter in JavaScript names.

7

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (**hidden**)" variables.

JavaScript Operators

Example

Java Script Programming Language

149

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Operators</h2>

<p>x = 5, y = 2, calculate z = x + y, and display z:</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 2;
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Operators

x = 5, y = 2, calculate z = x + y, and display z:

7

The assignment operator (=) assigns a value to a variable.

Assignment

```
<!DOCTYPE html>
<html>
<body>

<h2>The = Operator</h2>

<p id="demo"></p>

<script>
var x = 10;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

150

The = Operator

10

The addition operator (+) adds numbers:

Adding

```
<!DOCTYPE html>
<html>
<body>

<h2>The + Operator</h2>

<p id="demo"></p>

<script>
var x = 5;
var y = 2;
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

The + Operator

7

The multiplication operator (*) multiplies numbers.

Multiplying

Java Script Programming Language

151

```
<!DOCTYPE html>
<html>
<body>

<h2>The * Operator</h2>

<p id="demo"></p>

<script>
var x = 5;
var y = 2;
var z = x * y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

The * Operator

10

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Java Script Programming Language

152

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

The addition assignment operator (**+=**) adds a value to a variable.

Assignment

```
<!DOCTYPE html>
<html>
<body>

<h2>The += Operator</h2>

<p id="demo"></p>

<script>
var x = 10;
x += 5;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

The += Operator

15

Java Script Programming Language

153

JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Operators</h2>
<p>The + operator concatenates (adds) strings.</p>
<p id="demo"></p>

<script>
var txt1 = "John";
var txt2 = "Doe";
document.getElementById("demo").innerHTML = txt1 + " " + txt2;
</script>

</body>
</html>
```

JavaScript Operators

The + operator concatenates (adds) strings.

John Doe

The += assignment operator can also be used to add (concatenate) strings:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Operators</h2>
<p>The assignment operator += can concatenate strings.</p>
<p id="demo"></p>

<script>
txt1 = "What a very ";
txt1 += "nice day";
document.getElementById("demo").innerHTML = txt1;
</script>

</body>
</html>
```

Java Script Programming Language

154

JavaScript Operators

The assignment operator `+=` can concatenate strings.

What a very nice day

When used on strings, the `+` operator is called the concatenation operator.

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Operators</h2>

<p>Adding a number and a string, returns a string.</p>

<p id="demo"></p>

<script>
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
document.getElementById("demo").innerHTML =
x + "<br>" + y + "<br>" + z;
</script>

</body>
</html>
```

JavaScript Operators

Adding a number and a string, returns a string.

10
55
Hello5

JavaScript Comparison Operators

Java Script Programming Language

155

Operator	Description
<code>==</code>	equal to
<code>===</code>	equal value and equal type
<code>!=</code>	not equal
<code>!==</code>	not equal value or not equal type
<code>></code>	greater than
<code><</code>	less than
<code>>=</code>	greater than or equal to
<code><=</code>	less than or equal to
<code>?</code>	ternary operator

JavaScript Logical Operators

Operator	Description
<code>&&</code>	logical and
<code> </code>	logical or
<code>!</code>	logical not

JavaScript Type Operators

Operator	Description
<code>typeof</code>	Returns the type of a variable
<code>instanceof</code>	Returns true if an object is an instance of an object type

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Java Script Programming Language

156

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
	OR	5 1	0101 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

• JavaScript Data Types

JavaScript variables can hold many data types: numbers, strings, objects and more:

```
var length = 16;                                // Number
var lastName = "Johnson";                         // String
var x = {firstName:"John", lastName:"Doe"};        // Object
```

The Concept of Data Types

In programming, data types are an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
var x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

Java Script Programming Language

157

JavaScript will treat the example above as:

```
var x = "16" + "Volvo";
```

When adding a number and a string, JavaScript will treat the number as a string.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>When adding a number and a string, JavaScript will treat the number as a string.</p>

<p id="demo"></p>

<script>
var x = 16 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript

When adding a number and a string, JavaScript will treat the number as a string.

16Volvo

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>When adding a string and a number, JavaScript will treat the number as a string.</p>

<p id="demo"></p>

<script>
var x = "Volvo" + 16;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

158

JavaScript

When adding a string and a number, JavaScript will treat the number as a string.

Volvo16

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>JavaScript evaluates expressions from left to right. Different sequences can produce different results:</p>

<p id="demo"></p>

<script>
var x = 16 + 4 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

20Volvo

Java Script Programming Language

159

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>JavaScript evaluates expressions from left to right. Different sequences can produce different results:</p>

<p id="demo"></p>

<script>
var x = "Volvo" + 16 + 4;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

Volvo164

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "**Volvo**".

In the second example, since the first operand is a string, all operands are treated as strings.

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

Java Script Programming

Language

160

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Data Types</h2>

<p>JavaScript has dynamic types. This means that the same variable can be used to hold different data types:</p>

<p id="demo"></p>

<script>
var x;           // Now x is undefined
x = 5;          // Now x is a Number
x = "John";     // Now x is a String

document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Data Types

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

John

JavaScript Strings

A **string** (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

Java Script Programming

Language

161

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings are written with quotes. You can use single or double quotes:</p>

<p id="demo"></p>

<script>
var carName1 = "Volvo XC60";
var carName2 = 'Volvo XC60';

document.getElementById("demo").innerHTML =
carName1 + "<br>" +
carName2;
</script>

</body>
</html>
```

JavaScript Strings

Strings are written with quotes. You can use single or double quotes:

Volvo XC60
Volvo XC60

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

Java Script Programming Language

162

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>You can use quotes inside a string, as long as they don't match the quotes surrounding the string:</p>

<p id="demo"></p>

<script>
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';

document.getElementById("demo").innerHTML =
answer1 + "<br>" +
answer2 + "<br>" +
answer3;
</script>

</body>
</html>
```

JavaScript Strings

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

It's alright
He is called 'Johnny'
He is called "Johnny"

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

Java Script Programming Language

163

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Numbers can be written with, or without decimals:</p>

<p id="demo"></p>

<script>
var x1 = 34.00;
var x2 = 34;
var x3 = 3.14;

document.getElementById("demo").innerHTML =
x1 + "<br>" + x2 + "<br>" + x3;
</script>

</body>
</html>
```

JavaScript Numbers

Numbers can be written with, or without decimals:

34
34
3.14

Extra-large or extra small numbers can be written with scientific (exponential) notation:

Example

Java Script Programming

Language

164

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Extra large or extra small numbers can be written with scientific (exponential) notation:</p>

<p id="demo"></p>

<script>
var y = 123e5;
var z = 123e-5;

document.getElementById("demo").innerHTML =
y + "<br>" + z;
</script>

</body>
</html>
```

JavaScript Numbers

Extra large or extra small numbers can be written with scientific (exponential) notation:

12300000
0.00123

JavaScript Booleans

Booleans can only have two values: true or false.

Example

Java Script Programming Language

165

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Booleans</h2>

<p>Booleans can have two values: true or false:</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 5;
var z = 6;
document.getElementById("demo").innerHTML =
(x == y) + "<br>" + (x == z);
</script>

</body>
</html>
```

JavaScript Booleans

Booleans can have two values: true or false:

true
false

Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example

Java Script Programming Language

166

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Array indexes are zero-based, which means the first item is [0].</p>

<p id="demo"></p>

<script>
var cars = ["Saab","Volvo","BMW"];

document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

JavaScript Arrays

Array indexes are zero-based, which means the first item is [0].

Saab

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

JavaScript Objects

JavaScript objects are written with curly braces {}.

Object properties are written as **name: value** pairs, separated by commas.

Example

Java Script Programming Language

167

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
var person = {
    firstName : "John",
    lastName  : "Doe",
    age       : 50,
    eyeColor  : "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

The object (person) in the example above has 4 properties: **firstName**, **lastName**, **age**, and **eyeColor**.

The **typeof** Operator

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable.

The **typeof** operator returns the type of a variable or an expression:

Example

Java Script Programming Language

168

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof "" + "<br>" +
typeof "John" + "<br>" +
typeof "John Doe";
</script>
</body>
</html>
```

JavaScript typeof

The typeof operator returns the type of a variable or an expression.

string
string
string

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof 0 + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof (3) + "<br>" +
typeof (3 + 4);
</script>

</body>
</html>
```

Java Script Programming Language

169

JavaScript typeof

The **typeof** operator returns the type of a variable or an expression.

```
number  
number  
number  
number  
number
```

Undefined

In **JavaScript**, a variable without a value, has the value **undefined**. The type is also **undefined**.

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript</h2>  
  
<p>The value (and the data type) of a variable with no value is <b>undefined</b>.</p>  
  
<p id="demo"></p>  
  
<script>  
var car;  
document.getElementById("demo").innerHTML =  
car + "<br>" + typeof car;  
</script>  
  
</body>  
</html>
```

JavaScript

The value (and the data type) of a variable with no value is **undefined**.

```
undefined  
undefined
```

Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

Java Script Programming Language

170

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>Variables can be emptied if you set the value to <b>undefined</b>.</p>

<p id="demo"></p>

<script>
var car = "Volvo";
car = undefined;
document.getElementById("demo").innerHTML =
car + "<br>" + typeof car;
</script>

</body>
</html>
```

JavaScript

Variables can be emptied if you set the value to **undefined**.

```
undefined
undefined
```

Empty Values

An **empty value** has nothing to do with undefined.

An **empty string** has both a legal value and a type.

Example

Java Script Programming Language

171

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>An empty string has both a legal value and a type:</p>

<p id="demo"></p>

<script>
var car = "";
document.getElementById("demo").innerHTML =
"The value is: " +
car + "<br>" +
"The type is: " + typeof car;
</script>

</body>
</html>
```

JavaScript

An empty string has both a legal value and a type:

The value is:
The type is: string

Null

In **JavaScript** null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

You can consider it a bug in JavaScript that `typeof null` is an object. It should be `null`.

You can empty an object by setting it to null:

Example

Java Script Programming Language

172

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>Objects can be emptied by setting the value to <b>null</b>. </p>

<p id="demo"></p>

<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;
document.getElementById("demo").innerHTML = typeof person;
</script>

</body>
</html>
```

JavaScript

Objects can be emptied by setting the value to **null**.

object

You can also empty an object by setting it to **undefined**:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>Objects can be emptied by setting the value to <b>undefined</b>. </p>

<p id="demo"></p>

<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = undefined;
document.getElementById("demo").innerHTML = person;
</script>

</body>
</html>
```

Java Script Programming Language

173

JavaScript

Objects can be emptied by setting the value to **undefined**.

undefined

Difference Between Undefined and Null

undefined and **null** are equal in value but different in type:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>Undefined and null are equal in value but different in type:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof undefined + "<br>" +
typeof null + "<br><br>" +
(null === undefined) + "<br>" +
(null == undefined);
</script>

</body>
</html>
```

JavaScript

Undefined and null are equal in value but different in type:

undefined
object

false
true

Primitive Data

A **primitive data** value is a single simple data value with no additional properties and methods.

Java Script Programming Language

174

The **typeof** operator can return one of these primitive types:

- string
- number
- boolean
- undefined

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof "john" + "<br>" +
typeof 3.14 + "<br>" +
typeof true + "<br>" +
typeof false + "<br>" +
typeof x;
</script>

</body>
</html>
```

JavaScript typeof

The **typeof** operator returns the type of a variable or an expression.

```
string
number
boolean
boolean
undefined
```

Complex Data

The **typeof** operator can return one of two complex types:

- function

Java Script Programming

Language

175

- object

The **typeof** operator returns "object" for objects, arrays, and null.

The **typeof** operator does not return "object" for functions.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns object for both objects, arrays, and null.</p>
<p>The typeof operator does not return object for functions.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof {name:'john', age:34} + "<br>" +
typeof [1,2,3,4] + "<br>" +
typeof null + "<br>" +
typeof function myFunc(){}
</script>

</body>
</html>
```

JavaScript typeof

The **typeof** operator returns object for both objects, arrays, and null.

The **typeof** operator does not return object for functions.

```
object
object
object
function
```

The **typeof** operator returns "object" for arrays because in JavaScript arrays are objects.

JavaScript Functions

A **JavaScript** function is a block of code designed to perform a particular task.

A **JavaScript** function is executed when "something" invokes it (calls it).

Example

Java Script Programming Language

176

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation, and returns the result:</p>

<p id="demo"></p>

<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>
```

JavaScript Functions

This example calls a function which performs a calculation, and returns the result:

12

JavaScript Function Syntax

A **JavaScript** function is defined with the `function` keyword, followed by a name, followed by **parentheses ()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
`(parameter1, parameter2, ...)`

The code to be executed, by the function, is placed inside curly brackets: `{ }`

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function parameters are listed inside the parentheses () in the function definition.

Java Script Programming Language

177

Function arguments are the values received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an **event occurs** (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

When **JavaScript** reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

Example

Java Script Programming Language

178

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation and returns the result:</p>

<p id="demo"></p>

<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>

</body>
</html>
```

JavaScript Functions

This example calls a function which performs a calculation and returns the result:

12

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Example

Java Script Programming Language

179

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function to convert from Fahrenheit to Celsius:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
    return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
</script>

</body>
</html>
```

JavaScript Functions

This example calls a function to convert from Fahrenheit to Celsius:

25

The () Operator Invokes the Function

Using the example above, **toCelsius** refers to the function object, and **toCelsius()** refers to the function result.

Accessing a function without () will return the function object instead of the function result.

Example

Java Script Programming Language

180

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>Accessing a function without () will return the function definition instead of the function result:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
    return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius;
</script>

</body>
</html>
```

JavaScript Functions

Accessing a function without () will return the function definition instead of the function result:

```
function toCelsius(f) { return (5/9) * (f-32); }
```

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Java Script Programming Language

181

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"The temperature is " + toCelsius(77) + " Celsius";

function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
</script>

</body>
</html>
```

JavaScript Functions

The temperature is 25 Celsius

Local Variables

Variables declared within a JavaScript function, become LOCAL to the function.

Local variables can only be accessed from within the function.

Example

Java Script Programming Language

182

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>Outside myFunction() carName is undefined.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
myFunction();

function myFunction() {
  var carName = "Volvo";
  document.getElementById("demo1").innerHTML =
    typeof carName + " " + carName;
}

document.getElementById("demo2").innerHTML =
typeof carName;
</script>

</body>
</html>
```

JavaScript Functions

Outside myFunction() carName is undefined.

string Volvo

undefined

Since **local variables** are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

• JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a simple value (Fiat) to a variable named car:

Java Script Programming Language

183

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p id="demo"></p>

<script>
// Create and display a variable:
var car = "Fiat";
document.getElementById("demo").innerHTML = car;
</script>

</body>
</html>
```

JavaScript Variables

Fiat

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

This code assigns many values (Fiat, 500, white) to a variable named car:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
var car = {type:"Fiat", model:"500", color:"white"};

// Display some data from the object:
document.getElementById("demo").innerHTML = "The car type is " + car.type;
</script>

</body>
</html>
```

JavaScript Objects

The car type is Fiat

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

Java Script Programming Language

184

Object Definition

You define (and create) a JavaScript object with an object literal:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

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

Example

Java Script Programming Language

185

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

Object Properties

The **name:values** pairs in JavaScript objects are called properties:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	Blue

Accessing Object Properties

You can access object properties in two ways:

Java Script Programming Language

186

objectName.propertyName

Or

objectName["propertyName"]

Example1

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p>There are two different ways to access an object property.</p>
<p>You can use person.property or person["property"].</p>

<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566
};
// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " " + person.lastName;
</script>

</body>
</html>
```

JavaScript Objects

There are two different ways to access an object property.

You can use `person.property` or `person["property"]`.

John Doe

Example2

Java Script Programming Language

187

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p>There are two different ways to access an object property.</p>
<p>You can use person.property or person["property"].</p>
<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566
};
// Display some data from the object:
document.getElementById("demo").innerHTML =
person["firstName"] + " " + person["lastName"];
</script>

</body>
</html>
```

JavaScript Objects

There are two different ways to access an object property.

You can use person.property or person["property"].

John Doe

Object Methods

Objects can also have methods.

Methods are actions that can be performed on objects.

Methods are stored in properties as function definitions.

Property	Property Value
firstName	John
lastName	Doe

Java Script Programming Language

188

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

Example

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The this Keyword

In a function definition, this refers to the "owner" of the function.

In the example above, this is the person object that "owns" the **fullName** function.

In other words, **this.firstName** means the **firstName** property of this object.

Accessing Object Methods

You access an object method with the following syntax:

objectName.methodName()

Example

Java Script Programming

Language

189

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p>An object method is a function definition, stored as a property value.</p>

<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>

</body>
</html>
```

JavaScript Objects

An object method is a function definition, stored as a property value.

John Doe

If you access a method without the **() parentheses**, it will return the function definition:

Example

Java Script Programming Language

190

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p>If you access an object method without (), it will return the function definition:</p>

<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName;
</script>

</body>
</html>
```

JavaScript Objects

If you access an object method without (), it will return the function definition:

```
function() { return this.firstName + " " + this.lastName; }
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();           // Declares x as a String object
var y = new Number();          // Declares y as a Number object
var z = new Boolean();         // Declares z as a Boolean object
```

Java Script Programming Language

191

Avoid **String**, **Number**, and **Boolean** objects. They complicate your code and slow down execution speed.

• JavaScript Events

HTML events are "things" that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can "react" on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an onclick attribute (with code), is added to a **<button>** element:

Example

Java Script Programming Language

192

```
<!DOCTYPE html>
<html>
<body>

<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>
<p id="demo"></p>

</body>
</html>
```

The time is?

The time is?

Sat May 16 2020 02:38:41 GMT+0300 (Arabian Standard Time)

In the example above, the JavaScript code changes the content of the element with **id="demo"**.

In the next example, the code changes the content of its own element (using `this.innerHTML`):

Example

```
<!DOCTYPE html>
<html>
<body>

<button onclick="this.innerHTML=Date()">The time is?</button>

</body>
</html>
```

The time is?

Sat May 16 2020 02:40:01 GMT+0300 (Arabian Standard Time)

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Java Script Programming Language

193

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

<p id="demo"></p>

</body>
</html>
```

Click the button to display the date.

The time is?

Click the button to display the date.

The time is?

Sat May 16 2020 02:41:44 GMT+0300 (Arabian Standard Time)

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element

Java Script Programming

Language

194

onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

• **JavaScript Strings**

A JavaScript string is zero or more characters written inside quotes.

Example

Java Script Programming Language

195

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p id="demo"></p>

<script>
var x = "John Doe"; // String written inside quotes
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Strings

John Doe

You can use single or double quotes:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings are written inside quotes. You can use single or double quotes:</p>

<p id="demo"></p>

<script>

var carName1 = "Volvo XC60"; // Double quotes
var carName2 = 'Volvo XC60'; // Single quotes

document.getElementById("demo").innerHTML =
carName1 + " " + carName2;

</script>

</body>
</html>
```

Java Script Programming Language

196

JavaScript Strings

Strings are written inside quotes. You can use single or double quotes:

Volvo XC60 Volvo XC60

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>You can use quotes inside a string, as long as they don't match the quotes
surrounding the string.</p>

<p id="demo"></p>

<script>
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';

document.getElementById("demo").innerHTML =
answer1 + "<br>" + answer2 + "<br>" + answer3;
</script>

</body>
</html>
```

JavaScript Strings

You can use quotes inside a string, as long as they don't match the quotes surrounding the string.

It's alright
He is called 'Johnny'
He is called "Johnny"

String Length

To find the length of a string, use the built-in length property:

Java Script Programming

Language

197

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Properties</h2>

<p>The length property returns the length of a string:</p>

<p id="demo"></p>

<script>
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
</script>

</body>
</html>
```

JavaScript String Properties

The length property returns the length of a string:

26

Escape Character

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
var x = "We are the so-called "Vikings" from the north.;"
```

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the backslash escape character.

The backslash (\) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\\	\	Backslash

The sequence \" inserts a double quote in a string:

Java Script Programming Language

198

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>The escape sequence \" inserts a double quote in a string.</p>

<p id="demo"></p>

<script>
var x = "We are the so-called \"Vikings\" from the north.";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Strings

The escape sequence \" inserts a double quote in a string.

We are the so-called "Vikings" from the north.

The sequence \' inserts a single quote in a string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>The escape sequence \'' inserts a single quote in a string.</p>

<p id="demo"></p>

<script>
var x = 'It\'s alright.';
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

199

JavaScript Strings

The escape sequence \ inserts a single quote in a string.

It's alright.

The sequence \\ inserts a backslash in a string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>The escape sequence \\ inserts a backslash in a string.</p>

<p id="demo"></p>

<script>
var x = "The character \\ is called backslash.";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Strings

The escape sequence \\ inserts a backslash in a string.

The character \\ is called backslash.

Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line

Java Script Programming Language

200

\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

Breaking Long Code Lines

For best readability, programmers often like to avoid code lines longer than 80 characters. If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>
The best place to break a code line is after an operator or a comma.
</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Hello Dolly!";
</script>

</body>
</html>
```

JavaScript Statements

The best place to break a code line is after an operator or a comma.

Hello Dolly!

You can also break up a code line within a text string with a single backslash:

Example

Java Script Programming Language

201

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>
You can break a code line within a text string with a backslash.
</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello \
Dolly!";
</script>

</body>
</html>
```

JavaScript Strings

You can break a code line within a text string with a backslash.

Hello Dolly!

The \ method is not the preferred method. It might not have universal support.

Some browsers do not allow spaces behind the \ character.

A safer way to break up a string, is to use string addition:

Example

Java Script Programming Language

202

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>The safest way to break a code line in a string is using string addition.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello " +
"Dolly!";
</script>

</body>
</html>
```

JavaScript Strings

The safest way to break a code line in a string is using string addition.

