scripting

18 June 2023 20:22

cascading style sheets

What is CSS

CSS stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.
CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

What does CSS do

- You can add new looks to your old HTML documents.
- You can completely change the look of your website with only a few changes in CSS code.

Why use CSS

These are the three major benefits of CSS:

1) Solves a big problem

- Before CSS, tags like font, color, background style, element alignments, border and size had to be repeated on every web page.
- This was a very long process.
- For example: If you are developing a large website where fonts and color information are added on every single page, it will be become a long and expensive process. CSS was created to solve this problem. It was a W3C recommendation.

2) Saves a lot of time

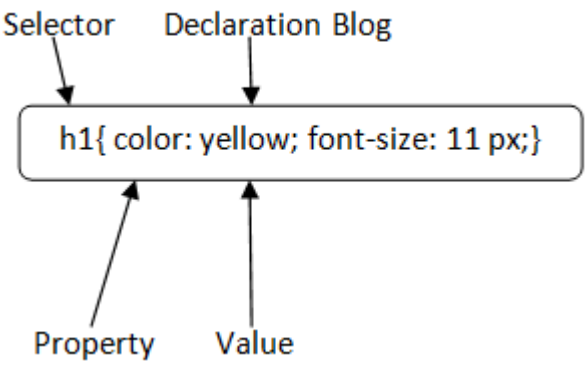
CSS style definitions are saved in external CSS files so it is possible to change the entire website by changing just one file.

3) Provide more attributes

CSS provides more detailed attributes than plain HTML to define the look and feel of the website.

CSS Syntax

A CSS rule set contains a selector and a declaration block.



Selector: Selector indicates the HTML element you want to style. It could be any tag like <h1>, <title> etc.

Declaration Block: The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

1. color: yellow;
2. font-size: 11 px;

Each declaration contains a property name and value, separated by a colon.

Property: A Property is a type of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

```
Selector{Property1: value1; Property2: value2; .....;}
```

CSS Selector

CSS selectors are used *to select the content you want to style*.
Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.
There are several different types of selectors in CSS.

a. CSS Element Selector p { }
b. CSS Id Selector #id { }
c. CSS Class Selector .class { }
d. CSS Universal Selector * { }
e. CSS Group Selector h1,h2,p { }

1) CSS Element Selector

The element selector selects the HTML element by name.

```
<style>
p{
    text-align: center;
    color: blue;
```

```
}
</style>
```

2) CSS Id Selector

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.
It is written with the hash character (#), followed by the id of the element.

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
    text-align: center;
    color: blue;
}
</style>
</head>
<body>
<p id="para1">Hello Javatpoint.com</p>
<p>This paragraph will not be affected.</p>
</body>
</html>
```

3) CSS Class Selector

The class selector selects HTML elements with a specific class attribute. It is used with a period character . (full stop symbol) followed by the class name.

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
    text-align: center;
    color: blue;
}
</style>
</head>
<body>
<h1 class="center">This heading is blue and center-aligned.</h1>
<p class="center">This paragraph is blue and center-aligned.</p>
</body>
</html>
```

4) CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
* {
  color: green;
  font-size: 20px;
}
</style>
</head>
<body>
<h2>This is heading</h2>
<p>This style will be applied on every paragraph.</p>
<p id="para1">Me too!</p>
<p>And me!</p>
</body>
</html>
```

5) CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.
Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

```
<!DOCTYPE html>
<html>
<head>
<style>
h1, h2, p {
    text-align: center;
    color: blue;
}
</style>
</head>
<body>
<h1>Hello Javatpoint.com</h1>
<h2>Hello Javatpoint.com (In smaller font)</h2>
<p>This is a paragraph.</p>
</body>
</html>
```

How to add CSS

CSS is added to HTML pages to format the document according to information in the style sheet. There are three ways to insert CSS in HTML documents.

1. Inline CSS
2. Internal CSS
3. External CSS

1) Inline CSS

Inline CSS is used to apply CSS on a single line or element.
For example:

```
<p style="color:blue">Hello CSS</p>
```

2) Internal CSS

Internal CSS is used to apply CSS on a single document or page. It can affect all the elements of the page. It is written inside the style tag within head section of html.
For example:

```
<style>
p{color:blue}
</style>
```

3) External CSS

- The external style sheet is generally used when you want to make changes on multiple pages.
- It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.
- It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

CSS Text Formatting Properties:

These are the following text formatting properties.

- Text Color: This property is used to set the color of the text and the color can be set by using a color name like “red”, hex value “#ff0000”, or by its RGB value “rgb(255,0,0)”;

- text-align: This property in CSS is used to specify the horizontal alignment of text in an element inside a block element or table-cell box.

- text-align-last: It is used to set the last line of the paragraph just before the line break. It sets the alignment of all the last lines occurring in the element in which the text-align-last property is applied.

- text-decoration: text-decoration property is used to “decorate ” the content of the text.

- text-decoration-color: It is used to set the color of the decorations (overlines, underlines, and line-throughs) over the text.

- text-decoration-line: It is used to set the various kinds of text decorations. this may include many values such as underline, overline, line-through, etc.

- text-decoration-style: This property is used to set the text-decoration of the element. It is the combination of the text-

decoration-line and text-decoration-color properties.

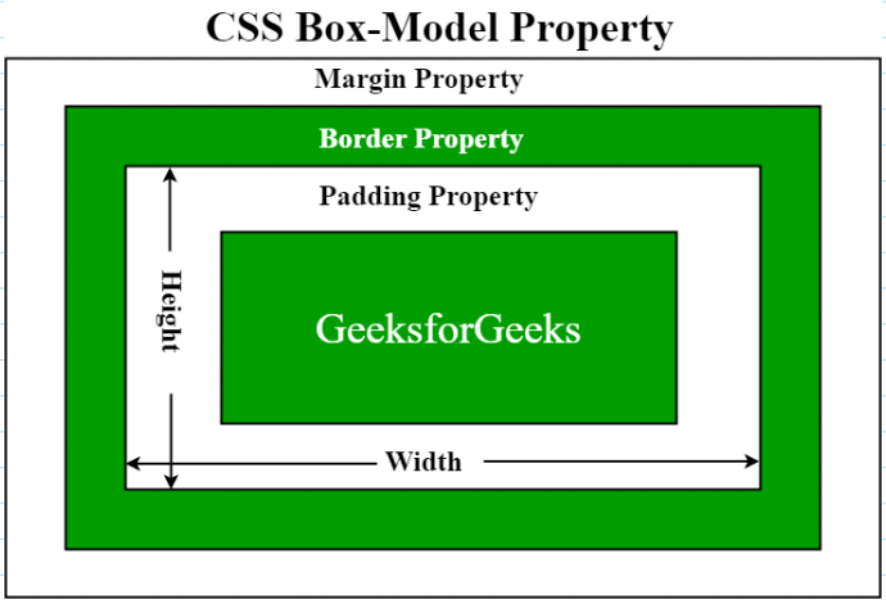
- text-indent: It is used to indent the first line of the paragraph and the size can be in px, cm, pt.
- text-justify: This property is used to set the text-align to justify. It spreads the words into complete lines.
- text-overflow: This property of text formatting specify that some text has overflown and is hidden from the view.
- text-transform: It is used to control the capitalization of the text.
- text-shadow: it is used to add shadow to the text.
- letter-spacing: This property is used to specify the space between the characters of the text.
- line-height: It is used to set the space between the lines.
- direction: This property is used to set the direction of the text.
- word-spacing: It is used to specify the space between the words of the line.

CSS Box model

The **CSS box model** is a container that contains multiple properties including borders, margins, padding, and the content itself. It is used to create the design and layout of web pages. It can be used as a toolkit