Hello Dolly!

You cannot break up a code line with a backslash:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p id="demo">You cannot break a code line with a \ backslash.</p>

<script>
document.getElementById("demo").innerHTML = \
"Hello Dolly.";
</script>

</body>
</html>
```

JavaScript Statements

You cannot break a code line with a \ backslash.

Java Script Programming Language

203

Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

```
var firstName = "John";
```

But strings can also be defined as objects with the keyword new:

```
var firstName = new String("John");
```

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var x = "John";           // x is a string
var y = new String("John"); // y is an object

document.getElementById("demo").innerHTML =
typeof x + "<br>" + typeof y;
</script>

</body>
</html>
```

string
object

Don't create strings as objects. It slows down execution speed. The new keyword complicates the code. This can produce some unexpected results:

When using the == operator, equal strings are equal:

Example

Java Script Programming Language

204

```
<!DOCTYPE html>
<html>
<body>

<h2>Never Create Strings as objects.</h2>
<p>Strings and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = "John";           // x is a string
var y = new String("John"); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

Never Create Strings as objects.

Strings and objects cannot be safely compared.

true

When using the `==` operator, equal strings are not equal, because the `==` operator expects equality in both type and value.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Never Create Strings as objects.</h2>
<p>Strings and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = "John";           // x is a string
var y = new String("John"); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

Java Script Programming Language

205

Never Create Strings as objects.

Strings and objects cannot be safely compared.

false

Note the difference between `(x==y)` and `(x===y)`. Comparing two JavaScript objects will **always** return `false`.

• JavaScript String Methods

String methods help you to work with strings.

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The length property returns the length of a string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Properties</h2>

<p>The length property returns the length of a string:</p>

<p id="demo"></p>

<script>
var txt = "ABCDEFGHIJKLMNPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
</script>

</body>
</html>
```

Java Script Programming Language

206

JavaScript String Properties

The `length` property returns the length of a string:

26

Finding a String in a String

The `indexOf()` method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method returns the position of the first occurrence of a specified text:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The `indexOf()` method returns the position of the first occurrence of a specified text:

7

The `lastIndexOf()` method returns the index of the last occurrence of a specified text in a string:

Example

Java Script Programming Language

207

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The lastIndexOf() method returns the position of the last occurrence of a specified text:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The lastIndexOf() method returns the position of the last occurrence of a specified text:

21

Both **indexOf()**, and **lastIndexOf()** return -1 if the text is not found.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>Both indexOf(), and lastIndexOf() return -1 if the text is not found:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("John");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

Java Script Programming Language

208

JavaScript String Methods

Both `indexOf()`, and `lastIndexOf()` return -1 if the text is not found:

-1

Both methods accept a second parameter as the starting position for the search:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method accepts a second parameter as the starting position for the search:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate",15);
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The `indexOf()` method accepts a second parameter as the starting position for the search:

21

The `lastIndexOf()` methods searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.

Example

Java Script Programming Language

209

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The lastIndexOf() method accepts a second parameter as the starting position for the search.</p>
<p>Remember that the lastIndexOf() method searches backwards, so position 15 means start the search at position 15, and search to the beginning.</p>
<p>Position 15 is position 15 from the beginning.</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate", 15);
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The `lastIndexOf()` method accepts a second parameter as the starting position for the search.

Remember that the `lastIndexOf()` method searches backwards, so position 15 means start the search at position 15, and search to the beginning.

Position 15 is position 15 from the beginning.

7

Searching for a String in a String

The `search()` method searches a string for a specified value and returns the position of the match:

Example

Java Script Programming Language

210

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The search() method returns the position of the first occurrence of a specified text in a string:</p>

<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
document.getElementById("demo").innerHTML = pos;
</script>

</body>
</html>
```

JavaScript String Methods

The search() method returns the position of the first occurrence of a specified text in a string:

7

Did You Notice?

The two methods, **indexOf()** and **search()**, are equal?

They accept the same arguments (parameters), and return the same value?

The two methods are NOT equal. These are the differences:

- The **search()** method cannot take a second start position argument.
- The **indexOf()** method cannot take powerful search values (regular expressions).

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`
- `substr(start, length)`

Java Script Programming Language

211

The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana

Remember: JavaScript counts positions from zero. First position is 0.

If a parameter is negative, the position is counted from the end of the string

This example slices out a portion of a string from position -12 to position -6:

Example

Java Script Programming Language

212

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12,-6);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana

If you omit the second parameter, the method will slice out the rest of the string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(7);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

Java Script Programming Language

213

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana, Kiwi

or, counting from the end:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12)
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana, Kiwi

Example

Java Script Programming Language

214

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The substr() method extract a part of a string  
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.substring(7,13);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The substr() method extract a part of a string and returns the extracted parts in a new string:

Banana

If you omit the second parameter, **substring()** will slice out the rest of the string.

The substr() Method

substr() is similar to **slice()**.

The difference is that the second parameter specifies the length of the extracted part.

Example

Java Script Programming Language

215

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The substr() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.substr(7,6);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The substr() method extract a part of a string and returns the extracted parts in a new string:

Banana

If you omit the second parameter, **substr()** will slice out the rest of the string.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The substr() method extract a part of a string
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.substr(7);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

Java Script Programming Language

216

JavaScript String Methods

The substr() method extract a part of a string and returns the extracted parts in a new string:

Banana, Kiwi

If the first parameter is negative, the position counts from the end of the string.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The substr() method extract a part of a string  
and returns the extracted parts in a new string:</p>

<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.substr(-4);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

JavaScript String Methods

The substr() method extract a part of a string and returns the extracted parts in a new string:

Kiwi

Replacing String Content

The **replace()** method replaces a specified value with another value in a string:

Example

Java Script Programming Language

217

```
<!DOCTYPE html>
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>Replace "Microsoft" with "W3Schools" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft!</p>

<script>
function myFunction() {
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace("Microsoft","W3Schools");
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

JavaScript String Methods

Replace "Microsoft" with "W3Schools" in the paragraph below:

Try it

Please visit Microsoft!

JavaScript String Methods

Replace "Microsoft" with "W3Schools" in the paragraph below:

Try it

Please visit W3Schools!

The **replace()** method does not change the string it is called on. It returns a new string.

By **default**, the **replace()** method replaces only the first match:

Example

Java Script Programming Language

218

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>Replace "Microsoft" with "W3Schools" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft and Microsoft!</p>

<script>
function myFunction() {
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace("Microsoft", "W3Schools");
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

JavaScript String Methods

Replace "Microsoft" with "W3Schools" in the paragraph below:

Try it

Please visit Microsoft and Microsoft!

JavaScript String Methods

Replace "Microsoft" with "W3Schools" in the paragraph below:

Try it

Please visit W3Schools and Microsoft!

By default, the **replace ()** method is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example

Java Script Programming Language

219

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>Try to replace "Microsoft" with "W3Schools" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft!</p>

<script>
function myFunction() {
    var str = document.getElementById("demo").innerHTML;
    var txt = str.replace("MICROSOFT","W3Schools");
    document.getElementById("demo").innerHTML = txt;
}
</script>

<p><strong>Note:</strong> Nothing will happen. By default, the replace() method is case sensitive. Writing MICROSOFT (with upper-case) will not work.</p>

</body>
</html>
```

JavaScript String Methods

Try to replace "Microsoft" with "W3Schools" in the paragraph below:

[Try it](#)

Please visit Microsoft!

Note: Nothing will happen. By default, the replace() method is case sensitive. Writing MICROSOFT (with upper-case) will not work.

To replace case insensitive, use a regular expression with an **i** flag (insensitive):

Example

Java Script Programming Language

220

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>Replace "Microsoft" with "W3Schools" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft!</p>

<script>
function myFunction() {
    var str = document.getElementById("demo").innerHTML;
    var txt = str.replace(/MICROSOFT/i,"W3Schools");
    document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

JavaScript String Methods

Replace "Microsoft" with "W3Schools" in the paragraph below:

Try it

Please visit Microsoft!

JavaScript String Methods

Replace "Microsoft" with "W3Schools" in the paragraph below:

Try it

Please visit W3Schools!

To replace all matches, use a regular expression with a **/g** flag (global match):

Example

Java Script Programming Language

221

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>Replace all occurrences of "Microsoft" with "W3Schools" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft and Microsoft!</p>

<script>
function myFunction() {
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace(/Microsoft/g,"W3Schools");
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

JavaScript String Methods

Replace all occurrences of "Microsoft" with "W3Schools" in the paragraph below:

[Try it](#)

Please visit Microsoft and Microsoft!

JavaScript String Methods

Replace all occurrences of "Microsoft" with "W3Schools" in the paragraph below:

[Try it](#)

Please visit W3Schools and W3Schools!

Converting to Upper and Lower Case

A string is converted to upper case with **toUpperCase()**:

Example

Java Script Programming

Language

222

```
<!DOCTYPE html>
<html>
<body>

<p>Convert string to upper case:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Hello World!</p>

<script>
function myFunction() {
  var text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML = text.toUpperCase();
}
</script>

</body>
</html>
```

Convert string to upper case:

[Try it](#)

Hello World!

Convert string to upper case:

[Try it](#)

HELLO WORLD!

A string is converted to lower case with **toLowerCase()**:

Example

Java Script Programming Language

223

```
<!DOCTYPE html>
<html>
<body>

<p>Convert string to lower case:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Hello World!</p>

<script>
function myFunction() {
  var text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML = text.toLowerCase();
}
</script>

</body>
</html>
```

Convert string to lower case:

[Try it](#)

Hello World!

Convert string to lower case:

[Try it](#)

hello world!

The concat() Method

concat() joins two or more strings:

Example

Java Script Programming Language

224

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The concat() method joins two or more strings:</p>

<p id="demo"></p>

<script>
var text1 = "Hello";
var text2 = "World!";
var text3 = text1.concat(" ",text2);
document.getElementById("demo").innerHTML = text3;
</script>

</body>
</html>
```

JavaScript String Methods

The concat() method joins two or more strings:

Hello World!

The **concat()** method can be used instead of the plus operator. These two lines do the same:

Example

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

String.trim()

The **trim()** method removes whitespace from both sides of a string:

Example

Java Script Programming Language

225

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String.trim()</h2>

<p>Click the button to alert the string with removed whitespace.</p>

<button onclick="myFunction()">Try it</button>

<p><strong>Note:</strong> The trim() method is not supported in Internet Explorer 8 and earlier versions.</p>

<script>
function myFunction() {
  var str = "    Hello World!    ";
  alert(str.trim());
}
</script>

</body>
</html>
```

JavaScript String.trim()

Click the button to alert the string with removed whitespace.

Try it

Note: The trim() method is not supported in Internet Explorer 8 and earlier versions.



The **trim()** method is not supported in Internet Explorer 8 or lower.

If you need to support IE 8, you can use **replace()** with a regular expression instead:

Example

Java Script Programming Language

226

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String.trim()</h2>

<p>IE 8 does not support String.trim(). To trim a string you can use a regular expression instead.</p>

<script>
var str = "    Hello World!      ";
alert(str.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, ''));

</script>

</body>
</html>
```

JavaScript String.trim()

IE 8 does not support String.trim(). To trim a string you can use a regular expression instead.

You can also use the replace solution above to add a trim function to the JavaScript **String.prototype**:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String.trim()</h2>

<p>IE 8 does not support String.trim(). To trim a string you can use a polyfill.</p>

<script>
if (!String.prototype.trim) {
  String.prototype.trim = function () {
    return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, '');
  };
}
var str = "    Hello World!      ";
alert(str.trim());
</script>

</body>
</html>
```

Java Script Programming Language

227

An embedded page on this page says

Hello World!

OK

JavaScript String.trim()

IE 8 does not support String.trim(). To trim a string you can use a polyfill.

Extracting String Characters

There are 3 methods for extracting string characters:

- charAt(position)
- charCodeAt(position)
- Property access []

The charAt() Method

The **charAt()** method returns the character at a specified index (position) in a string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The charAt() method returns the character at a given position in a string:</p>
<p id="demo"></p>

<script>
var str = "HELLO WORLD";
document.getElementById("demo").innerHTML = str.charAt(0);
</script>

</body>
</html>
```

Java Script Programming Language

228

JavaScript String Methods

The `charAt()` method returns the character at a given position in a string:

H

The `charCodeAt()` Method

The `charCodeAt()` method returns the **unicode** of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example

```
<!DOCTYPE html>
<html>
<body>

<p>The charCodeAt() method returns the unicode of the character at a given position in a string:</p>

<p id="demo"></p>

<script>
var str = "HELLO WORLD";
document.getElementById("demo").innerHTML = str.charCodeAt(0);
</script>

</body>
</html>
```

The `charCodeAt()` method returns the unicode of the character at a given position in a string:

72

Property Access

ECMAScript 5 (2009) allows property access [] on strings:

Example

Java Script Programming Language

229

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>ECMAScript 5 allows property access on strings:</p>

<p id="demo"></p>

<script>
var str = "HELLO WORLD";
document.getElementById("demo").innerHTML = str[0];
</script>
</body>
</html>
```

JavaScript String Methods

ECMAScript 5 allows property access on strings:

H

Property access might be a little **unpredictable**:

- It does not work in Internet Explorer 7 or earlier
- It makes strings look like arrays (but they are not)
- If no character is found, [] returns undefined, while charAt() returns an empty string.
- It is read only. str[0] = "A" gives no error (but does not work!)

Example

Java Script Programming Language

230

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>ECMAScript 5 allows property acces on strings. but read only:</p>

<p id="demo"></p>

<script>
var str = "HELLO WORLD";
str[0] = "A"; // Does not work
document.getElementById("demo").innerHTML = str[0];
</script>
</body>
</html>
```

JavaScript String Methods

ECMAScript 5 allows property acces on strings. but read only:

H

If you want to work with a string as an array, you can convert it to an array.

Converting a String to an Array

A string can be converted to an array with the **split()** method:

Example

Java Script Programming Language

231

```
<!DOCTYPE html>
<html>
<body>

<p>Click "Try it" to display the first array element, after a string split.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var str = "a,b,c,d,e,f";
  var arr = str.split(",");
  document.getElementById("demo").innerHTML = arr[0];
}
</script>

</body>
</html>
```

Click "Try it" to display the first array element, after a string split.

Try it

Click "Try it" to display the first array element, after a string split.

Try it

a

If the separator is omitted, the returned array will contain the whole string in **index [0]**.

If the separator is "", the returned array will be an array of single characters:

Example

Java Script Programming Language

232

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var str = "Hello";
var arr = str.split("");
var text = "";
var i;
for (i = 0; i < arr.length; i++) {
  text += arr[i] + "<br>"
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

H
e
l
l
o

- **JavaScript Numbers**

JavaScript has only one type of number. Numbers can be written with or without decimals.

Example

Java Script Programming Language

233

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Numbers can be written with or without decimals:</p>

<p id="demo"></p>

<script>
var x = 3.14;
var y = 3;
document.getElementById("demo").innerHTML = x + "<br>" + y;
</script>

</body>
</html>
```

JavaScript Numbers

Numbers can be written with or without decimals:

3.14
3

Extra-large or **extra small** numbers can be written with scientific (exponent) notation:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Extra large or extra small numbers can be written with scientific (exponent)
notation:</p>

<p id="demo"></p>

<script>
var x = 123e5;
var y = 123e-5;
document.getElementById("demo").innerHTML = x + "<br>" + y;
</script>

</body>
</html>
```

Java Script Programming Language

234

JavaScript Numbers

Extra large or extra small numbers can be written with scientific (exponent) notation:

12300000
0.00123

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international **IEEE 754** standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

Value (aka Fraction/Mantissa)	Exponent	Sign
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

Precision

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

Example

Java Script Programming Language

235

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Integers (numbers without a period or exponent notation) are accurate up to 15 digits:</p>

<p id="demo"></p>

<script>
var x = 999999999999999;
var y = 999999999999999;
document.getElementById("demo").innerHTML = x + "<br>" + y;
</script>

</body>
</html>
```

JavaScript Numbers

Integers (numbers without a period or exponent notation) are accurate up to 15 digits:

999999999999999
1000000000000000

The maximum number of decimals is 17, but **floating-point** arithmetic is not always 100% accurate:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Floating point arithmetic is not always 100% accurate.</p>

<p id="demo"></p>

<script>
var x = 0.2 + 0.1;
document.getElementById("demo").innerHTML = "0.2 + 0.1 = " + x;
</script>

</body>
</html>
```

Java Script Programming Language

236

JavaScript Numbers

Floating point arithmetic is not always 100% accurate.

$0.2 + 0.1 = 0.30000000000000004$

To solve the problem above, it helps to multiply and divide:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Floating point arithmetic is not always 100% accurate:</p>
<p id="demo1"></p>

<p>But it helps to multiply and divide:</p>

<p id="demo2"></p>

<script>
var x = 0.2 + 0.1;
document.getElementById("demo1").innerHTML = "0.2 + 0.1 = " + x;
var y = (0.2*10 + 0.1*10) / 10;
document.getElementById("demo2").innerHTML = "0.2 + 0.1 = " + y;
</script>

</body>
</html>
```

JavaScript Numbers

Floating point arithmetic is not always 100% accurate:

$0.2 + 0.1 = 0.30000000000000004$

But it helps to multiply and divide:

$0.2 + 0.1 = 0.3$

Adding Numbers and Strings

WARNING!!

JavaScript uses the **+** operator for both addition and concatenation.

Java Script Programming Language

237

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>If you add two numbers, the result will be a number:</p>

<p id="demo"></p>

<script>
var x = 10;
var y = 20;
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Numbers

If you add two numbers, the result will be a number:

30

If you add two strings, the result will be a string concatenation:

Example

Java Script Programming Language

238

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>If you add two numeric strings, the result will be a concatenated string:</p>

<p id="demo"></p>

<script>
var x = "10";
var y = "20";
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Numbers

If you add two numeric strings, the result will be a concatenated string:

1020

If you add a number and a string, the result will be a string concatenation:

Example

Java Script Programming

Language

239

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>If you add a number and a numeric string, the result will be a concatenated string:</p>

<p id="demo"></p>

<script>
var x = 10;
var y = "20";
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Numbers

If you add a number and a numeric string, the result will be a concatenated string:

1020

If you add a string and a number, the result will be a string concatenation:

Example

Java Script Programming Language

240

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>If you add a numeric string and a number, the result will be a concatenated string:</p>

<p id="demo"></p>

<script>
var x = "10";
var y = 20;
document.getElementById("demo").innerHTML =
"The result is: " + x + y;
</script>

</body>
</html>
```

JavaScript Numbers

If you add a numeric string and a number, the result will be a concatenated string:

The result is: 1020

A common mistake is to expect this result to be 30:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>A common mistake is to expect this result to be 30:</p>

<p id="demo"></p>

<script>
var x = 10;
var y = 20;
document.getElementById("demo").innerHTML =
"The result is: " + x + y;
</script>

</body>
</html>
```

Java Script Programming Language

241

JavaScript Numbers

A common mistake is to expect this result to be 30:

The result is: 1020

A common mistake is to expect this result to be 102030:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>A common mistake is to expect this result to be 102030:</p>

<p id="demo"></p>

<script>
var x = 10;
var y = 20;
var z = "30";
var result = x + y + z;
document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

JavaScript Numbers

A common mistake is to expect this result to be 102030:

3030

Numeric Strings

JavaScript strings can have numeric content:

```
var x = 100;           // x is a number
```

```
var y = "100";         // y is a string
```

Java Script Programming

Language

242

JavaScript will try to convert strings to numbers in all numeric operations:

This will work:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>JavaScript will try to convert strings to numbers when dividing:</p>

<p id="demo"></p>

<script>
var x = "100";
var y = "10";
var z = x / y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Numbers

JavaScript will try to convert strings to numbers when dividing:

10

This will also work:

Java Script Programming Language

243

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>JavaScript will try to convert strings to numbers when multiplying:</p>

<p id="demo"></p>

<script>
var x = "100";
var y = "10";
var z = x * y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Numbers

JavaScript will try to convert strings to numbers when multiplying:

1000

And this will work:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>JavaScript will try to convert strings to numbers when subtracting:</p>

<p id="demo"></p>

<script>
var x = "100";
var y = "10";
var z = x - y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

Java Script Programming Language

244

JavaScript Numbers

JavaScript will try to convert strings to numbers when subtracting:

90

But this will not work:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>JavaScript will NOT convert strings to numbers when adding:</p>

<p id="demo"></p>

<script>
var x = "100";
var y = "10";
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

JavaScript Numbers

JavaScript will NOT convert strings to numbers when adding:

10010

In the last example JavaScript uses the + operator to concatenate the strings.

NaN - Not a Number

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in **NaN** (Not a Number):

Example

Java Script Programming Language

245

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>A number divided by a non-numeric string becomes NaN (Not a Number):</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 100 / "Apple";
</script>

</body>
</html>
```

JavaScript Numbers

A number divided by a non-numeric string becomes NaN (Not a Number):

NaN

However, if the string contains a numeric value , the result will be a number:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>A number divided by a numeric string becomes a number:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 100 / "10";
</script>

</body>
</html>
```

Java Script Programming Language

246

JavaScript Numbers

A number divided by a numeric string becomes a number:

10

You can use the global JavaScript function **isNaN()** to find out if a value is a number:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>You can use the global JavaScript function isNaN() to find out if a value is a
number:</p>

<p id="demo"></p>

<script>
var x = 100 / "Apple";
document.getElementById("demo").innerHTML = isNaN(x);
</script>

</body>
</html>
```

JavaScript Numbers

You can use the global JavaScript function **isNaN()** to find out if a value is a number:

true

Watch out for **NaN**. If you use **NaN** in a mathematical operation, the result will also be **NaN**:

Example

Java Script Programming Language

247

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>If you use NaN in a mathematical operation, the result will also be NaN:</p>

<p id="demo"></p>

<script>
var x = NaN;
var y = 5;
document.getElementById("demo").innerHTML = x + y;
</script>

</body>
</html>
```

JavaScript Numbers

If you use NaN in a mathematical operation, the result will also be NaN:

NaN

Or the result might be a concatenation:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>If you use NaN in a mathematical operation, the result can be a concatenation:</p>

<p id="demo"></p>

<script>
var x = NaN;
var y = "5";
document.getElementById("demo").innerHTML = x + y;
</script>

</body>
</html>
```

Java Script Programming Language

248

JavaScript Numbers

If you use NaN in a mathematical operation, the result can be a concatenation:

NaN5

NaN is a number: **typeof NaN** returns number:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>The typeof NaN is number:</p>

<p id="demo"></p>

<script>
var x = NaN;
document.getElementById("demo").innerHTML = typeof x;
</script>

</body>
</html>
```

JavaScript Numbers

The typeof NaN is number:

number

Infinity

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

Example

Java Script Programming

Language

249

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Infinity is returned if you calculate a number outside the largest possible number:
</p>

<p id="demo"></p>

<script>
var myNumber = 2;
var txt = "";
while (myNumber != Infinity) {
    myNumber = myNumber * myNumber;
    txt = txt + myNumber + "<br>";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

JavaScript Numbers

Infinity is returned if you calculate a number outside the largest possible number:

```
4
16
256
65536
4294967296
18446744073709552000
3.402823669209385e+38
1.157920892373162e+77
1.3407807929942597e+154
Infinity
```

Division by 0 (zero) also generates **Infinity**:

Example

Java Script Programming Language

250

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Division by zero generates Infinity;</p>

<p id="demo"></p>

<script>
var x = 2/0;
var y = -2/0;
document.getElementById("demo").innerHTML = x + "<br>" + y;
</script>

</body>
</html>
```

JavaScript Numbers

Division by zero generates Infinity;

Infinity
-Infinity

Infinity is a number: **typeof** Infinity returns number.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Infinity is a number:</p>

<p id="demo"></p>

<script>
var x = Infinity;
document.getElementById("demo").innerHTML = typeof x;
</script>

</body>
</html>
```

Java Script Programming Language

251

JavaScript Numbers

Infinity is a number:

number

Hexadecimal

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Numeric constants, preceded by 0x, are interpreted as hexadecimal:</p>

<p id="demo"></p>

<script>
var x = 0xFF;
document.getElementById("demo").innerHTML = "0xFF = " + x;
</script>

</body>
</html>
```

JavaScript Numbers

Numeric constants, preceded by 0x, are interpreted as hexadecimal:

0xFF = 255

By default, JavaScript displays numbers as base 10 decimals.

But you can use the **toString()** method to output numbers from base 2 to base 36.

Hexadecimal is base 16. Decimal is base 10. Octal is base 8. Binary is base 2.

Example

Java Script Programming Language

252

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>The toString() method can output numbers from base 2 to 36:</p>

<p id="demo"></p>

<script>
var myNumber = 32;
document.getElementById("demo").innerHTML =
"32 = " + "<br>" +
" Decimal " + myNumber.toString(10) + "<br>" +
" Hexadecimal " + myNumber.toString(16) + "<br>" +
" Octal " + myNumber.toString(8) + "<br>" +
" Binary " + myNumber.toString(2);
</script>

</body>
</html>
```

JavaScript Numbers

The `toString()` method can output numbers from base 2 to 36:

```
32 =
Decimal 32
Hexadecimal 20
Octal 40
Binary 100000
```

Numbers Can be Objects

Normally JavaScript numbers are primitive values created from literals:

```
var x = 123;
```

But numbers can also be defined as objects with the keyword new:

```
var y = new Number(123);
```

Example

Java Script Programming Language

253

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>A number can be an object, but there is no need to create a number as an object.</p>

<p id="demo"></p>

<script>
var x = 123;
var y = new Number(123);
document.getElementById("demo").innerHTML = typeof x + "<br>" + typeof y;
</script>

</body>
</html>
```

JavaScript Numbers

A number can be an object, but there is no need to create a number as an object.

number
object

Do not create Number objects. It slows down execution speed.

The new keyword complicates the code. This can produce some unexpected results:

When using the == operator, equal numbers are equal:

Example

Java Script Programming Language

254

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Never create numbers as objects.</p>
<p>Numbers and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = 500;          // x is a number
var y = new Number(500); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

JavaScript Numbers

Never create numbers as objects.

Numbers and objects cannot be safely compared.

true

When using the `==` operator, equal numbers are not equal, because the `==` operator expects equality in both type and value.

Example

Java Script Programming Language

255

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Never create numbers as objects.</p>
<p>Numbers and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = 500;          // x is a number
var y = new Number(500); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

JavaScript Numbers

Never create numbers as objects.

Numbers and objects cannot be safely compared.

false

Or even worse. Objects cannot be compared:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Never create numbers as objects.</p>
<p>JavaScript objects cannot be compared.</p>

<p id="demo"></p>

<script>
var x = new Number(500); // x is an object
var y = new Number(500); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

Java Script Programming Language

256

JavaScript Numbers

Never create numbers as objects.

JavaScript objects cannot be compared.

`false`

Note the difference between `(x==y)` and `(x=====y)`. Comparing two JavaScript objects will always return `false`.

• JavaScript Number Methods

Number methods help you work with numbers.

Number Methods and Properties

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

The `toString()` Method

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

Java Script Programming Language

257

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toString() method converts a number to a string.</p>

<p id="demo"></p>

<script>
var x = 123;
document.getElementById("demo").innerHTML =
  x.toString() + "<br>" +
  (123).toString() + "<br>" +
  (100 + 23).toString();
</script>

</body>
</html>
```

JavaScript Number Methods

The `toString()` method converts a number to a string.

123
123
123

The `toExponential()` Method

`toExponential()` returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

Example

Java Script Programming Language

258

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toExponential() method returns a string, with the number rounded and written using exponential notation.</p>
<p>An optional parameter defines the number of digits behind the decimal point.</p>

<p id="demo"></p>

<script>
var x = 9.656;
document.getElementById("demo").innerHTML =
  x.toExponential() + "<br>" +
  x.toExponential(2) + "<br>" +
  x.toExponential(4) + "<br>" +
  x.toExponential(6);
</script>

</body>
</html>
```

JavaScript Number Methods

The `toExponential()` method returns a string, with the number rounded and written using exponential notation.

An optional parameter defines the number of digits behind the decimal point.

```
9.656e+0
9.66e+0
9.6560e+0
9.656000e+0
```

The parameter is optional. If you don't specify it, JavaScript will not round the number.

The `toFixed()` Method

`toFixed()` returns a string, with the number written with a specified number of decimals:

Example

Java Script Programming Language

259

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toFixed() method rounds a number to a given number of digits.</p>
<p>For working with money, toFixed(2) is perfect.</p>

<p id="demo"></p>

<script>
var x = 9.656;
document.getElementById("demo").innerHTML =
  x.toFixed(0) + "<br>" +
  x.toFixed(2) + "<br>" +
  x.toFixed(4) + "<br>" +
  x.toFixed(6);
</script>

</body>
</html>
```

JavaScript Number Methods

The toFixed() method rounds a number to a given number of digits.

For working with money, toFixed(2) is perfect.

10
9.66
9.6560
9.656000

toFixed(2) is perfect for working with money.

The toPrecision() Method

toPrecision() returns a string, with a number written with a specified length:

Example

Java Script Programming

Language

260

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toPrecision() method returns a string, with a number written with a specified length:</p>

<p id="demo"></p>

<script>
var x = 9.656;
document.getElementById("demo").innerHTML =
  x.toPrecision() + "<br>" +
  x.toPrecision(2) + "<br>" +
  x.toPrecision(4) + "<br>" +
  x.toPrecision(6);
</script>

</body>
</html>
```

JavaScript Number Methods

The toPrecision() method returns a string, with a number written with a specified length:

9.656
9.7
9.656
9.65600

The valueOf() Method

valueOf() returns a number as a number.

Example

Java Script Programming Language

261

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The valueOf() method returns a number as a number:</p>

<p id="demo"></p>

<script>
var x = 123;

document.getElementById("demo").innerHTML =
  x.valueOf() + "<br>" +
  (123).valueOf() + "<br>" +
  (100 + 23).valueOf();
</script>

</body>
</html>
```

JavaScript Number Methods

The valueOf() method returns a number as a number:

```
123
123
123
```

In JavaScript, a number can be a primitive value (**typeof = number**) or an object (**typeof = object**).

The **valueOf()** method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

All JavaScript data types have a **valueOf()** and a **toString()** method.

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

- The **Number()** method
- The **parseInt()** method

Java Script Programming Language

262

- The parseFloat() method

These methods are not number methods, but global JavaScript methods.

Global JavaScript Methods

JavaScript global methods can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

The Number() Method

Number() can be used to convert JavaScript variables to numbers:

Example

Java Script Programming Language

263

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Global Methods</h2>

<p>The Number() method converts variables to numbers:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  Number(true) + "<br>" +
  Number(false) + "<br>" +
  Number("10") + "<br>" +
  Number(" 10 ") + "<br>" +
  Number("10   ") + "<br>" +
  Number(" 10   ") + "<br>" +
  Number("10.33") + "<br>" +
  Number("10,33") + "<br>" +
  Number("10 33") + "<br>" +
  Number("John");
</script>

</body>
</html>
```

JavaScript Global Methods

The Number() method converts variables to numbers:

```
1
0
10
10
10
10
10
10.33
NaN
NaN
NaN
```

If the number cannot be converted, **NaN** (Not a Number) is returned.

The Number() Method Used on Dates

Number() can also convert a date to a number:

Example

Java Script Programming Language

264

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Global Methods</h2>

<p>The Number() method can convert a date to a number:</p>

<p id="demo"></p>

<script>
var x = new Date("2017-09-30");
document.getElementById("demo").innerHTML = Number(x);
</script>

</body>
</html>
```

JavaScript Global Methods

The Number() method can convert a date to a number:

1506729600000

The Number() method above returns the number of milliseconds since 1.1.1970.

The parseInt() Method

parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

Java Script Programming Language

265

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Global Functions</h2>

<p>The global JavaScript function parseInt() converts strings to numbers:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  parseInt("10") + "<br>" +
  parseInt("10.33") + "<br>" +
  parseInt("10 6") + "<br>" +
  parseInt("10 years") + "<br>" +
  parseInt("years 10");
</script>

</body>
</html>
```

JavaScript Global Functions

The global JavaScript function parseInt() converts strings to numbers:

```
10
10
10
10
NaN
```

If the number cannot be converted, **NaN** (Not a Number) is returned.

The parseFloat() Method

parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

Java Script Programming Language

266

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Global Methods</h2>

<p>The parseFloat() method converts strings to numbers:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  parseFloat("10") + "<br>" +
  parseFloat("10.33") + "<br>" +
  parseFloat("10 6") + "<br>" +
  parseFloat("10 years") + "<br>" +
  parseFloat("years 10");
</script>

</body>
</html>
```

JavaScript Global Methods

The parseFloat() method converts strings to numbers:

```
10
10.33
10
10
NaN
```

If the number cannot be converted, **NaN** (Not a Number) is returned.

Number Properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
POSITIVE_INFINITY	Represents infinity (returned on overflow)
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)

Java Script Programming Language

267

NaN

Represents a "Not-a-Number" value

JavaScript MIN_VALUE and MAX_VALUE

MAX_VALUE returns the largest possible number in JavaScript.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>MAX_VALUE returns the largest possible number in JavaScript.</p>

<p id="demo"></p>

<script>
var x = Number.MAX_VALUE;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Number Object Properties

MAX_VALUE returns the largest possible number in JavaScript.

1.7976931348623157e+308

MIN_VALUE returns the lowest possible number in JavaScript.

Example

Java Script Programming Language

268

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>MIN_VALUE returns the smallest number possible in JavaScript.</p>

<p id="demo"></p>

<script>
var x = Number.MIN_VALUE;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Number Object Properties

MIN_VALUE returns the smallest number possible in JavaScript.

5e-324

JavaScript POSITIVE_INFINITY

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>POSITIVE_INFINITY</p>

<p id="demo"></p>

<script>
var x = Number.POSITIVE_INFINITY;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

269

JavaScript Number Object Properties

POSITIVE_INFINITY

Infinity

POSITIVE_INFINITY is returned on overflow:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>POSITIVE_INFINITY is returned on overflow:</p>

<p id="demo"></p>

<script>
var x = 1 / 0;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Number Object Properties

POSITIVE_INFINITY is returned on overflow:

Infinity

JavaScript NEGATIVE_INFINITY

Example

Java Script Programming Language

270

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>NEGATIVE_INFINITY</p>

<p id="demo"></p>

<script>
var x = Number.NEGATIVE_INFINITY;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Number Object Properties

NEGATIVE_INFINITY

-Infinity

NEGATIVE_INFINITY is returned on overflow:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>NEGATIVE_INFINITY</p>

<p id="demo"></p>

<script>
var x = -1 / 0;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Java Script Programming Language

271

JavaScript Number Object Properties

NEGATIVE_INFINITY

-Infinity

JavaScript NaN - Not a Number

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Number.NaN;
</script>

</body>
</html>
```

JavaScript Number Object Properties

NaN

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in **NaN** (Not a Number):

Example

Java Script Programming Language

272

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>A number divided by a non-numeric string becomes NaN (Not a Number):</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 100 / "Apple";
</script>

</body>
</html>
```

JavaScript Numbers

A number divided by a non-numeric string becomes NaN (Not a Number):

NaN

Number Properties Cannot be Used on Variables

Number properties belongs to the JavaScript's number object wrapper called Number.

These properties can only be accessed as Number.MAX_VALUE.

Using **myNumber.MAX_VALUE**, where **myNumber** is a variable, expression, or value, will return undefined:

Example

Java Script Programming Language

273

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Object Properties</h2>

<p>Using a Number property on a variable, expression, or value, will return undefined:</p>

<p id="demo"></p>

<script>
var x = 6;
document.getElementById("demo").innerHTML = x.MAX_VALUE;
</script>

</body>
</html>
```

JavaScript Number Object Properties

Using a Number property on a variable, expression, or value, will return undefined:

undefined

• JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Java Script Programming Language

274

JavaScript Arrays

Saab,Volvo,BMW

What is an Array?

An **array** is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

Creating an Array

Using an **array** literal is the easiest way to create a JavaScript Array.

Syntax:

```
var array_name = [item1, item2, ...];
```

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Java Script Programming Language

275

JavaScript Arrays

Saab,Volvo,BMW

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

Java Script Programming Language

276

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW

Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

Java Script Programming Language

277

JavaScript Arrays

JavaScript array elements are accessed using numeric indexes (starting from 0).

Saab

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

This statement changes the value of the first element in **cars**:

```
cars[0] = "Opel";
```

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>
<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

JavaScript array elements are accessed using numeric indexes (starting from 0).

Opel,Volvo,BMW

Access the Full Array

With **JavaScript**, the full array can be accessed by referring to the array name:

Java Script Programming Language

278

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW

Arrays are Objects

Arrays are a special type of objects. The **typeof** operator in JavaScript returns "**object**" for arrays. But JavaScript arrays are best described as arrays.

Arrays use numbers to access its "elements". In this example, **person[0]** returns John:

Array:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Arrays use numbers to access its elements.</p>

<p id="demo"></p>

<script>
var person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML = person[0];
</script>

</body>
</html>
```

Java Script Programming Language

279

JavaScript Arrays

Arrays use numbers to access its elements.

John

Objects use names to access its "members". In this example, **person.firstName** returns John:

Object:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<p>JavaScript uses names to access object properties.</p>
<p id="demo"></p>

<script>
var person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML = person["firstName"];
</script>

</body>
</html>
```

JavaScript Objects

JavaScript uses names to access object properties.

John

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Java Script Programming Language

280

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Example

```
var x = cars.length; // The length property returns the number of elements  
var y = cars.sort(); // The sort() method sorts arrays
```

The length Property

The **length** property of an array returns the length of an array (the number of array elements).

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript Arrays</h2>  
<p>The length property returns the length of an array.</p>  
  
<p id="demo"></p>  
  
<script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.length;  
</script>  
  
</body>  
</html>
```

JavaScript Arrays

The length property returns the length of an array.

4

The length property is always one more than the highest array index.

Accessing the First Array Element

Example

Java Script Programming Language

281

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accesses using numeric indexes (starting from 0).</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
document.getElementById("demo").innerHTML = first;
</script>

</body>
</html>
```

JavaScript Arrays

JavaScript array elements are accesses using numeric indexes (starting from 0).

Banana

Accessing the Last Array Element

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accesses using numeric indexes (starting from 0).</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length-1];
document.getElementById("demo").innerHTML = last;
</script>

</body>
</html>
```

Java Script Programming Language

282

JavaScript Arrays

JavaScript array elements are accessed using numeric indexes (starting from 0).

Mango

Looping Array Elements

The safest way to loop through an array, is using a for loop:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>

<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript Arrays

The best way to loop through an array is using a standard for loop:

- Banana
- Orange
- Apple
- Mango

Java Script Programming Language

283

You can also use the `Array.forEach()` function:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Array.forEach() calls a function for each array element.</p>
<p>Array.forEach() is not supported in Internet Explorer 8 or earlier.</p>

<p id="demo"></p>

<script>
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";
document.getElementById("demo").innerHTML = text;

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
</script>

</body>
</html>
```

JavaScript Arrays

`Array.forEach()` calls a function for each array element.

`Array.forEach()` is not supported in Internet Explorer 8 or earlier.

- Banana
- Orange
- Apple
- Mango

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example

Java Script Programming Language

284

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The push method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Arrays

The push method appends a new element to an array.

Try it

Banana,Orange,Apple,Mango

JavaScript Arrays

The push method appends a new element to an array.

Try it

Banana,Orange,Apple,Mango,Lemon

New element can also be added to an array using the length property:

Example

Java Script Programming Language

285

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The length property provides an easy way to append new elements to an array without using the push() method.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
    fruits[fruits.length] = "Lemon";
    document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Arrays

The length property provides an easy way to append new elements to an array without using the push() method.

Try it

Banana,Orange,Apple,Mango

JavaScript Arrays

The length property provides an easy way to append new elements to an array without using the push() method.

Try it

Banana,Orange,Apple,Mango,Lemon

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

Example

Java Script Programming Language

286

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Adding elements with high indexes can create undefined "holes" in an array.</p>

<p id="demo"></p>

<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";

fLen = fruits.length;
text = "";
for (i = 0; i < fLen; i++) {
    text += fruits[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript Arrays

Adding elements with high indexes can create undefined "holes" in an array.

Banana
Orange
Apple
Mango
undefined
undefined
Lemon

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does not support arrays with named indexes.

In JavaScript, arrays always use numbered indexes.

Example

Java Script Programming Language

287

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>

</body>
</html>
```

JavaScript Arrays

John 3

WARNING !!

If you use named indexes, JavaScript will redefine the array to a standard object.

After that, some array methods and properties will produce **incorrect results**.

Example

Java Script Programming Language

288

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>If you use a named index when accessing an array, JavaScript will redefine the array to a standard object, and some array methods and properties will produce undefined or incorrect results.</p>

<p id="demo"></p>

<script>
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>

</body>
</html>
```

JavaScript Arrays

If you use a named index when accessing an array, JavaScript will redefine the array to a standard object, and some array methods and properties will produce undefined or incorrect results.

undefined 0

The Difference Between Arrays and Objects

In JavaScript, arrays use numbered **indexes**.

In JavaScript, objects use named **indexes**.

When to Use Arrays. When to use Objects.

JavaScript does not support associative arrays.

You should use objects when you want the element names to be **strings** (text).

You should use arrays when you want the element names to be **numbers**.

Avoid **new Array()**

There is no need to use the JavaScript's built-in array constructor **new Array()**.

Java Script Programming Language

289

Use [] instead.

These two different statements both create a new empty array named points:

```
var points = new Array();      // Bad
var points = [];                // Good
```

These two different statements both create a new array containing 6 numbers:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Avoid using new Array(). Use [] instead.</p>

<p id="demo"></p>

<script>
//var points = new Array(40, 100, 1, 5, 25, 10);
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points[0];
</script>

</body>
</html>
```

JavaScript Arrays

Avoid using new Array(). Use [] instead.

40

The **new** keyword only complicates the code. It can also produce some unexpected results:

```
var points = new Array(40, 100); // Creates an array with two elements (40 and 100)
```

What if I remove one of the elements?

Java Script Programming Language

290

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Avoid using new Array().</p>

<p id="demo"></p>

<script>
var points = new Array(40);
document.getElementById("demo").innerHTML = points[0];
</script>

</body>
</html>
```

JavaScript Arrays

Avoid using new Array().

undefined

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator **typeof** returns "**object**":

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The typeof operator, when used on an array, returns object:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = typeof fruits;
</script>

</body>
</html>
```

Java Script Programming Language

291

JavaScript Arrays

The `typeof` operator, when used on an array, returns `object`:

`object`

The `typeof` operator returns `object` because a JavaScript array is an object.

Solution 1:

To solve this problem ECMAScript 5 defines a new method `Array.isArray()`:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The new ECMASCIPT 5 method isArray returns true when used on an array:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = Array.isArray(fruits);
</script>

</body>
</html>
```

JavaScript Arrays

The new ECMASCIPT 5 method isArray returns true when used on an array:

`true`

The problem with this solution is that `ECMAScript 5` is not supported in older browsers.

Solution 2:

To solve this problem you can create your own `isArray()` function:

Java Script Programming Language

292

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>This "home made" isArray() function returns true when used on an array:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = isArray(fruits);

function isArray(myArray) {
  return myArray.constructor.toString().indexOf("Array") > -1;
}
</script>

</body>
</html>
```

JavaScript Arrays

This "home made" isArray() function returns true when used on an array:

true

The function above always returns true if the argument is an array.

Or more precisely: it returns true if the object prototype contains the word "**Array**".

Solution 3:

The **instanceof** operator returns true if an object is created by a given constructor:

Java Script Programming Language

293

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The instanceof operator returns true when used on an array:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits instanceof Array;
</script>

</body>
</html>
```

JavaScript Arrays

The instanceof operator returns true when used on an array:

true

• JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method **toString()** converts an array to a string of (comma separated) array values.

Example

Java Script Programming Language

294

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>toString()</h2>

<p>The toString() method returns an array as a comma separated string:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>

</body>
</html>
```

JavaScript Array Methods

toString()

The **toString()** method returns an array as a comma separated string:

Banana,Orange,Apple,Mango

The **join()** method also joins all array elements into a string.

It behaves just like **toString()**, but in addition you can specify the separator:

Example

Java Script Programming Language

295

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>join()</h2>

<p>The join() method joins array elements into a string.</p>
<p>In this example we have used " * " as a separator between the elements:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>

</body>
</html>
```

JavaScript Array Methods

join()

The join() method joins array elements into a string.

In this example we have used " * " as a separator between the elements:

Banana * Orange * Apple * Mango

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

Popping

The **pop()** method removes the last element from an array:

Example

Java Script Programming

Language

296

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>pop()</h2>

<p>The pop() method removes the last element from an array.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

pop()

The **pop()** method removes the last element from an array.

Banana,Orange,Apple,Mango

Banana,Orange,Apple

The **pop()** method returns the value that was "popped out":

Example

Java Script Programming Language

297

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>pop()</h2>

<p>The pop() method removes the last element from an array.</p>
<p>The return value of the pop() method is the removed item.</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.pop();
document.getElementById("demo3").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

pop()

The **pop()** method removes the last element from an array.

The return value of the **pop()** method is the removed item.

Banana,Orange,Apple,Mango

Mango

Banana,Orange,Apple

Pushing

The **push()** method adds a new element to an array (at the end):

Example

Java Script Programming Language

298

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>push()</h2>

<p>The push() method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Kiwi");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

push()

The push() method appends a new element to an array.

[Try it](#)

Banana,Orange,Apple,Mango

JavaScript Array Methods

push()

The push() method appends a new element to an array.

[Try it](#)

Banana,Orange,Apple,Mango,Kiwi

Java Script Programming

Language

299

The **push()** method returns the new array length:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>push()</h2>

<p>The push() method returns the new array length.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

function myFunction() {
  document.getElementById("demo2").innerHTML = fruits.push("Kiwi");
  document.getElementById("demo1").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

push()

The push() method returns the new array length.

Try it

Banana,Orange,Apple,Mango

Java Script Programming Language

300

JavaScript Array Methods

push()

The push() method returns the new array length.

Try it

Banana,Orange,Apple,Mango,Kiwi

5

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The **shift()** method removes the first array element and "shifts" all other elements to a lower index.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>shift()</h2>

<p>The shift() method removes the first element of an array (and "shifts" all other elements to the left):</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.shift();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

Java Script Programming

Language

301

JavaScript Array Methods

shift()

The **shift()** method removes the first element of an array (and "shifts" all other elements to the left):

Banana,Orange,Apple,Mango

Orange,Apple,Mango

The **shift()** method returns the string that was "**shifted out**":

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>shift()</h2>

<p>The shift() method returns the element that was shifted out.</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.shift();
document.getElementById("demo3").innerHTML = fruits;
</script>

</body>
</html>
```

Java Script Programming Language

302

JavaScript Array Methods

shift()

The **shift()** method returns the element that was shifted out.

Banana,Orange,Apple,Mango

Banana

Orange,Apple,Mango

The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>unshift()</h2>

<p>The unshift() method adds new elements to the beginning of an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.unshift("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Java Script Programming Language

303

JavaScript Array Methods

unshift()

The unshift() method adds new elements to the beginning of an array.

[Try it](#)

Banana,Orange,Apple,Mango

JavaScript Array Methods

unshift()

The unshift() method adds new elements to the beginning of an array.

[Try it](#)

Lemon,Banana,Orange,Apple,Mango

The **unshift()** method returns the new array length.

Example

Java Script Programming Language

304

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>unshift()</h2>

<p>The unshift() method returns the length of the new array:</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.unshift("Lemon");
document.getElementById("demo3").innerHTML = fruits;
</script>

<p><b>Note:</b> The unshift() method does not work properly in Internet Explorer 8 and earlier, the values will be inserted, but the return value will be <em>undefined</em>.</p>

</body>
</html>
```

JavaScript Array Methods

unshift()

The unshift() method returns the length of the new array:

Banana,Orange,Apple,Mango

5

Lemon,Banana,Orange,Apple,Mango

Note: The unshift() method does not work properly in Internet Explorer 8 and earlier, the values will be inserted, but the return value will be *undefined*.

Changing Elements

Array elements are accessed using their index number:

Java Script Programming

Language

305

Array **indexes** start with 0. [0] is the first array element, [1] is the second, [2] is the third

...

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>Array elements are accessed using their index number:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits[0] = "Kiwi";
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

Array elements are accessed using their index number:

Banana,Orange,Apple,Mango

Kiwi,Orange,Apple,Mango

The **length** property provides an easy way to append a new element to an array:

Example

Java Script Programming Language

306

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>The length property provides an easy way to append new elements to an array without using the push() method.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits[fruits.length] = "Kiwi";
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

The length property provides an easy way to append new elements to an array without using the push() method.

Try it

Banana,Orange,Apple,Mango

JavaScript Array Methods

The length property provides an easy way to append new elements to an array without using the push() method.

Try it

Banana,Orange,Apple,Mango,Kiwi

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

Example

Java Script Programming Language

307

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>Deleting elements leaves undefined holes in an array.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML =
"The first fruit is: " + fruits[0];
delete fruits[0];
document.getElementById("demo2").innerHTML =
"The first fruit is: " + fruits[0];
</script>

</body>
</html>
```

JavaScript Array Methods

Deleting elements leaves undefined holes in an array.

The first fruit is: Banana

The first fruit is: undefined

Splicing an Array

The **splice()** method can be used to add new items to an array:

Example

Java Script Programming Language

308

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() method adds new elements to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original Array:<br>" + fruits;
function myFunction() {
  fruits.splice(2, 0, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() method adds new elements to an array.

Try it

Original Array:
Banana,Orange,Apple,Mango

Java Script Programming Language

309

JavaScript Array Methods

splice()

The `splice()` method adds new elements to an array.

[Try it](#)

Original Array:

Banana,Orange,Apple,Mango

New Array:

Banana,Orange,Lemon,Kiwi,Apple,Mango

The **first parameter (2)** defines the position where new elements should be added (spliced in).

The **second parameter (0)** defines how many elements should be removed.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

The `splice()` method returns an array with the deleted items:

Example

Java Script Programming Language

310

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() method adds new elements to an array, and returns an array with the deleted elements (if any).</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original Array:<br> " + fruits;

function myFunction() {
  var removed = fruits.splice(2, 2, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;
  document.getElementById("demo3").innerHTML = "Removed Items:<br> " + removed;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() method adds new elements to an array, and returns an array with the deleted elements (if any).

[Try it](#)

Original Array:
Banana,Orange,Apple,Mango

Java Script Programming Language

311

JavaScript Array Methods

splice()

The `splice()` method adds new elements to an array, and returns an array with the deleted elements (if any).

Try it

Original Array:

Banana,Orange,Apple,Mango

New Array:

Banana,Orange,Lemon,Kiwi

Removed Items:

Apple,Mango

Using `splice()` to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

Java Script Programming Language

312

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() methods can be used to remove array elements.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits.splice(0, 1);
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() methods can be used to remove array elements.

Try it

Banana,Orange,Apple,Mango

JavaScript Array Methods

splice()

The splice() methods can be used to remove array elements.

Try it

Orange,Apple,Mango

Java Script Programming Language

313

The first parameter (0) defines the position where new elements should be added (spliced in).

The second parameter (1) defines how many elements should be removed.

The rest of the parameters are omitted. No new elements will be added.

Merging (Concatenating) Arrays

The **concat()** method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>concat()</h2>
<p>The concat() method is used to merge (concatenate) arrays:</p>
<p id="demo"></p>

<script>
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method is used to merge (concatenate) arrays:

Cecilie,Lone,Emil,Tobias, Linus

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments:

Java Script Programming Language

314

Example (Merging Three Arrays)

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method is used to merge (concatenate) arrays:</p>

<p id="demo"></p>

<script>
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];

var myChildren = arr1.concat(arr2, arr3);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method is used to merge (concatenate) arrays:

Cecilie,Lone,Emil,Tobias,Robin,Morgan

The **concat()** method can also take strings as arguments:

Example (Merging an Array with Values)

Java Script Programming Language

315

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method can also merge string values to arrays:</p>

<p id="demo"></p>

<script>
var arr1 = ["Emil", "Tobias", "Linus"];
var myChildren = arr1.concat("Peter");
document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method can also merge string values to arrays:

Emil,Tobias,Linus,Peter

Slicing an Array

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

Java Script Programming Language

316

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>slice()</h2>

<p>This example slices out a part of an array starting from array element 1 ("Orange"):</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 1 ("Orange"):

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon,Apple,Mango

The **slice()** method creates a new array. It does not remove any elements from the source array.

This example slices out a part of an array starting from array element 3 ("Apple"):

Example

Java Script Programming

Language

317

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>slice()</h2>

<p>This example slices out a part of an array starting from array element 3 ("Apple")</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(3);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 3 ("Apple")

Banana,Orange,Lemon,Apple,Mango

Apple,Mango

The **slice()** method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

Java Script Programming Language

318

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>slice()</h2>

<p>When the slice() method is given two arguments, it selects array elements from the start argument, and up to (but not included) the end argument:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1,3);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

When the slice() method is given two arguments, it selects array elements from the start argument, and up to (but not included) the end argument:

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon

If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.

Example

Java Script Programming Language

319

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>slice()</h2>

<p>This example slices out a part of an array starting from array element 2 ("Lemon"):</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(2);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 2 ("Lemon"):

Banana,Orange,Lemon,Apple,Mango

Lemon,Apple,Mango

Automatic `toString()`

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

Example

Java Script Programming Language

320

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>toString()</h2>

<p>The toString() method returns an array as a comma separated string:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>

</body>
</html>
```

JavaScript Array Methods

toString()

The `toString()` method returns an array as a comma separated string:

Banana,Orange,Apple,Mango

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>JavaScript automatically converts an array to a comma separated string when a simple value is expected:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
</script>

</body>
</html>
```

Java Script Programming Language

321

JavaScript Array Methods

JavaScript automatically converts an array to a comma separated string when a simple value is expected:

Banana,Orange,Apple,Mango

• JavaScript Date Objects

Creating Date Objects

Date objects are created with the **new Date()** constructor.

There are 4 ways to create a new date object:

```
new Date()  
new Date(year, month, day, hours, minutes, seconds, milliseconds)  
new Date(milliseconds)  
new Date(date string)
```

new Date()

new Date() creates a new date object with the current date and time:

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript new Date()</h2>  
  
<p>Using new Date(), creates a new date object with the current date and time:</p>  
  
<p id="demo"></p>  
  
<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d;  
</script>  
  
</body>  
</html>
```

Java Script Programming Language

322

JavaScript new Date()

Using `new Date()`, creates a new date object with the current date and time:

Sun May 17 2020 03:43:10 GMT+0300 (Arabian Standard Time)

`new Date(year, month, ...)`

new Date(year, month, ...) creates a new date object with a specified date and time.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript new Date()</h2>

<p>Using new Date(7 numbers), creates a new date object with the specified date and time:</p>

<p id="demo"></p>

<script>
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

JavaScript new Date()

Using `new Date(7 numbers)`, creates a new date object with the specified date and time:

Mon Dec 24 2018 10:33:30 GMT+0300 (Arabian Standard Time)

6 numbers specify year, month, day, hour, minute, second:

Example

Java Script Programming Language

323

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript new Date()</h2>

<p>6 numbers specify year, month, day, hour, minute and second:</p>

<p id="demo"></p>

<script>
var d = new Date(2018, 11, 24, 10, 33, 30);
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

JavaScript new Date()

6 numbers specify year, month, day, hour, minute and second:

Mon Dec 24 2018 10:33:30 GMT+0300 (Arabian Standard Time)

Previous Century

One and two digit years will be interpreted as 19xx:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript new Date()</h2>

<p>Two digit years will be interpreted as 19xx:</p>

<p id="demo"></p>

<script>
var d = new Date(99, 11, 24);
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

Java Script Programming Language

324

JavaScript new Date()

Two digit years will be interpreted as 19xx:

Fri Dec 24 1999 00:00:00 GMT+0300 (Arabian Standard Time)

new Date(dateString)

new Date(dateString) creates a new date object from a date string:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript new Date()</h2>

<p>A Date object can be created with a specified date and time:</p>

<p id="demo"></p>

<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

JavaScript new Date()

A Date object can be created with a specified date and time:

Mon Oct 13 2014 11:13:00 GMT+0300 (Arabian Standard Time)

new Date(milliseconds)

new Date(milliseconds) creates a new date object as zero time plus milliseconds:

Example

Java Script Programming Language

325

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript new Date()</h2>

<p>Using new Date(milliseconds), creates a new date object as January 1, 1970, 00:00:00 Universal Time (UTC) plus the milliseconds:</p>

<p id="demo"></p>

<script>
var d = new Date(0);
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

JavaScript new Date()

Using new Date(milliseconds), creates a new date object as January 1, 1970, 00:00:00 Universal Time (UTC) plus the milliseconds:

Thu Jan 01 1970 03:00:00 GMT+0300 (Arabian Standard Time)

Displaying Dates

JavaScript will (by default) output dates in full text string format:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript new Date()</h2>

<p id="demo"></p>

<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

Java Script Programming Language

326

JavaScript new Date()

Sun May 17 2020 03:51:33 GMT+0300 (Arabian Standard Time)

The **toUTCString()** method converts a date to a UTC string (a date display standard).

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Date()</h2>

<p>The toUTCString() method converts a date to a UTC string (a date display standard):</p>

<p id="demo"></p>

<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
</script>

</body>
</html>
```

JavaScript Date()

The **toUTCString()** method converts a date to a UTC string (a date display standard):

Sun, 17 May 2020 00:52:28 GMT

The **toDateString()** method converts a date to a more readable format:

Example

Java Script Programming Language

327

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript toDateString()</h2>
<p>The toDateString() method converts a date to a date string:</p>
<p id="demo"></p>

<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toDateString();
</script>

</body>
</html>
```

JavaScript toDateString()

The toDateString() method converts a date to a date string:

Sun May 17 2020

• JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.PI</h2>
<p>Math.PI returns the ratio of a circle's circumference to its diameter:</p>
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.PI;
</script>

</body>
</html>
```

Java Script Programming Language

328

JavaScript Math.PI

Math.PI returns the ratio of a circle's circumference to its diameter:

3.141592653589793

Math.round()

Math.round(x) returns the value of x rounded to its nearest integer:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.round()</h2>

<p>Math.round(x) returns the value of x rounded to its nearest integer:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.round(4.4);
</script>

</body>
</html>
```

JavaScript Math.round()

Math.round(x) returns the value of x rounded to its nearest integer:

4

Math.pow()

Math.pow(x, y) returns the value of x to the power of y:

Example

Java Script Programming Language

329

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.pow()</h2>

<p>Math.pow(x,y) returns the value of x to the power of y:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.pow(8,2);
</script>

</body>
</html>
```

JavaScript Math.pow()

Math.pow(x,y) returns the value of x to the power of y:

64

Math.sqrt()

Math.sqrt(x) returns the square root of x:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.sqrt()</h2>

<p>Math.sqrt(x) returns the square root of x:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.sqrt(64);
</script>

</body>
</html>
```

Java Script Programming Language

330

JavaScript Math.sqrt()

Math.sqrt(x) returns the square root of x:

8

Math.abs()

Math.abs(x) returns the absolute (positive) value of x:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.abs()</h2>
<p>Math.abs(x) returns the absolute (positive) value of x:</p>
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.abs(-4.4);
</script>

</body>
</html>
```

JavaScript Math.abs()

Math.abs(x) returns the absolute (positive) value of x:

4.4

Math.ceil()

Math.ceil(x) returns the value of x rounded up to its nearest integer:

Example

Java Script Programming Language

331

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.ceil()</h2>

<p>Math.ceil() rounds a number <strong>up</strong> to its nearest integer:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.ceil(4.4);
</script>

</body>
</html>
```

JavaScript Math.ceil()

Math.ceil() rounds a number up to its nearest integer:

5

Math.floor()

Math.floor(x) returns the value of x rounded down to its nearest integer:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.floor()</h2>

<p>Math.floor(x) returns the value of x rounded <strong>down</strong> to its nearest
integer:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.floor(4.7);
</script>

</body>
</html>
```

Java Script Programming Language

332

JavaScript Math.floor()

Math.floor(x) returns the value of x rounded down to its nearest integer:

4

Math.sin()

Math.sin(x) returns the sine (a value between -1 and 1) of the angle x (given in radians).

If you want to use degrees instead of radians, you have to convert degrees to radians:

Angle in radians = Angle in degrees x PI / 180.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.sin()</h2>

<p>Math.sin(x) returns the sin of x (given in radians):</p>
<p>Angle in radians = (angle in degrees) * PI / 180.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"The sine value of 90 degrees is " + Math.sin(90 * Math.PI / 180);
</script>

</body>
</html>
```

JavaScript Math.sin()

Math.sin(x) returns the sin of x (given in radians):

Angle in radians = (angle in degrees) * PI / 180.

The sine value of 90 degrees is 1

Math.cos()

Java Script Programming Language

333

Math.cos(x) returns the cosine (a value between -1 and 1) of the angle x (given in radians).

If you want to use degrees instead of radians, you have to convert degrees to radians:

Angle in radians = Angle in degrees x PI / 180.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.cos()</h2>

<p>Math.cos(x) returns the cosine of x (given in radians):</p>
<p>Angle in radians = (angle in degrees) * PI / 180.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"The cosine value of 0 degrees is " + Math.cos(0 * Math.PI / 180);
</script>

</body>
</html>
```

JavaScript Math.cos()

Math.cos(x) returns the cosine of x (given in radians):

Angle in radians = (angle in degrees) * PI / 180.

The cosine value of 0 degrees is 1

Math.min() and Math.max()

Math.min() and **Math.max()** can be used to find the lowest or highest value in a list of arguments:

Example

Java Script Programming Language

334

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.min()</h2>

<p>Math.min() returns the lowest value in a list of arguments:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
Math.min(0, 150, 30, 20, -8, -200);
</script>

</body>
</html>
```

JavaScript Math.min()

Math.min() returns the lowest value in a list of arguments:

-200

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.max()</h2>

<p>Math.max() returns the highest value in a list of arguments.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
Math.max(0, 150, 30, 20, -8, -200);
</script>

</body>
</html>
```

JavaScript Math.max()

Math.max() returns the highest value in a list of arguments.

150

Java Script Programming Language

335

Math.random()

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.random()</h2>

<p>Math.random() returns a random number between 0 and 1:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.random();
</script>

</body>
</html>
```

JavaScript Math.random()

Math.random() returns a random number between 0 and 1:

0.14186863112187642

Math Properties (Constants)

JavaScript provides 8 mathematical constants that can be accessed with the Math object:

Example

Java Script Programming Language

336

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math Constants</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"<p><b>Math.E:</b> " + Math.E + "</p>" +
"<p><b>Math.PI:</b> " + Math.PI + "</p>" +
"<p><b>Math.SQRT2:</b> " + Math.SQRT2 + "</p>" +
"<p><b>Math.SQRT1_2:</b> " + Math.SQRT1_2 + "</p>" +
"<p><b>Math.LN2:</b> " + Math.LN2 + "</p>" +
"<p><b>Math.LN10:</b> " + Math.LN10 + "</p>" +
"<p><b>Math.LOG2E:</b> " + Math.LOG2E + "</p>" +
"<p><b>Math.Log10E:</b> " + Math.LOG10E + "</p>";
</script>

</body>
</html>
```

JavaScript Math Constants

Math.E: 2.718281828459045

Math.PI: 3.141592653589793

Math.SQRT2: 1.4142135623730951

Math.SQRT1_2: 0.7071067811865476

Math.LN2: 0.6931471805599453

Math.LN10: 2.302585092994046

Math.LOG2E: 1.4426950408889634

Math.Log10E: 0.4342944819032518

Math Constructor

Unlike other global objects, the Math object has no constructor. Methods and properties are static.

All methods and properties (constants) can be used without creating a Math object first.

Math Object Methods

Java Script Programming

Language

337

Method	Description
<code>abs(x)</code>	Returns the absolute value of x
<code>acos(x)</code>	Returns the arccosine of x, in radians
<code>acosh(x)</code>	Returns the hyperbolic arccosine of x
<code>asin(x)</code>	Returns the arcsine of x, in radians
<code>asinh(x)</code>	Returns the hyperbolic arcsine of x
<code>atan(x)</code>	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
<code>atan2(y, x)</code>	Returns the arctangent of the quotient of its arguments
<code>atanh(x)</code>	Returns the hyperbolic arctangent of x
<code>cbrt(x)</code>	Returns the cubic root of x
<code>ceil(x)</code>	Returns x, rounded upwards to the nearest integer
<code>cos(x)</code>	Returns the cosine of x (x is in radians)
<code>cosh(x)</code>	Returns the hyperbolic cosine of x
<code>exp(x)</code>	Returns the value of E ^x
<code>floor(x)</code>	Returns x, rounded downwards to the nearest integer
<code>log(x)</code>	Returns the natural logarithm (base E) of x
<code>max(x, y, z, ..., n)</code>	Returns the number with the highest value
<code>min(x, y, z, ..., n)</code>	Returns the number with the lowest value
<code>pow(x, y)</code>	Returns the value of x to the power of y
<code>random()</code>	Returns a random number between 0 and 1

Java Script Programming Language

338

round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of x
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle
tanh(x)	Returns the hyperbolic tangent of a number
trunc(x)	Returns the integer part of a number (x)

• JavaScript Random

Math.random()

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.random()</h2>

<p>Math.random() returns a random number between 0 (included) and 1 (excluded):</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Math.random();
</script>

</body>
</html>
```

JavaScript Math.random()

Math.random() returns a random number between 0 (included) and 1 (excluded):

0.025300771300114988

Java Script Programming Language

339

Math.random() always returns a number lower than 1.

JavaScript Random Integers

Math.random() used with **Math.floor()** can be used to return random integers.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math</h2>

<p>Math.floor(Math.random() * 10) returns a random integer between 0 and 9 (both included):</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
Math.floor(Math.random() * 10);
</script>

</body>
</html>
```

JavaScript Math

`Math.floor(Math.random() * 10)` returns a random integer between 0 and 9 (both included):

8

A Proper Random Function

This JavaScript function always returns a random number between min (included) and max (excluded):

Example

Java Script Programming Language

340

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.random()</h2>

<p>Every time you click the button, getRndInteger(min, max) returns a random number between 0 and 9 (both included):</p>

<button onclick="document.getElementById('demo').innerHTML = getRndInteger(0,10)">Click Me</button>

<p id="demo"></p>

<script>
function getRndInteger(min, max) {
    return Math.floor(Math.random() * (max - min)) + min;
}
</script>

</body>
</html>
```

JavaScript Math.random()

Every time you click the button, getRndInteger(min, max) returns a random number between 0 and 9 (both included):

Click Me

This **JavaScript** function always returns a random number between min and max (both included):

Example

Java Script Programming Language

341

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.random()</h2>

<p>Every time you click the button, getRndInteger(min, max) returns a random number between 1 and 10 (both included):</p>

<button onclick="document.getElementById('demo').innerHTML = getRndInteger(1,10)">Click Me</button>

<p id="demo"></p>

<script>
function getRndInteger(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
</script>

</body>
</html>
```

JavaScript Math.random()

Every time you click the button, getRndInteger(min, max) returns a random number between 1 and 10 (both included):

Click Me

JavaScript Math.random()

Every time you click the button, getRndInteger(min, max) returns a random number between 1 and 10 (both included):

Click Me

5

• JavaScript Booleans

A **JavaScript Boolean** represents one of two values: **true** or **false**.

Boolean Values

Java Script Programming

Language

342

Very often, in programming, you will need a data type that can only have one of two values, like

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, JavaScript has a Boolean data type. It can only take the values true or false.

The Boolean() Function

You can use the **Boolean()** function to find out if an expression (or a variable) is true:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Display the value of Boolean(10 > 9):</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = Boolean(10 > 9);
}
</script>

</body>
</html>
```

Display the value of Boolean(10 > 9):

[Try it](#)

Display the value of Boolean(10 > 9):

[Try it](#)

true

Or even easier:

Java Script Programming Language

343

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Display the value of 10 > 9:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = 10 > 9;
}
</script>

</body>
</html>
```

Display the value of 10 > 9:

[Try it](#)

Display the value of 10 > 9:

[Try it](#)

true

Everything With a "Value" is True

Example

Java Script Programming

Language

344

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var b1 = Boolean(100);
var b2 = Boolean(3.14);
var b3 = Boolean(-15);
var b4 = Boolean("Hello");
var b5 = Boolean('false');
var b6 = Boolean(1 + 7 + 3.14);

document.getElementById("demo").innerHTML =
"100 is " + b1 + "<br>" +
"3.14 is " + b2 + "<br>" +
"-15 is " + b3 + "<br>" +
"Any (not empty) string is " + b4 + "<br>" +
"Even the string 'false' is " + b5 + "<br>" +
"Any expression (except zero) is " + b6;
</script>

</body>
</html>
```

100 is true
3.14 is true
-15 is true
Any (not empty) string is true
Even the string 'false' is true
Any expression (except zero) is true

Everything Without a "Value" is False

The Boolean value of 0 (zero) is false:

Java Script Programming

Language

345

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of 0:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = 0;
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Display the Boolean value of 0:

[Try it](#)

Display the Boolean value of 0:

[Try it](#)

false

The Boolean value of -0 (minus zero) is false:

Java Script Programming

Language

346

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of -0:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = -0;
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Display the Boolean value of -0:

[Try it](#)

Display the Boolean value of -0:

[Try it](#)

false

The Boolean value of "" (empty string) is false:

Java Script Programming

Language

347

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of "":</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = "";
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Display the Boolean value of "":

Try it

Display the Boolean value of "":

Try it

false

The Boolean value of undefined is false:

Java Script Programming

Language

348

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of undefined:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x;
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Display the Boolean value of undefined:

[Try it](#)

Display the Boolean value of undefined:

[Try it](#)

false

The Boolean value of null is false:

Java Script Programming

Language

349

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of null:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = null;
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Display the Boolean value of null:

[Try it](#)

Display the Boolean value of null:

[Try it](#)

false

The Boolean value of false is (you guessed it) false:

Java Script Programming Language

350

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of false:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = false;
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Display the Boolean value of false:

Try it

Display the Boolean value of false:

Try it

false

The Boolean value of NaN is false:

```
<!DOCTYPE html>
<html>
<body>

<p>Display the Boolean value of NaN:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var x = 10 / "H";
  document.getElementById("demo").innerHTML = Boolean(x);
}
</script>

</body>
</html>
```

Java Script Programming Language

351

Display the Boolean value of NaN:

[Try it](#)

Display the Boolean value of NaN:

[Try it](#)

false

Booleans Can be Objects

Normally **JavaScript Booleans** are primitive values created from literals:

```
var x = false;
```

But **Booleans** can also be defined as objects with the keyword new:

```
var y = new Boolean(false);
```

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Never create booleans as objects.</p>
<p>Booleans and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = false;          // x is a boolean
var y = new Boolean(false); // y is an object
document.getElementById("demo").innerHTML = typeof x + "<br>" + typeof y;
</script>

</body>
</html>
```

Never create booleans as objects.

Booleans and objects cannot be safely compared.

```
boolean
object
```

Java Script Programming Language

352

Do not create Boolean objects. It slows down execution speed. The `new` keyword complicates the code. This can produce some unexpected results:

When using the `==` operator, equal Booleans are equal:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Never create booleans as objects.</p>
<p>Booleans and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = false;          // x is a boolean
var y = new Boolean(false); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

Never create booleans as objects.

Booleans and objects cannot be safely compared.

true

When using the `==` operator, equal Booleans are not equal, because the `==` operator expects equality in both type and value.

Example

Java Script Programming Language

353

```
<!DOCTYPE html>
<html>
<body>

<p>Never create booleans as objects.</p>
<p>Booleans and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = false;          // x is a number
var y = new Boolean(false); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

Never create booleans as objects.

Booleans and objects cannot be safely compared.

false

Or even worse. Objects cannot be compared:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Never create booleans as objects.</p>
<p>Booleans and objects cannot be safely compared.</p>

<p id="demo"></p>

<script>
var x = new Boolean(false); // x is an object
var y = new Boolean(false); // y is an object
document.getElementById("demo").innerHTML = (x==y);
</script>

</body>
</html>
```

Never create booleans as objects.

Booleans and objects cannot be safely compared.

false

Java Script Programming

Language

354

Note the difference between (**x==y**) and (**x====y**). Comparing two JavaScript objects will always return false.

• JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for **true** or **false**.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that $x = 5$, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
<code>==</code>	equal to	<code>x == 8</code>	false
		<code>x == 5</code>	true
		<code>x == "5"</code>	true
<code>====</code>	equal value and equal type	<code>x === 5</code>	true
		<code>x === "5"</code>	false
<code>!=</code>	not equal	<code>x != 8</code>	true
<code>!==</code>	not equal value or not equal type	<code>x !== 5</code>	false

Java Script Programming Language

355

x	<code>!== "5"</code>	true
x	<code>!== 8</code>	true
>	greater than	<code>x > 8</code>
<	less than	<code>x < 8</code>
<code>>=</code>	greater than or equal to	<code>x >= 8</code>
<code><=</code>	less than or equal to	<code>x <= 8</code>

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age < 18) text = "Too young";
```

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that `x = 6` and `y = 3`, the table below explains the logical operators:

Operator	Description	Example
<code>&&</code>	and	<code>(x < 10 && y > 1)</code> is true
<code> </code>	or	<code>(x == 5 y == 5)</code> is false
!	not	<code>!(x == y)</code> is true

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Java Script Programming Language

356

Syntax

variablename = (condition) ? value1:value2

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Input your age and click the button:</p>

<input id="age" value="18" />

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var age, voteable;
  age = document.getElementById("age").value;
  voteable = (age < 18) ? "Too young": "Old enough";
  document.getElementById("demo").innerHTML = voteable + " to vote.";
}
</script>

</body>
</html>
```

Input your age and click the button:

Input your age and click the button:

Old enough to vote.

If the variable age is a value below 18, the value of the variable voteable will be "**Too young**", otherwise the value of voteable will be "**Old enough**".

Comparing Different Types

Comparing data of different types may give unexpected results.

Java Script Programming Language

357

When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison. An empty string converts to 0. A **non-numeric** string converts to **NaN** which is always **false**.

Case	Value
<code>2 < 12</code>	True
<code>2 < "12"</code>	True
<code>2 < "John"</code>	False
<code>2 > "John"</code>	False
<code>2 == "John"</code>	False
<code>"2" < "12"</code>	False
<code>"2" > "12"</code>	True
<code>"2" == "12"</code>	False

When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

To secure a proper result, variables should be converted to the proper type before comparison:

Java Script Programming Language

358

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Comparisons</h2>

<p>Input your age and click the button:</p>

<input id="age" value="18" />

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var age, voteable;
  age = Number(document.getElementById("age").value);
  if (isNaN(age)) {
    voteable = "Input is not a number";
  } else {
    voteable = (age < 18) ? "Too young" : "Old enough";
  }
  document.getElementById("demo").innerHTML = voteable;
}
</script>

</body>
</html>
```

JavaScript Comparisons

Input your age and click the button:

JavaScript Comparisons

Input your age and click the button:

Old enough

- **JavaScript if else and else if**

Conditional statements are used to perform different actions based on different conditions.

Java Script Programming

Language

359

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example

Make a "Good day" greeting if the hour is less than 18:00:

Java Script Programming

Language

360

```
<!DOCTYPE html>
<html>
<body>

<p>Display "Good day!" if the hour is less than 18:00:</p>
<p id="demo">Good Evening!</p>

<script>
if (new Date().getHours() < 18) {
  document.getElementById("demo").innerHTML = "Good day!";
}
</script>

</body>
</html>
```

Display "Good day!" if the hour is less than 18:00:

Good day!

The else Statement

Use the **else statement** to specify a block of code to be executed if the condition is **false**.

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example

Java Script Programming Language

361

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to display a time-based greeting:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var hour = new Date().getHours();
  var greeting;
  if (hour < 18) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>
```

Click the button to display a time-based greeting:

[Try it](#)

Click the button to display a time-based greeting:

[Try it](#)

Good day

The else if Statement

Use the **else if statement** to specify a new condition if the first condition is false.

Syntax

```
if (condition1)
  // block of code to be executed if condition1 is true
} else if (condition2)
  // block of code to be executed if the condition1 is false and condition2 is
true
} else {
```

Java Script Programming Language

362

```
// block of code to be executed if the condition1 is false and condition2 is
false
}
```

Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to get a time-based greeting:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var greeting;
  var time = new Date().getHours();
  if (time < 10) {
    greeting = "Good morning";
  } else if (time < 20) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>
```

Click the button to get a time-based greeting:

[Try it](#)

Click the button to get a time-based greeting:

[Try it](#)

Good morning

- **JavaScript Switch Statement**

Java Script Programming Language

363

The **switch** statement is used to perform different actions based on different conditions.

The JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Example

The **getDay()** method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

Java Script Programming Language

364

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
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";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>

</body>
</html>
```

Today is Monday

The break Keyword

When JavaScript reaches a **break** keyword, it breaks out of the switch block.

This will stop the execution of inside the block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

The default Keyword

Java Script Programming Language

365

The **default keyword** specifies the code to run if there is no case match:

Example

The **getDay()** method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
var text;
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript switch

Looking forward to the Weekend

The **default** case does not have to be the last case in a switch block:

Example

Java Script Programming Language

366

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
var text;
switch (new Date().getDay()) {
  default:
    text = "Looking forward to the Weekend";
    break;
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript switch

Looking forward to the Weekend

If **default** is not the last case in the switch block, remember to end the default case with a **break**.

Common Code Blocks

Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

Example

Java Script Programming Language

367

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
var text;
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript switch

Looking forward to the Weekend

Switching Details

If multiple cases match a case value, the first case is selected.

If no matching cases are found, the program continues to the default label.

If no default label is found, the program continues to the statement(s) after the switch.

Strict Comparison

Switch cases use strict comparison (`====`).

The values must be of the same type to match.

A **strict comparison** can only be true if the operands are of the same type.

In this example there will be no match for x:

Java Script Programming Language

368

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
var x = "0";

switch (x) {
  case 0:
    text = "Off";
    break;
  case 1:
    text = "On";
    break;
  default:
    text = "No value found";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript switch

No value found

• JavaScript For Loop

Loops can execute a block of code a number of times.

Different Kinds of Loops

JavaScript supports different kinds of loops:

- **for - loops** through a block of code a number of times
- **for/in - loops** through the properties of an object
- **for/of - loops** through the values of an iterable object
- **while - loops** through a block of code while a specified condition is true
- **do/while - loops** through a block of code while a specified condition is true

Java Script Programming

Language

369

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript For Loop</h2>  
  
<p id="demo"></p>  
  
<script>  
var text = "";  
var i;  
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>"  
}  
document.getElementById("demo").innerHTML = text;  
</script>  
  
</body>  
</html>
```

JavaScript For Loop

```
The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Java Script Programming Language

370

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

The For/In Loop

The **JavaScript for/in** statement loops through the properties of an object:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For/In Loop</h2>

<p>The for/in statement loops through the properties of an object.</p>

<p id="demo"></p>

<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

JavaScript For/In Loop

The for/in statement loops through the properties of an object.

John Doe 25

The For/Of Loop

The JavaScript for/of statement loops through the values of an iterable objects

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The **for/of** loop has the following syntax:

Java Script Programming Language

371

```
for (variable of iterable) {  
    // code block to be executed  
}
```

variable - For every iteration the value of the next property is assigned to the variable.

Variable can be declared with **const**, **let**, or **var**.

iterable - An object that has iterable properties.

Looping over an Array

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript For/Of Loop</h2>  
  
<p>The for/of statement loops through the values of an iterable object.</p>  
  
<script>  
var cars = ['BMW', 'Volvo', 'Mini'];  
var x;  
  
for (x of cars) {  
    document.write(x + "<br >");  
}  
</script>  
  
</body>  
</html>
```

JavaScript For/Of Loop

The for/of statement loops through the values of an iterable object.

BMW
Volvo
Mini

Looping over a String

Java Script Programming Language

372

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For/Of Loop</h2>

<p>The for/of statement loops through the values of an iterable object.</p>

<script>
var txt = 'JavaScript';
var x;

for (x of txt) {
  document.write(x + "<br >");
}
</script>

</body>
</html>
```

JavaScript For/Of Loop

The for/of statement loops through the values of an iterable object.

J
a
v
a
S
c
r
i
p
t

- ### JavaScript While Loop

Loops can execute a block of code as long as a specified condition is true.

The While Loop

The **while loop** loops through a block of code as long as a specified condition is true.

Syntax

Java Script Programming

Language

373

```
while (condition) {
  // code block to be executed
}
```

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (**i**) is less than 10:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript While Loop</h2>

<p id="demo"></p>

<script>
var text = "";
var i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript While Loop

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

The Do/While Loop

Java Script Programming Language

374

The **do/while loop** is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {  
    // code block to be executed  
}  
while (condition);
```

Example

The example below uses a **do/while** loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript Do/While Loop</h2>  
  
<p id="demo"></p>  
  
<script>  
var text = ""  
var i = 0;  
  
do {  
    text += "<br>The number is " + i;  
    i++;  
}  
while (i < 10);  
  
document.getElementById("demo").innerHTML = text;  
</script>  
  
</body>  
</html>
```

Java Script Programming

Language

375

JavaScript Do/While Loop

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a for loop to collect the car names from the cars array:

Example

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";
for (;cars[i]); {
    text += cars[i] + "<br>";
    i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Java Script Programming Language

376

BMW
Volvo
Saab
Ford

The loop in this example uses a **while** loop to collect the car names from the cars array:

Example

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";
while (cars[i]) {
    text += cars[i] + "<br>";
    i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

BMW
Volvo
Saab
Ford

• JavaScript Break and Continue

The **break** statement "jumps out" of a loop.

The **continue** statement "jumps over" one iteration in the loop.

The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a **switch()** statement.

The **break** statement can also be used to jump out of a loop.

Java Script Programming Language

377

The **break** statement breaks the loop and continues executing the code after the loop (if any):

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Loops</h2>

<p>A loop with a <b>break</b> statement.</p>

<p id="demo"></p>

<script>
var text = "";
var i;
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript Loops

A loop with a **break** statement.

```
The number is 0
The number is 1
The number is 2
```

The Continue Statement

The **continue** statement breaks one iteration (in the loop), if a specified condition occurs, and **continues** with the next iteration in the loop.

This example skips the value of 3:

Java Script Programming Language

378

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Loops</h2>

<p>A loop with a <b>continue</b> statement.</p>
<p>A loop which will skip the step where i = 3.</p>

<p id="demo"></p>

<script>
var text = "";
var i;
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript Loops

A loop with a **continue** statement.

A loop which will skip the step where $i = 3$.

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

JavaScript Labels

To **label JavaScript** statements you precede the statements with a label name and a colon:

```
label:
statements
```

Java Script Programming

Language

379

The **break** and the **continue** statements are the only JavaScript statements that can "jump out of" a code block.

Syntax:

```
break Labelname;
```

```
continue Labelname;
```

The **continue** statement (with or without a label reference) can only be used to skip one loop iteration.

The **break** statement, without a label reference, can only be used to jump out of a loop or a switch.

With a **label** reference, the break statement can be used to jump out of any code block

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript break</h2>
<p id="demo"></p>

<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var text = "";

list: {
  text += cars[0] + "<br>";
  text += cars[1] + "<br>";
  break list;
  text += cars[2] + "<br>";
  text += cars[3] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Java Script Programming Language

380

JavaScript break

BMW
Volvo

• JavaScript Type Conversion

`Number()` converts to a **Number**, `String()` converts to a **String**, `Boolean()` converts to a Boolean.

JavaScript Data Types

In JavaScript there are 5 different data types that can contain values:

- `string`
- `number`
- `boolean`
- `object`
- `function`

There are 6 types of objects:

- `Object`
- `Date`
- `Array`
- `String`
- `Number`
- `Boolean`

And 2 data types that cannot contain values:

- `null`

Java Script Programming Language

381

- undefined

The typeof Operator

You can use the **typeof** operator to find the data type of a JavaScript variable.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript typeof Operator</h2>

<p>The typeof operator returns the type of a variable, object, function or expression.
</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  typeof "john" + "<br>" +
  typeof 3.14 + "<br>" +
  typeof NaN + "<br>" +
  typeof false + "<br>" +
  typeof [1,2,3,4] + "<br>" +
  typeof {name:'john', age:34} + "<br>" +
  typeof new Date() + "<br>" +
  typeof function () {} + "<br>" +
  typeof myCar + "<br>" +
  typeof null;
</script>

</body>
</html>
```

Java Script Programming Language

382

The JavaScript typeof Operator

The `typeof` operator returns the type of a variable, object, function or expression.

```
string  
number  
number  
boolean  
object  
object  
object  
function  
undefined  
object
```

Please observe:

- The data type of **NaN** is **number**
- The data type of an **array** is **object**
- The data type of a **date** is **object**
- The data type of **null** is **object**
- The data type of an **undefined** variable is **undefined** *
- The data type of a variable that has not been **assigned** a value is also **undefined** *

You cannot use `typeof` to determine if a JavaScript object is an array (or a date)

• JavaScript Bitwise Operations

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits

Java Script Programming Language

383

<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

Examples

Operation	Result	Same as	Result
<code>5 & 1</code>	1	<code>0101 & 0001</code>	0001
<code>5 1</code>	5	<code>0101 0001</code>	0101
<code>~ 5</code>	10	<code>~0101</code>	1010
<code>5 << 1</code>	10	<code>0101 << 1</code>	1010
<code>5 ^ 1</code>	4	<code>0101 ^ 0001</code>	0100
<code>5 >> 1</code>	2	<code>0101 >> 1</code>	0010
<code>5 >>> 1</code>	2	<code>0101 >>> 1</code>	0010

Bitwise AND

When a bitwise AND is performed on a pair of bits, it returns 1 if both bits are 1.

One bit example:

Operation	Result	Operation	Result
<code>0 & 0</code>	0	<code>1111 & 0000</code>	0000
<code>0 & 1</code>	0	<code>1111 & 0001</code>	0001
<code>1 & 0</code>	0	<code>1111 & 0010</code>	0010
<code>1 & 1</code>	1	<code>1111 & 0100</code>	0100

Java Script Programming Language

384

Bitwise OR

When a **bitwise OR** is performed on a pair of bits, it returns 1 if one of the bits are 1:

One bit example:

Operation	Result
0 0	0
0 1	1
1 0	1
1 1	1

Operation	Result
1111 0000	1111
1111 0001	1111
1111 0010	1111
1111 0100	1111

Bitwise XOR

When a **bitwise XOR** is performed on a pair of bits, it returns 1 if the bits are different:

One bit example:

Operation	Result
0 ^ 0	0
0 ^ 1	1
1 ^ 0	1
1 ^ 1	0

Operation	Result
1111 ^ 0000	1111
1111 ^ 0001	1110
1111 ^ 0010	1101
1111 ^ 0100	1011

JavaScript Bitwise AND (&)

Bitwise AND returns 1 only if both bits are 1:

Decimal	Binary
5	0000000000000000000000000000000101
1	0000000000000000000000000000000001
5 & 1	0000000000000000000000000000000001 (1)

Example

Java Script Programming Language

385

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Bitwise AND</h2>

<p id="demo">My First Paragraph.</p>

<script>
document.getElementById("demo").innerHTML = 5 & 1;
</script>

</body>
</html>
```

JavaScript Bitwise AND

1

JavaScript Bitwise OR (|)

Bitwise OR returns 1 if one of the bits are 1:

Decimal	Binary
5	00000000000000000000000000000000101
1	00000000000000000000000000000000001
5 1	00000000000000000000000000000000101 (5)

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Bitwise OR</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 | 1;
</script>

</body>
</html>
```

Java Script Programming Language

386

JavaScript Bitwise OR

5

JavaScript Bitwise XOR (^)

Bitwise XOR returns 1 if the bits are different:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Bitwise XOR</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 ^ 1;
</script>

</body>
</html>
```

JavaScript Bitwise XOR

4

JavaScript Bitwise NOT (~)

Example

Java Script Programming Language

387

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Bitwise NOT</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = ~ 5;
</script>

</body>
</html>
```

JavaScript Bitwise NOT

-6

Converting Decimal to Binary

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Convert Decimal to Binary</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = dec2bin(-5);
function dec2bin(dec){
    return (dec >>> 0).toString(2);
}
</script>

</body>
</html>
```

JavaScript Convert Decimal to Binary

11111111111111111111111111111111011

Converting Binary to Decimal

Example

Java Script Programming

Language

388

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Convert Binary to Decimal</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = bin2dec(101);
function bin2dec(bin){
  return parseInt(bin, 2).toString(10);
}
</script>

</body>
</html>
```

JavaScript Convert Binary to Decimal

5

jQuery Programming Language

JQUERY INTRODUCTION

The purpose of jQuery is to make it much easier to use JavaScript on your website.

What You Should Already Know

Before you start studying jQuery, you should have a basic knowledge of:

- HTML
- CSS
- JavaScript

What is jQuery?

jQuery is a lightweight, "**write less, do more**", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Why jQuery?

There are lots of other JavaScript libraries out there, but **jQuery** is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

jQuery Programming Language

390

- Google
- Microsoft
- IBM
- Netflix

JQUERY GET STARTED

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

- Download the jQuery library from [jQuery.com](#)
- Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

Production version - this is for your live website because it has been minified and compressed

Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from [jQuery.com](#).

The **jQuery** library is a single JavaScript file, and you reference it with the HTML **<script>** tag (notice that the **<script>** tag should be inside the **<head>** section):

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

Tip: Place the downloaded file in the same directory as the pages where you wish to use it.

jQuery CDN

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

Google is an example of someone who host jQuery:

Google CDN:

jQuery Programming Language

392

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
</script>
</head>
<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Click me</button>

</body>
</html>
```

This is a heading

This is a paragraph.

This is another paragraph.

This is a heading

One big advantage of using the hosted jQuery from Google:

Many users already have downloaded jQuery from Google when visiting another site. As a result, it will be loaded from cache when they visit your site, which leads to faster loading time. Also, most CDN's will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time.

• jQuery Syntax

With **jQuery** you select (query) HTML elements and perform "actions" on them.

jQuery Syntax

The **jQuery** syntax is tailor-made for selecting HTML elements and performing some action on the element(s).

Basic syntax is: `$(selector).action()`

A \$ sign to define/access jQuery

- A (selector) to "query (or find)" HTML elements
- A jQuery action() to be performed on the element(s)

Examples:

- `$(this).hide()` - hides the current element.
- `$('p').hide()` - hides all <p> elements.
- `$(".test").hide()` - hides all elements with class="test".
- `$("#test").hide()` - hides the element with id="test".

The Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a document ready event:

```
$(document).ready(function(){

    // jQuery methods go here...

});
```

This is to prevent any **jQuery** code from running before the document is finished loading (is ready).

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

Tip: The jQuery team has also created an even shorter method for the document ready event:

```
$(function(){  
    // jQuery methods go here...  
});
```

Use the syntax you prefer. We think that the document ready event is easier to understand when reading the code.

- ## jQuery Selectors

jQuery selectors are one of the most important parts of the jQuery library.

jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing **CSS Selectors**, and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: **\$()**.

The element Selector

The **jQuery** element selector selects elements based on the element name.

You can select all **<p>** elements on a page like this:

```
$( "p" )
```

Example

When a user clicks on a button, all **<p>** elements will be hidden:

jQuery Programming Language

395

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
</script>
</head>
<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Click me to hide paragraphs</button>

</body>
</html>
```

This is a heading

This is a paragraph.

This is another paragraph.

Click me to hide paragraphs

This is a heading

Click me to hide paragraphs

The #id Selector

The **jQuery #id** selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific **id**, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

Example

When a user clicks on a button, the element with **id="test"** will be hidden:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
</script>
</head>
<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>
<p id="test">This is another paragraph.</p>

<button>Click me</button>

</body>
</html>
```

This is a heading

This is a paragraph.

This is another paragraph.

This is a heading

This is a paragraph.

The .class Selector

The **jQuery .class** selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

jQuery Programming Language

397

```
$(".test")
```

Example

When a user clicks on a button, the elements with **class="test"** will be hidden:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
    });
});
</script>
</head>
<body>

<h2 class="test">This is a heading</h2>

<p class="test">This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Click me</button>

</body>
</html>
```

This is a heading

This is a paragraph.

This is another paragraph.

This is another paragraph.

More Examples of jQuery Selectors

Syntax	Description
<code>\$(*)</code>	Selects all elements
<code>\$(this)</code>	Selects the current HTML element

jQuery Programming Language

398

\$(“p.intro”)	Selects all <p> elements with class="intro"
\$(“p:first”)	Selects the first <p> element
\$(“ul li:first”)	Selects the first element of the first
\$(“ul li:first-child”)	Selects the first element of every
\$(“[href]”)	Selects all elements with an href attribute
\$(“a[target='_blank']”)	Selects all <a> elements with a target attribute value equal to "_blank"
\$(“a[target!='_blank']”)	Selects all <a> elements with a target attribute value NOT equal to "_blank"
\$(“:button”)	Selects all <button> elements and <input> elements of type="button"
\$(“tr:even”)	Selects all even <tr> elements
\$(“tr:odd”)	Selects all odd <tr> elements

Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate **.js** file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the **<head>** section. However, sometimes it is preferable to place them in a separate file, like this (use the **src** attribute to refer to the **.js** file):

Example

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="my_jquery_functions.js"></script>
</head>
```

• jQuery Event Methods

jQuery is tailor-made to respond to events in an HTML page.

What are Events?

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term "**"fires/fired"** is often used with events. Example: "The keypress event is fired, the moment you press a key".

Here are some common DOM events:

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	Keypress	submit	load
dblclick	Keydown	change	resize
mouseenter	Keyup	focus	scroll
mouseleave		blur	unload

jQuery Syntax For Event Methods

In **jQuery**, most **DOM** events have an equivalent **jQuery** method.

To assign a click event to all paragraphs on a page, you can do this:

```
$("p").click();
```

The next step is to define what should happen when the event fires. You must pass a function to the event:

```
$("p").click(function(){  
    // action goes here!!  
});
```

Commonly Used jQuery Event Methods

jQuery Programming Language

400

\$(document).ready()

The **\$(document).ready()** method allows us to execute a function when the document is fully loaded.

click()

The **click()** method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a **<p>** element; hide the current **<p>** element:

Examples:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("p").click(function(){
        $(this).hide();
    });
});
</script>
</head>
<body>

<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

If you click on me, I will disappear.

Click me away!

Click me too!

dblclick()

The **dblclick()** method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

Examples:

jQuery Programming Language

401

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("p").dblclick(function(){
        $(this).hide();
    });
});
</script>
</head>
<body>

<p>If you double-click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

If you double-click on me, I will disappear.

Click me away!

Click me too!

mouseenter()

The **mouseenter()** method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

Examples:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#p1").mouseenter(function(){
        alert("You entered p1!");
    });
});
</script>
</head>
<body>

<p id="p1">Enter this paragraph.</p>

</body>
</html>
```

Enter this paragraph.

mouseleave()

The **mouseleave()** method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer leaves the HTML element:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#p1").mouseleave(function(){
        alert("Bye! You now leave p1!");
    });
});
</script>
</head>
<body>

<p id="p1">This is a paragraph.</p>

</body>
</html>
```

This is a paragraph.

mousedown()

The **mousedown()** method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#p1").mousedown(function(){
        alert("Mouse down over p1!");
    });
});
</script>
</head>
<body>

<p id="p1">This is a paragraph.</p>

</body>
</html>
```

This is a paragraph.

mouseup()

The **mouseup()** method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#p1").mouseup(function(){
        alert("Mouse up over p1!");
    });
});
</script>
</head>
<body>

<p id="p1">This is a paragraph.</p>

</body>
</html>
```

This is a paragraph.

hover()

The **hover()** method takes two functions and is a combination of the **mouseenter()** and **mouseleave()** methods.

The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

Example:

jQuery Programming Language

404

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#p1").hover(function(){
    alert("You entered p1!");
  },
  function(){
    alert("Bye! You now leave p1!");
  });
});
</script>
</head>
<body>

<p id="p1">This is a paragraph.</p>

</body>
</html>
```

This is a paragraph.

focus()

The **focus()** method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("input").focus(function(){
    $(this).css("background-color", "yellow");
  });
  $("input").blur(function(){
    $(this).css("background-color", "green");
  });
});
</script>
</head>
<body>

Name: <input type="text" name="fullname"><br>
Email: <input type="text" name="email">

</body>
</html>
```

jQuery Programming Language

405

Name:

Email:

blur()

The **blur()** method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("input").focus(function(){
        $(this).css("background-color", "yellow");
    });
    $("input").blur(function(){
        $(this).css("background-color", "green");
    });
});
</script>
</head>
<body>

Name: <input type="text" name="fullname"><br>
Email: <input type="text" name="email">

</body>
</html>
```

Name:

Email:

The on() Method

The **on()** method attaches one or more event handlers for the selected elements.

Attach a click event to a **<p>** element:

Example:

jQuery Programming Language

406

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("p").on("click", function(){
        $(this).hide();
    });
});
</script>
</head>
<body>

<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

If you click on me, I will disappear.

Click me away!

Click me too!

Attach multiple event handlers to a **<p>** element:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("p").on({
        mouseenter: function(){
            $(this).css("background-color", "lightgray");
        },
        mouseleave: function(){
            $(this).css("background-color", "lightblue");
        },
        click: function(){
            $(this).css("background-color", "yellow");
        }
    });
});
</script>
</head>
<body>

<p>Click or move the mouse pointer over this paragraph.</p>

</body>
</html>
```

Click or move the mouse pointer over this paragraph.

JQUERY EFFECTS

- Hide and Show

jQuery hide() and show()

With **jQuery**, you can hide and show HTML elements with the **hide()** and **show()** methods:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#hide").click(function(){
    $("p").hide();
  });
  $("#show").click(function(){
    $("p").show();
  });
});
</script>
</head>
<body>

<p>If you click on the "Hide" button, I will disappear.</p>

<button id="hide">Hide</button>
<button id="show">Show</button>

</body>
</html>
```

If you click on the "Hide" button, I will disappear.

If you click on the "Hide" button, I will disappear.

Syntax:

\$(selector).hide(speed,callback);

jQuery Programming Language

408

`$(selector).show(speed,callback);`

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the **hide()** or **show()** method completes.

The following example demonstrates the speed parameter with **hide()**:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide(1000);
    });
});
</script>
</head>
<body>

<button>Hide</button>

<p>This is a paragraph with little content.</p>
<p>This is another small paragraph.</p>

</body>
</html>
```

This is a paragraph with little content.

This is another small paragraph.

jQuery toggle()

You can also toggle between hiding and showing an element with the **toggle()** method.

Shown elements are hidden and hidden elements are shown:

Example:

jQuery Programming Language

409

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").toggle();
    });
});
</script>
</head>
<body>

<button>Toggle between hiding and showing the paragraphs</button>

<p>This is a paragraph with little content.</p>
<p>This is another small paragraph.</p>

</body>
</html>
```

Toggle between hiding and showing the paragraphs

This is a paragraph with little content.

This is another small paragraph.

Toggle between hiding and showing the paragraphs

Syntax:

\$(selector).toggle(speed,callback);

The optional speed parameter can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after **toggle()** completes.

• jQuery Effects – Fading

jQuery Fading Methods

With **jQuery** you can fade an element in and out of visibility.

jQuery has the following fade methods:

- fadeIn()

- fadeOut()
- fadeToggle()
- fadeTo()

jQuery fadeIn() Method

The jQuery **fadeIn()** method is used to fade in a hidden element.

Syntax:

\$(selector).fadeIn(speed,callback);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the **fadeIn()** method with different parameters:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").fadeIn();
        $("#div2").fadeIn("slow");
        $("#div3").fadeIn(3000);
    });
});
</script>
</head>
<body>

<p>Demonstrate fadeIn() with different parameters.</p>

<button>Click to fade in boxes</button><br><br>

<div id="div1" style="width:80px;height:80px;display:none;background-color:red;"></div>
<br>
<div id="div2" style="width:80px;height:80px;display:none;background-color:green;">
</div><br>
<div id="div3" style="width:80px;height:80px;display:none;background-color:blue;"></div>

</body>
</html>
```

jQuery Programming Language

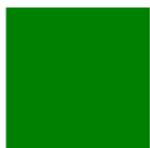
411

Demonstrate fadeIn() with different parameters.

Click to fade in boxes

Demonstrate fadeIn() with different parameters.

Click to fade in boxes



jQuery fadeOut() Method

The jQuery **fadeOut()** method is used to fade out a visible element.

Syntax:

`$(selector).fadeOut(speed,callback);`

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the **fadeOut()** method with different parameters:

Example:

jQuery Programming Language

412

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").fadeOut();
        $("#div2").fadeOut("slow");
        $("#div3").fadeOut(3000);
    });
});
</script>
</head>
<body>

<p>Demonstrate fadeOut() with different parameters.</p>

<button>Click to fade out boxes</button><br><br>

<div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>

</body>
</html>
```

Demonstrate fadeOut() with different parameters.

Click to fade out boxes



Demonstrate fadeOut() with different parameters.

Click to fade out boxes

jQuery fadeToggle() Method

jQuery Programming Language

413

The jQuery **fadeToggle()** method toggles between the **fadeIn()** and **fadeOut()** methods.

If the elements are faded out, **fadeToggle()** will fade them in.

If the elements are faded in, **fadeToggle()** will fade them out.

Syntax:

\$(selector).fadeToggle(speed,callback);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the **fadeToggle()** method with different parameters:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").fadeToggle();
        $("#div2").fadeToggle("slow");
        $("#div3").fadeToggle(3000);
    });
});
</script>
</head>
<body>

<p>Demonstrate fadeToggle() with different speed parameters.</p>

<button>Click to fade in/out boxes</button><br><br>

<div id="div1" style="width:80px;height:80px;background-color:red;"></div>
<br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div>
<br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>

</body>
</html>
```

Demonstrate fadeToggle() with different speed parameters.

Click to fade in/out boxes



Demonstrate fadeToggle() with different speed parameters.

Click to fade in/out boxes

jQuery fadeTo() Method

The jQuery **fadeTo()** method allows fading to a given opacity (value between 0 and 1).

Syntax:

`$(selector).fadeTo(speed,opacity,callback);`

The required speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The required opacity parameter in the **fadeTo()** method specifies fading to a given opacity (value between 0 and 1).

The optional callback parameter is a function to be executed after the function completes.

The following example demonstrates the **fadeTo()** method with different parameters:

Example:

jQuery Programming Language

415

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").fadeTo("slow", 0.15);
        $("#div2").fadeTo("slow", 0.4);
        $("#div3").fadeTo("slow", 0.7);
    });
});
</script>
</head>
<body>

<p>Demonstrate fadeTo() with different parameters.</p>

<button>Click to fade boxes</button><br><br>

<div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>

</body>
</html>
```

Demonstrate fadeTo() with different parameters.

Click to fade boxes



Demonstrate fadeTo() with different parameters.

Click to fade boxes



- **jQuery Effects – Sliding**

jQuery Sliding Methods

With jQuery you can create a sliding effect on elements.

jQuery has the following slide methods:

- slideDown()
- slideUp()
- slideToggle()

jQuery slideDown() Method

The jQuery **slideDown()** method is used to slide down an element.

Syntax:

`$(selector).slideDown(speed,callback);`

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the **slideDown()** method:

jQuery Programming Language

417

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#flip").click(function(){
    $("#panel").slideDown("slow");
  });
});
</script>
<style>
#panel, #flip {
  padding: 5px;
  text-align: center;
  background-color: #e5eecc;
  border: solid 1px #c3c3c3;
}
#panel {
  padding: 50px;
  display: none;
}
</style>
</head>
<body>

<div id="flip">Click to slide down panel</div>
<div id="panel">Hello world!</div>

</body>
</html>
```

Click to slide down panel

Click to slide down panel

Hello world!

jQuery slideUp() Method

The jQuery **slideUp()** method is used to slide up an element.

Syntax:

jQuery Programming Language

418

`$(selector).slideUp(speed,callback);`

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

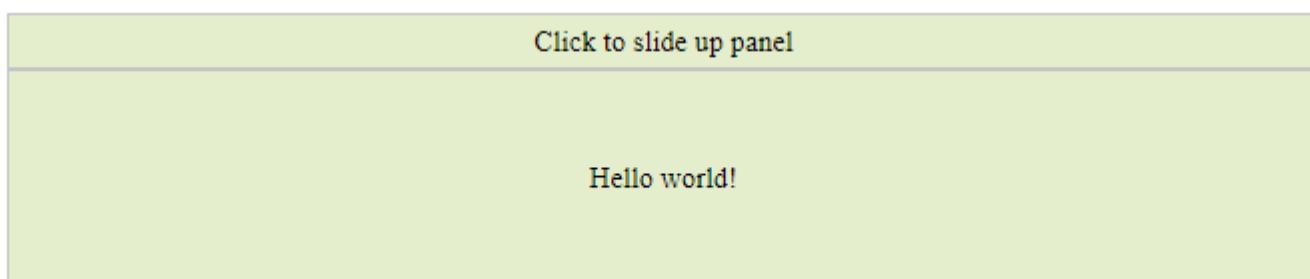
The following example demonstrates the **slideUp()** method:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#flip").click(function(){
    $("#panel").slideUp("slow");
  });
});
</script>
<style>
#panel, #flip {
  padding: 5px;
  text-align: center;
  background-color: #e5eecc;
  border: solid 1px #c3c3c3;
}
#panel {
  padding: 50px;
}
</style>
</head>
<body>

<div id="flip">Click to slide up panel</div>
<div id="panel">Hello world!</div>

</body>
</html>
```



Click to slide up panel

jQuery slideToggle() Method

The jQuery **slideToggle()** method toggles between the **slideDown()** and **slideUp()** methods.

If the elements have been slid down, **slideToggle()** will slide them up.

If the elements have been slid up, **slideToggle()** will slide them down.

`$(selector).slideToggle(speed,callback);`

The optional speed parameter can take the following values: "slow", "fast", milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the **slideToggle()** method:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#flip").click(function(){
    $("#panel").slideToggle("slow");
  });
});
</script>
<style>
#panel, #flip {
  padding: 5px;
  text-align: center;
  background-color: #e5eecc;
  border: solid 1px #c3c3c3;
}
#panel {
  padding: 50px;
  display: none;
}
</style>
</head>
<body>

<div id="flip">Click to slide the panel down or up</div>
<div id="panel">Hello world!</div>
```

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

Click to slide the panel down or up

Click to slide the panel down or up

Hello world!

- **jQuery Effects – Animation**

jQuery Animations - The animate() Method

The jQuery **animate()** method is used to create custom animations.

Syntax:

```
$(selector).animate({params},speed,callback);
```

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

The following example demonstrates a simple use of the **animate()** method; it moves a **<div>** element to the right, until it has reached a left property of 250px:

Example:

jQuery Programming Language

421

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({left: '250px'});
    });
});
</script>
</head>
<body>

<button>Start Animation</button>

<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



jQuery animate() - Manipulate Multiple Properties

Notice that multiple properties can be animated at the same time:

Example:

jQuery Programming Language

422

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({
            left: '250px',
            opacity: '0.5',
            height: '150px',
            width: '150px'
        });
    });
});
</script>
</head>
<body>

<button>Start Animation</button>
```

pBy default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

```
<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>
</body>
</html>
```

Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



jQuery animate() - Using Relative Values

It is also possible to define relative values (the value is then relative to the element's current value). This is done by putting `+ =` or `- =` in front of the value:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({
            left: '250px',
            height: '+=150px',
            width: '+=150px'
        });
    });
});
</script>
</head>
<body>

<button>Start Animation</button>

<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



jQuery Programming Language

424

[Start Animation](#)

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



jQuery animate() - Using Pre-defined Values

You can even specify a property's animation value as "show", "hide", or "toggle":

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({
            height: 'toggle'
        });
    });
});
</script>
</head>
<body>

<p>Click the button multiple times to toggle the animation.</p>

<button>Start Animation</button>

<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

jQuery Programming Language

425

Click the button multiple times to toggle the animation.

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



Click the button multiple times to toggle the animation.

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

jQuery animate() - Uses Queue Functionality

By default, jQuery comes with queue functionality for animations.

This means that if you write multiple **animate()** calls after each other, jQuery creates an "internal" queue with these method calls. Then it runs the animate calls ONE by ONE.

So, if you want to perform different animations after each other, we take advantage of the queue functionality:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        var div = $("div");
        div.animate({height: '300px', opacity: '0.4'}, "slow");
        div.animate({width: '300px', opacity: '0.8'}, "slow");
        div.animate({height: '100px', opacity: '0.4'}, "slow");
        div.animate({width: '100px', opacity: '0.8'}, "slow");
    });
});
</script>
</head>
<body>

<button>Start Animation</button>

<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

jQuery Programming Language

426

[Start Animation](#)

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



The example below first moves the <div> element to the right, and then increases the font size of the text:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var div = $("div");
    div.animate({left: '100px'}, "slow");
    div.animate({fontSize: '3em'}, "slow");
  });
});
</script>
</head>
<body>

<button>Start Animation</button>

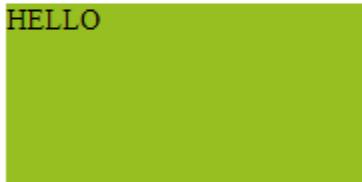
<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:200px;position:absolute;">HELLO</div>

</body>
</html>
```

[Start Animation](#)

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



[Start Animation](#)

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



- ## jQuery Stop Animations

jQuery stop() Method

The jQuery **stop()** method is used to stop an animation or effect before it is finished.

The **stop()** method works for all jQuery effect functions, including sliding, fading and custom animations.

Syntax:

`$(selector).stop(stopAll,goToEnd);`

The optional **stopAll** parameter specifies whether also the animation queue should be cleared or not. Default is false, which means that only the active animation will be stopped, allowing any queued animations to be performed afterwards.

The optional **goToEnd** parameter specifies whether or not to complete the current animation immediately. Default is false.

So, by default, the **stop()** method kills the current animation being performed on the selected element.

jQuery Programming Language

428

The following example demonstrates the **stop()** method, with no parameters:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("#flip").click(function(){
        $("#panel").slideDown(5000);
    });
    $("#stop").click(function(){
        $("#panel").stop();
    });
});
</script>
<style>
#panel, #flip {
    padding: 5px;
    font-size: 18px;
    text-align: center;
    background-color: #555;
    color: white;
    border: solid 1px #666;
    border-radius: 3px;
}
#panel {
    padding: 50px;
    display: none;
}
</style>
</head>
<body>

<button id="stop">Stop sliding</button>

<div id="flip">Click to slide down panel</div>
<div id="panel">Hello world!</div>

</body>
</html>
```



• jQuery Callback Functions

A callback function is executed after the current effect is 100% finished.

jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical **syntax:** `$(selector).hide(speed,callback);`

Examples

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

Example with Callback

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide("slow", function(){
            alert("The paragraph is now hidden");
        });
    });
});
</script>
</head>
<body>

<button>Hide</button>

<p>This is a paragraph with little content.</p>

</body>
</html>
```

Hide

This is a paragraph with little content.

The example below has no callback parameter, and the alert box will be displayed before the hide effect is completed:

Example without Callback

jQuery Programming Language

430

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide(1000);
        alert("The paragraph is now hidden");
    });
});
</script>
</head>
<body>

<button>Hide</button>

<p>This is a paragraph with little content.</p>

</body>
</html>
```

[Hide](#)

This is a paragraph with little content.

An embedded page on this page says

The paragraph is now hidden

OK

• jQuery – Chaining

With jQuery, you can chain together actions/methods.

Chaining allows us to run multiple jQuery methods (on the same element) within a single statement.

jQuery Method Chaining

Until now we have been writing **jQuery** statements one at a time (one after the other).

However, there is a technique called chaining, that allows us to run multiple jQuery commands, one after the other, on the same element(s).

Tip: This way, browsers do not have to find the same element(s) more than once.

jQuery Programming Language

431

To chain an action, you simply append the action to the previous action.

The following example chains together the **css()**, **slideUp()**, and **slideDown()** methods.

The "**p1**" element first changes to red, then it slides up, and then it slides down:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#p1").css("color", "red").slideUp(2000).slideDown(2000);
    });
});
</script>
</head>
<body>

<p id="p1">jQuery is fun!!</p>

<button>Click me</button>

</body>
</html>
```

jQuery is fun!!

jQuery is fun!!

We could also have added more method calls if needed.

Tip: When chaining, the line of code could become quite long. However, jQuery is not very strict on the syntax; you can format it like you want, including line breaks and indentations.

This also works just fine:

Example

jQuery Programming Language

432

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#p1").css("color", "red")
        .slideUp(2000)
        .slideDown(2000);
    });
});
</script>
</head>
<body>

<p id="p1">jQuery is fun!!</p>

<button>Click me</button>

</body>
</html>
```

jQuery is fun!!

jQuery is fun!!

jQuery throws away extra whitespace and executes the lines above as one long line of code.

JQUERY - GET CONTENT AND ATTRIBUTES

jQuery contains powerful methods for changing and manipulating HTML elements and attributes.

jQuery DOM Manipulation

One very important part of jQuery is the possibility to manipulate the DOM.

jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.

Get Content - `text()`, `html()`, and `val()`

Three simples, but useful, jQuery methods for DOM manipulation are:

text() - Sets or returns the text content of selected elements

html() - Sets or returns the content of selected elements (including HTML markup)

val() - Sets or returns the value of form fields

The following example demonstrates how to get content with the jQuery `text()` and `html()` methods:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    alert("Text: " + $("#test").text());
  });
  $("#btn2").click(function(){
    alert("HTML: " + $("#test").html());
  });
});
</script>
</head>
<body>

<p id="test">This is some <b>bold</b> text in a paragraph.</p>

<button id="btn1">Show Text</button>
<button id="btn2">Show HTML</button>

</body>
</html>
```

jQuery Programming Language

434

This is some **bold** text in a paragraph.

Show Text

Show HTML

An embedded page on this page says

Text: This is some bold text in a paragraph.

OK

An embedded page on this page says

HTML: This is some bold text in a paragraph.

OK

The following example demonstrates how to get the value of an input field with the jQuery val() method:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        alert("Value: " + $("#test").val());
    });
});
</script>
</head>
<body>

<p>Name: <input type="text" id="test" value="Mickey Mouse"></p>

<button>Show Value</button>

</body>
</html>
```

jQuery Programming Language

435

Name:

Show Value

An embedded page on this page says

Value: Mickey Mouse

OK

Get Attributes - attr()

The jQuery **attr()** method is used to get attribute values.

The following example demonstrates how to get the value of the **href** attribute in a link:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        alert($("#w3s").attr("href"));
    });
});
</script>
</head>
<body>

<p><a href="https://www.w3schools.com" id="w3s">W3Schools.com</a></p>

<button>Show href Value</button>

</body>
</html>
```

[W3Schools.com](#)

Show href Value

- **jQuery - Set Content and Attributes**

Set Content - `text()`, `html()`, and `val()`

We will use the same three methods from the previous page to set content:

text() - Sets or returns the text content of selected elements

html() - Sets or returns the content of selected elements (including HTML markup)

val() - Sets or returns the value of form fields

The following example demonstrates how to set content with the jQuery `text()`, `html()`, and `val()` methods:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("#btn1").click(function(){
        $("#test1").text("Hello world!");
    });
    $("#btn2").click(function(){
        $("#test2").html("<b>Hello world!</b>");
    });
    $("#btn3").click(function(){
        $("#test3").val("Dolly Duck");
    });
});
</script>
</head>
<body>

<p id="test1">This is a paragraph.</p>
<p id="test2">This is another paragraph.</p>

<p>Input field: <input type="text" id="test3" value="Mickey Mouse"></p>

<button id="btn1">Set Text</button>
<button id="btn2">Set HTML</button>
<button id="btn3">Set Value</button>

</body>
</html>
```

jQuery Programming Language

437

This is a paragraph.

This is another paragraph.

Input field:

A Callback Function for text(), html(), and val()

All of the three jQuery methods above: **text()**, **html()**, and **val()**, also come with a callback function. The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) value. You then return the string you wish to use as the new value from the function.

The following example demonstrates **text()** and **html()** with a callback function:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("#btn1").click(function(){
        $("#test1").text(function(i, origText){
            return "Old text: " + origText + " New text: Hello world! (index: " + i + ")";
        });
    });

    $("#btn2").click(function(){
        $("#test2").html(function(i, origText){
            return "Old html: " + origText + " New html: Hello <b>world!</b> (index: " + i + ")";
        });
    });
});
</script>
</head>
<body>

<p id="test1">This is a <b>bold</b> paragraph.</p>
<p id="test2">This is another <b>bold</b> paragraph.</p>

<button id="btn1">Show Old/New Text</button>
<button id="btn2">Show Old/New HTML</button>

</body>
</html>
```

jQuery Programming Language

438

This is a **bold** paragraph.

This is another **bold** paragraph.

Show Old/New Text

Show Old/New HTML

Old text: This is a bold paragraph. New text: Hello world! (index: 0)

This is another **bold** paragraph.

Show Old/New Text

Show Old/New HTML

Old text: This is a bold paragraph. New text: Hello world! (index: 0)

Old html: This is another **bold** paragraph. New html: Hello **world!** (index: 0)

Show Old/New Text

Show Old/New HTML

Set Attributes - attr()

The jQuery **attr()** method is also used to set/change attribute values.

The following example demonstrates how to change (set) the value of the **href** attribute in a link:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#w3s").attr("href", "https://www.w3schools.com/jquery/");
    });
});
</script>
</head>
<body>

<p><a href="https://www.w3schools.com" id="w3s">W3Schools.com</a></p>

<button>Change href Value</button>

<p>Mouse over the link (or click on it) to see that the value of the href attribute has changed.</p>

</body>
</html>
```

jQuery Programming Language

439

[W3Schools.com](https://www.w3schools.com)

[Change href Value](#)

Mouse over the link (or click on it) to see that the value of the href attribute has changed.

[W3Schools.com](https://www.w3schools.com)

[Change href Value](#)

Mouse over the link (or click on it) to see that the value of the href attribute has changed.

A Callback Function for attr()

The jQuery method **attr()**, also comes with a callback function. The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) attribute value. You then return the string you wish to use as the new attribute value from the function.

The following example demonstrates **attr()** with a callback function:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#w3s").attr("href", function(i, origValue){
            return origValue + "/jquery/";
        });
    });
});
</script>
</head>
<body>

<p><a href="https://www.w3schools.com" id="w3s">W3Schools.com</a></p>

<button>Change href Value</button>

<p>Mouse over the link (or click on it) to see that the value of the href attribute has
changed.</p>

</body>
</html>
```

jQuery Programming Language

440

[W3Schools.com](#)

[Change href Value](#)

Mouse over the link (or click on it) to see that the value of the href attribute has changed.

[W3Schools.com](#)

[Change href Value](#)

Mouse over the link (or click on it) to see that the value of the href attribute has changed.

• jQuery - Add Elements

Add New HTML Content

We will look at four jQuery methods that are used to add new content:

- **append()** - Inserts content at the end of the selected elements
- **prepend()** - Inserts content at the beginning of the selected elements
- **after()** - Inserts content after the selected elements
- **before()** - Inserts content before the selected elements

jQuery append() Method

The jQuery **append()** method inserts content AT THE END of the selected HTML elements.

Example

jQuery Programming Language

441

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("#btn1").click(function(){
        $("p").append(" <b>Appended text</b>.");
    });

    $("#btn2").click(function(){
        $("ol").append("<li>Appended item</li>");
    });
});
</script>
</head>
<body>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<ol>
    <li>List item 1</li>
    <li>List item 2</li>
    <li>List item 3</li>
</ol>

<button id="btn1">Append text</button>
<button id="btn2">Append list items</button>

</body>
</html>
```

This is a paragraph.

This is another paragraph.

1. List item 1
2. List item 2
3. List item 3

This is a paragraph. **Appended text**.

This is another paragraph. **Appended text**.

1. List item 1
2. List item 2
3. List item 3

jQuery Programming Language

442

This is a paragraph. **Appended text.**

This is another paragraph. **Appended text.**

1. List item 1
2. List item 2
3. List item 3
4. Appended item

jQuery prepend() Method

The jQuery **prepend()** method inserts content AT THE BEGINNING of the selected HTML elements.

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("p").prepend("<b>Prepended text</b>. ");
  });
  $("#btn2").click(function(){
    $("ol").prepend("<li>Prepended item</li>");
  });
});
</script>
</head>
<body>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<ol>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ol>

<button id="btn1">Prepend text</button>
<button id="btn2">Prepend list item</button>

</body>
</html>
```

jQuery Programming Language

443

This is a paragraph.

This is another paragraph.

1. List item 1
2. List item 2
3. List item 3

[Prepend text](#) [Prepend list item](#)

Prepended text. This is a paragraph.

Prepended text. This is another paragraph.

1. List item 1
2. List item 2
3. List item 3

[Prepend text](#) [Prepend list item](#)

Prepended text. This is a paragraph.

Prepended text. This is another paragraph.

1. Prepended item
2. List item 1
3. List item 2
4. List item 3

[Prepend text](#) [Prepend list item](#)

Add Several New Elements With `append()` and `prepend()`

In both examples above, we have only inserted some text/HTML at the beginning/end of the selected HTML elements.

However, both the **append()** and **prepend()** methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the examples above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we append the new elements to the text with the **append()** method (this would have worked for `prepend()` too) :

Example

jQuery Programming Language

444

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
function appendText() {
    var txt1 = "<p>Text.</p>";           // Create text with HTML
    var txt2 = $("<p></p>").text("Text."); // Create text with jQuery
    var txt3 = document.createElement("p");
    txt3.innerHTML = "Text.";           // Create text with DOM
    $("body").append(txt1, txt2, txt3); // Append new elements
}
</script>
</head>
<body>

<p>This is a paragraph.</p>
<button onclick="appendText()">Append text</button>

</body>
</html>
```

This is a paragraph.

Append text

jQuery after() and before() Methods

The jQuery **after()** method inserts content AFTER the selected HTML elements.

The jQuery **before()** method inserts content BEFORE the selected HTML elements.

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#btn1").click(function(){
        $("img").before("<b>Before</b>");
    });
    $("#btn2").click(function(){
        $("img").after("<i>After</i>");
    });
});
</script>
</head>
<body>

<br><br>

<button id="btn1">Insert before</button>
<button id="btn2">Insert after</button>

</body>
</html>
```

jQuery Programming Language

445



[Insert before](#) [Insert after](#)



Before

[Insert before](#) [Insert after](#)



After

[Insert before](#) [Insert after](#)

Add Several New Elements With after() and before()

Also, both the after() and before() methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the example above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we insert the new elements to the text with the after() method (this would have worked for before() too) :

Example

jQuery Programming Language

446

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
function afterText() {
    var txt1 = "<b>I </b>";           // Create element with HTML
    var txt2 = $("<i></i>").text("love "); // Create with jQuery
    var txt3 = document.createElement("b"); // Create with DOM
    txt3.innerHTML = "jQuery!";
    $("img").after(txt1, txt2, txt3);    // Insert new elements after img
}
</script>
</head>
<body>



<p>Click the button to insert text after the image.</p>

<button onclick="afterText()">Insert after</button>

</body>
</html>
```



Click the button to insert text after the image.



I *love* jQuery!

Click the button to insert text after the image.

- **jQuery - Remove Elements**

Remove Elements/Content

To remove elements and content, there are mainly two jQuery methods:

- **remove()** - Removes the selected element (and its child elements)
- **empty()** - Removes the child elements from the selected element

jQuery remove() Method

The jQuery **remove()** method removes the selected element(s) and its child elements.

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").remove();
  });
});
</script>
</head>
<body>

<div id="div1" style="height:100px;width:300px;border:1px solid black;background-color:yellow;">
This is some text in the div.
<p>This is a paragraph in the div.</p>
<p>This is another paragraph in the div.</p>

</div>
<br>

<button>Remove div element</button>
</body>
</html>
```

This is some text in the div.
This is a paragraph in the div.
This is another paragraph in the div.

Remove div element

jQuery Programming Language

448

Remove div element

jQuery empty() Method

The jQuery **empty()** method removes the child elements of the selected element(s).

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").empty();
  });
});
</script>
</head>
<body>

<div id="div1" style="height:100px;width:300px;border:1px solid black;background-color:yellow;">

This is some text in the div.
<p>This is a paragraph in the div.</p>
<p>This is another paragraph in the div.</p>

</div>
<br>

<button>Empty the div element</button>

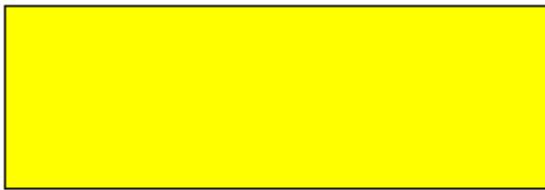
</body>
</html>
```

This is some text in the div.

This is a paragraph in the div.

This is another paragraph in the div.

Empty the div element



Empty the div element

Filter the Elements to be Removed

The jQuery **remove()** method also accepts one parameter, which allows you to filter the elements to be removed.

The parameter can be any of the jQuery selector syntaxes.

The following example removes all `<p>` elements with `class="test"`:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").remove(".test");
  });
});
</script>
<style>
.test {
  color: red;
  font-size: 20px;
}
</style>
</head>
<body>

<p>This is a paragraph.</p>
<p class="test">This is another paragraph.</p>
<p class="test">This is another paragraph.</p>

<button>Remove all p elements with class="test"</button>

</body>
</html>
```

jQuery Programming Language

450

This is a paragraph.

This is another paragraph.

This is another paragraph.

Remove all p elements with class="test"

This is a paragraph.

Remove all p elements with class="test"

This example removes all **<p>** elements with **class="test"** or **class="demo"**:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").remove(".test, .demo");
    });
});
</script>
<style>
.test {
    color: red;
    font-size: 20px;
}

.demo {
    color: green;
    font-size: 25px;
}
</style>
</head>
<body>

<p>This is a paragraph.</p>
<p class="test">This is p element with class="test".</p>
<p class="test">This is p element with class="test".</p>
<p class="demo">This is p element with class="demo".</p>

<button>Remove all p elements with class="test" and class="demo"</button>

</body>
</html>
```

This is a paragraph.

This is p element with class="test".

This is p element with class="test".

This is p element with class="demo".

[Remove all p elements with class="test" and class="demo"](#)

This is a paragraph.

[Remove all p elements with class="test" and class="demo"](#)

- **jQuery - Get and Set CSS Classes**

jQuery Manipulating CSS

jQuery has several methods for CSS manipulation. We will look at the following methods:

- **addClass()** - Adds one or more classes to the selected elements
- **removeClass()** - Removes one or more classes from the selected elements
- **toggleClass()** - Toggles between adding/removing classes from the selected elements
- **css()** - Sets or returns the style attribute

jQuery addClass() Method

The following example shows how to add class attributes to different elements. Of course, you can select multiple elements, when adding classes:

Example

jQuery Programming Language

452

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("h1, h2, p").addClass("blue");
        $("div").addClass("important");
    });
});
</script>
<style>
.important {
    font-weight: bold;
    font-size: xx-large;
}
.blue {
    color: blue;
}
</style>
</head>
<body>

<h1>Heading 1</h1>
<h2>Heading 2</h2>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<div>This is some important text!</div><br>

<button>Add classes to elements</button>

</body>
</html>
```

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

This is some important text!

Add classes to elements

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

This is some important text!

Add classes to elements

You can also specify multiple classes within the **addClass()** method:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").addClass("important blue");
    });
});
</script>
<style>
.important {
    font-weight: bold;
    font-size: xx-large;
}

.blue {
    color: blue;
}
</style>
</head>
<body>

<div id="div1">This is some text.</div>
<div id="div2">This is some text.</div>
<br>

<button>Add classes to first div element</button>

</body>
</html>
```

jQuery Programming Language

454

This is some text.
This is some text.

Add classes to first div element

This is some text.

This is some text.

Add classes to first div element

jQuery removeClass() Method

The following example shows how to remove a specific class attribute from different elements:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("h1, h2, p").removeClass("blue");
  });
});
</script>
<style>
.blue {
  color: blue;
}
</style>
</head>
<body>

<h1 class="blue">Heading 1</h1>
<h2 class="blue">Heading 2</h2>

<p class="blue">This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Remove class from elements</button>

</body>
</html>
```

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

[Remove class from elements](#)

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

[Remove class from elements](#)

jQuery toggleClass() Method

The following example will show how to use the jQuery **toggleClass()** method. This method toggles between **adding/removing** classes from the selected elements:

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("h1, h2, p").toggleClass("blue");
    });
});
</script>
<style>
.blue {
    color: blue;
}
</style>
</head>
<body>

<h1>Heading 1</h1>
<h2>Heading 2</h2>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Toggle class</button>

</body>
</html>
```

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

[Toggle class](#)

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

[Toggle class](#)

JSON Programming Language

JSON INTRODUCTION

JSON: JavaScript Object Notation.

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

Exchanging Data

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

Sending Data

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a JavaScript object into a JSON string, and send it to the server.</h2>

<script>
var myObj = { name: "John", age: 31, city: "New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
</script>

</body>
</html>
```

demo_json.php:

John from New York is 31

Receiving Data

JSON Programming Language

458

If you receive data in **JSON** format, you can convert it into a JavaScript object:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a string written in JSON format, into a JavaScript object.</h2>

<p id="demo"></p>

<script>
var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

Convert a string written in JSON format, into a JavaScript object.

John

Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Store and retrieve data from local storage.</h2>

<p id="demo"></p>

<script>
var myObj, myJSON, text, obj;

// Storing data:
myObj = { name: "John", age: 31, city: "New York" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>

</body>
</html>
```

Store and retrieve data from local storage.

John

What is JSON?

- **JSON** stands for JavaScript Object Notation
- **JSON** is a lightweight data-interchange format
- **JSON** is "self-describing" and easy to understand
- **JSON** is language independent *

Why use JSON?

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

JSON SYNTAX

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example

```
"name": "John"
```

JSON - Evaluates to JavaScript Objects

The **JSON** format is almost identical to JavaScript objects.

In **JSON**, keys must be strings, written with double quotes:

JSON

```
{ "name": "John" }
```

In JavaScript, keys can be strings, numbers, or identifier names:

JavaScript

```
{ name: "John" }
```

JSON Values

In JSON, values must be one of the following data types:

- a string

JSON Programming Language

461

- a number
- an object (JSON object)
- an array
- a boolean
- null

In JavaScript values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

In JSON, string values must be written with double quotes:

JSON

```
{ "name": "John" }
```

In JavaScript, you can write string values with double or single quotes:

JavaScript

```
{ name: 'John' }
```

JSON Uses JavaScript Syntax

Because **JSON** syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

Example

```
var person = { name: "John", age: 31, city: "New York" };
```

You can access a JavaScript object like this:

Example

JSON Programming Language

462

```
<!DOCTYPE html>
<html>
<body>

<p>Access a JavaScript object:</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = { name: "John", age: 30, city: "New York" };
x = myObj.name;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Access a JavaScript object:

John

It can also be accessed like this:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Access a JavaScript object using the bracket notation:</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = { name: "John", age: 30, city: "New York" };
x = myObj["name"];
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Access a JavaScript object using the bracket notation:

John

Data can be modified like this:

Example

JSON Programming Language

463

```
<!DOCTYPE html>
<html>
<body>

<p>Modify a JavaScript object:</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = { name: "John", age: 30, city: "New York" };
myObj.name = "Gilbert";
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

Modify a JavaScript object:

Gilbert

It can also be modified like this:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Modify a JavaScript object using the bracket notation:</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = { name: "John", age: 30, city: "New York" };
myObj["name"] = "Gilbert";
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

Modify a JavaScript object using the bracket notation:

Gilbert

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

JSON VS XML

Both **JSON** and **XML** can be used to receive data from a web server.

The following JSON and XML examples both define an employee's object, with an array of 3 employees:

JSON Example

```
{"employees": [
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON is Like XML Because

- Both **JSON** and **XML** are "self describing" (human readable)
- Both **JSON** and **XML** are hierarchical (values within values)
- Both **JSON** and **XML** can be parsed and used by lots of programming languages
- Both **JSON** and **XML** can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter

- JSON is quicker to read and write
- JSON can use arrays

The biggest difference is:

XML has to be parsed with an **XML parser**. **JSON** can be parsed by a standard JavaScript function.

Why JSON is Better Than XML

XML is much more difficult to parse than **JSON**.

JSON is parsed into a ready-to-use JavaScript object.

For **AJAX** applications, **JSON** is faster and easier than **XML**:

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- **JSON.Parse** the JSON string

JS JSON

• JSON Data Types

Valid Data Types

In **JSON**, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

JSON values cannot be one of the following data types:

- a function
- a date
- undefined

JSON Strings

Strings in **JSON** must be written in double quotes.

Example

```
{ "name": "John" }
```

JSON Numbers

Numbers in **JSON** must be an integer or a floating point.

Example

```
{ "age": 30 }
```

JSON Objects

Values in **JSON** can be objects.

JSON Programming Language

467

Example

```
{  
  "employee":{ "name":"John", "age":30, "city":"New York" }  
}
```

JSON Arrays

Values in JSON can be arrays.

Example

```
{  
  "employees":[ "John", "Anna", "Peter" ]  
}
```

JSON Booleans

Values in JSON can be true/false.

Example

```
{ "sale":true }
```

JSON null

Values in JSON can be null.

Example

```
{ "middlename":null }
```

• JSON.parse()

A common use of JSON is to exchange data to/from a web server.

When receiving data from a web server, the data is always a string.

Parse the data with **JSON.parse()**, and the data becomes a JavaScript object.

Example - Parsing JSON

Imagine we received this text from a web server:

```
'{ "name":"John", "age":30, "city":"New York"}'
```

JSON Programming Language

468

Use the JavaScript function **JSON.parse()** to convert text into a JavaScript object:

```
var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Create Object from JSON String</h2>

<p id="demo"></p>

<script>
var txt = '{"name":"John", "age":30, "city":"New York"}'
var obj = JSON.parse(txt);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
</script>

</body>
</html>
```

Create Object from JSON String

John, 30

JSON From the Server

You can request **JSON** from the server by using an **AJAX** request

As long as the response from the server is written in JSON format, you can parse the string into a JavaScript object.

Example

JSON Programming Language

469

```
<!DOCTYPE html>
<html>
<body>

<h2>Use the XMLHttpRequest to get the content of a file.</h2>
<p>The content is written in JSON format, and can easily be converted into a JavaScript object.</p>

<p id="demo"></p>

<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var myObj = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myObj.name;
  }
};
xmlhttp.open("GET", "json_demo.txt", true);
xmlhttp.send();
</script>

<p>Take a look at <a href="json_demo.txt" target="_blank">json_demo.txt</a></p>

</body>
</html>
```

Use the XMLHttpRequest to get the content of a file.

The content is written in JSON format, and can easily be converted into a JavaScript object.

John

Take a look at json_demo.txt

Array as JSON

When using the **JSON.parse()** on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

Example

The **JSON** returned from the server is an array:

JSON Programming Language

470

```
<!DOCTYPE html>
<html>
<body>

<h2>Content as Array.</h2>
<p>Content written as an JSON array will be converted into a JavaScript array.</p>

<p id="demo"></p>

<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var myArr = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myArr[0];
  }
};
xmlhttp.open("GET", "json_demo_array.txt", true);
xmlhttp.send();
</script>

<p>Take a look at <a href="json_demo_array.txt" target="_blank">json_demo_array.txt</a>
</p>

</body>
</html>
```

Content as Array.

Content written as an JSON array will be converted into a JavaScript array.

Ford

Take a look at json_demo_array.txt

Exceptions

Parsing Dates

Date objects are not allowed in **JSON**.

If you need to include a date, write it as a string.

You can convert it back into a date object later:

Example

Convert a string into a date:

JSON Programming Language

471

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a string into a date object.</h2>

<p id="demo"></p>

<script>
var text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text);
obj.birth = new Date(obj.birth);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
</script>

</body>
</html>
```

Convert a string into a date object.

John, Sun Dec 14 1986 03:00:00 GMT+0300 (Arabian Standard Time)

Or, you can use the second parameter, of the **JSON.parse()** function, called reviver.

The reviver parameter is a function that checks each property, before returning the value.

Example

Convert a string into a date, using the reviver function:

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a string into a date object.</h2>

<p id="demo"></p>

<script>
var text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text, function (key, value) {
  if (key == "birth") {
    return new Date(value);
  } else {
    return value;
  }
});
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
</script>

</body>
</html>
```

JSON Programming Language

472

Convert a string into a date object.

John, Sun Dec 14 1986 03:00:00 GMT+0300 (Arabian Standard Time)

Parsing Functions

Functions are not allowed in JSON.

If you need to include a function, write it as a string.

You can convert it back into a function later:

Example

Convert a string into a function:

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a string into a function.</h2>

<p id="demo"></p>

<script>
var text = '{"name":"John", "age":"function() {return 30;}", "city":"New York"}';
var obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();
</script>

</body>
</html>
```

Convert a string into a function.

John, 30

• **JSON.stringify()**

A common use of **JSON** is to exchange data to/from a web server.

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with **JSON.stringify()**.

Stringify a JavaScript Object

JSON Programming Language

473

Imagine we have this object in JavaScript:

```
var obj = { name: "John", age: 30, city: "New York" };
```

Use the JavaScript function **JSON.stringify()** to convert it into a string.

```
var myJSON = JSON.stringify(obj);
```

myJSON is now a string, and ready to be sent to a server:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Create JSON string from a JavaScript object.</h2>

<p id="demo"></p>

<script>
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

Create JSON string from a JavaScript object.

```
{"name": "John", "age": 30, "city": "New York"}
```

Stringify a JavaScript Array

It is also possible to stringify JavaScript arrays:

Imagine we have this array in JavaScript:

```
var arr = [ "John", "Peter", "Sally", "Jane" ];
```

Use the JavaScript function **JSON.stringify()** to convert it into a string.

```
var myJSON = JSON.stringify(arr);
```

myJSON is now a string, and ready to be sent to a server:

Example

JSON Programming Language

474

```
<!DOCTYPE html>
<html>
<body>

<h2>Create JSON string from a JavaScript array.</h2>
<p id="demo"></p>

<script>

var arr = [ "John", "Peter", "Sally", "Jane" ];
var myJSON = JSON.stringify(arr);
document.getElementById("demo").innerHTML = myJSON;

</script>

</body>
</html>
```

Create JSON string from a JavaScript array.

```
["John","Peter","Sally","Jane"]
```

Exceptions

Stringify Dates

In **JSON**, date objects are not allowed. The **JSON.stringify()** function will convert any dates into strings.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON.stringify will convert any date objects into strings.</h2>
<p id="demo"></p>

<script>
var obj = { name: "John", today: new Date(), city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

JSON.stringify will convert any date objects into strings.

```
{"name": "John", "today": "2020-05-19T19:15:12.454Z", "city": "New York"}
```

Stringify Functions

In **JSON**, functions are not allowed as object values.

The **JSON.stringify()** function will remove any functions from a JavaScript object, both the key and the value:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON.stringify will remove any functions from an object.</h2>

<p id="demo"></p>

<script>
var obj = { name: "John", age: function () {return 30;}, city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

JSON.stringify will remove any functions from an object.

```
{"name": "John", "city": "New York"}
```

This can be omitted if you convert your functions into strings before running the **JSON.stringify()** function.

Example

JSON Programming Language

476

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON.stringify will remove any functions from an object.</h2>
<p>Convert the functions into strings to keep them in the JSON object.</p>

<p id="demo"></p>

<script>
var obj = { name: "John", age: function () {return 30;}, city: "New York" };
obj.age = obj.age.toString();
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

JSON.stringify will remove any functions from an object.

Convert the functions into strings to keep them in the JSON object.

```
{"name":"John","age":"function () {return 30;}","city":"New York"}
```

• JSON Objects

Object Syntax

Example

```
{ "name":"John", "age":30, "car":null }
```

JSON objects are surrounded by curly braces {}.

JSON objects are written in key/value pairs.

Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).

Keys and values are separated by a colon.

Each key/value pair is separated by a comma.

Accessing Object Values

You can access the object values by using **dot (.)** notation:

JSON Programming Language

477

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Access a JSON object using dot notation:</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = {"name":"John", "age":30, "car":null};
x = myObj.name;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Access a JSON object using dot notation:

John

You can also access the object values by using bracket (**[]**) notation:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Access a JSON object using bracket notation:</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = {"name":"John", "age":30, "car":null};
x = myObj["name"];
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Access a JSON object using bracket notation:

John

Looping an Object

You can loop through object properties by using the for-in loop:

JSON Programming Language

478

Example

```
<!DOCTYPE html>
<html>
<body>

<p>How to loop through all properties in a JSON object.</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = {"name":"John", "age":30, "car":null};
for (x in myObj) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
</script>

</body>
</html>
```

How to loop through all properties in a JSON object.

```
name
age
car
```

In a for-in loop, use the bracket notation to access the property values:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Use bracket notation to access the property values.</p>

<p id="demo"></p>

<script>
var myObj, x;
myObj = {"name":"John", "age":30, "car":null};
for (x in myObj) {
  document.getElementById("demo").innerHTML += myObj[x] + "<br>";
}
</script>

</body>
</html>
```

JSON Programming Language

479

Use bracket notation to access the property values.

```
John  
30  
null
```

Nested JSON Objects

Values in a JSON object can be another **JSON** object.

Example

```
myObj = {  
    "name": "John",  
    "age": 30,  
    "cars": {  
        "car1": "Ford",  
        "car2": "BMW",  
        "car3": "Fiat"  
    }  
}
```

You can access nested JSON objects by using the dot notation or bracket notation:

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p>How to access nested JSON objects.</p>  
  
<p id="demo"></p>  
  
<script>  
var myObj = {  
    "name": "John",  
    "age": 30,  
    "cars": {  
        "car1": "Ford",  
        "car2": "BMW",  
        "car3": "Fiat"  
    }  
}  
document.getElementById("demo").innerHTML += myObj.cars.car2 + "<br>";  
//or:  
document.getElementById("demo").innerHTML += myObj.cars["car2"];  
</script>  
  
</body>  
</html>
```

JSON Programming Language

480

How to access nested JSON objects.

BMW
BMW

Modify Values

You can use the dot notation to modify any value in a **JSON** object:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>How to modify values in a JSON object.</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
    "name":"John",
    "age":30,
    "cars": {
        "car1":"Ford",
        "car2":"BMW",
        "car3":"Fiat"
    }
}
myObj.cars.car2 = "Mercedes";

for (i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

How to modify values in a JSON object.

Ford
Mercedes
Fiat

You can also use the bracket notation to modify a value in a JSON object:

Example

JSON Programming Language

481

```
<!DOCTYPE html>
<html>
<body>

<p>How to modify values in a JSON object using the bracket notation.</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
    "name": "John",
    "age": 30,
    "cars": {
        "car1": "Ford",
        "car2": "BMW",
        "car3": "Fiat"
    }
}
myObj.cars["car2"] = "Mercedes";

for (i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

How to modify values in a JSON object using the bracket notation.

Ford
Mercedes
Fiat

Delete Object Properties

Use the delete keyword to delete properties from a **JSON** object:

Example

JSON Programming Language

482

```
<!DOCTYPE html>
<html>
<body>

<p>How to delete properties of a JSON object.</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
    "name": "John",
    "age": 30,
    "cars": {
        "car1": "Ford",
        "car2": "BMW",
        "car3": "Fiat"
    }
}
delete myObj.cars.car2;

for (i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

How to delete properties of a JSON object.

Ford
Fiat

• JSON Arrays

Arrays as JSON Objects

Example

```
[ "Ford", "BMW", "Fiat" ]
```

Arrays in JSON are almost the same as arrays in JavaScript.

In JSON, array values must be of type string, number, object, array, Boolean or null.

In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and undefined.

Arrays in JSON Objects

Arrays can be values of an object property:

Example

```
{  
  "name": "John",  
  "age": 30,  
  "cars": [ "Ford", "BMW", "Fiat" ]  
}
```

Accessing Array Values

You access the array values by using the index number:

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p>Access an array value of a JSON object.</p>  
  
<p id="demo"></p>  
  
<script>  
var myObj, x;  
myObj = {  
  "name": "John",  
  "age": 30,  
  "cars": [ "Ford", "BMW", "Fiat" ]  
};  
x = myObj.cars[0];  
document.getElementById("demo").innerHTML = x;  
</script>  
  
</body>  
</html>
```

Access an array value of a JSON object.

Ford

Looping Through an Array

You can access array values by using a for-in loop:

Example

JSON Programming Language

484

```
<!DOCTYPE html>
<html>
<body>

<p>Looping through an array using a for in loop:</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
  "name": "John",
  "age": 30,
  "cars": [ "Ford", "BMW", "Fiat" ]
};

for (i in myObj.cars) {
  x += myObj.cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Looping through an array using a for in loop:

Ford
BMW
Fiat

Or you can use a for loop:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Loopin through an array using a for loop:</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
  "name": "John",
  "age": 30,
  "cars": [ "Ford", "BMW", "Fiat" ]
};

for (i = 0; i < myObj.cars.length; i++) {
  x += myObj.cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JSON Programming Language

485

Loopin through an array using a for loop:

```
Ford  
BMW  
Fiat
```

Nested Arrays in JSON Objects

Values in an array can also be another array, or even another JSON object:

Example

```
myObj = {  
    "name": "John",  
    "age": 30,  
    "cars": [  
        { "name": "Ford", "models": [ "Fiesta", "Focus", "Mustang" ] },  
        { "name": "BMW", "models": [ "320", "X3", "X5" ] },  
        { "name": "Fiat", "models": [ "500", "Panda" ] }  
    ]  
}
```

To access arrays inside arrays, use a for-in loop for each array:

Example

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p>Looping through arrays inside arrays.</p>  
  
<p id="demo"></p>  
  
<script>  
var myObj, i, j, x = "";  
myObj = {  
    "name": "John",  
    "age": 30,  
    "cars": [  
        { "name": "Ford", "models": [ "Fiesta", "Focus", "Mustang" ] },  
        { "name": "BMW", "models": [ "320", "X3", "X5" ] },  
        { "name": "Fiat", "models": [ "500", "Panda" ] }  
    ]  
}  
for (i in myObj.cars) {  
    x += "<h2>" + myObj.cars[i].name + "</h2>";  
    for (j in myObj.cars[i].models) {  
        x += myObj.cars[i].models[j] + "<br>";  
    }  
}  
document.getElementById("demo").innerHTML = x;  
</script>  
</body>  
</html>
```

JSON Programming Language

486

Looping through arrays inside arrays.

Ford

Fiesta
Focus
Mustang

BMW

320
X3
X5

Fiat

500
Panda

Modify Array Values

Use the index number to modify an array:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>How to modify an array value of a JSON object.</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
    "name":"John",
    "age":30,
    "cars":[ "Ford", "BMW", "Fiat" ]
};
myObj.cars[1] = "Mercedes";

for (i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JSON Programming Language

487

How to modify an array value of a JSON object.

Ford
Mercedes
Fiat

Delete Array Items

Use the delete keyword to delete items from an array:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>How to delete properties of an array.</p>

<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
    "name":"John",
    "age":30,
    "cars": ["Ford","BMW","Fiat"]
}
delete myObj.cars[1];

for (i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

How to delete properties of an array.

Ford
Fiat

Reference

488

- 1- Arnold, Ken, et al. The Java programming language. Vol. 2. Reading: Addison-wesley, 2000.
- 2- Moreira, Jose E., et al. "Java programming for high-performance numerical computing." IBM Systems Journal 39.1 (2000): 21-56.
- 3- Mikkonen, Tommi, and Antero Taivalsaari. "Using JavaScript as a real programming language." (2007).
- 4- Severance, Charles. "Javascript: Designing a language in 10 days." Computer 45.2 (2012): 7-8.
- 5- De Volder, Kris. "JQuery: A generic code browser with a declarative configuration language." International Symposium on Practical Aspects of Declarative Languages. Springer, Berlin, Heidelberg, 2006.
- 6- Nixon, Robin. Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. " O'Reilly Media, Inc.", 2014.
- 7- Crockford, Douglas. "Json." ECMA International (2012).
- 8- Lin, Boci, et al. "Comparison between JSON and XML in Applications Based on AJAX." 2012 International Conference on Computer Science and Service System. IEEE, 2012.
- 9- Pezoa, Felipe, et al. "Foundations of JSON schema." Proceedings of the 25th International Conference on World Wide Web. 2016.
- 10- <https://www.w3schools.com/>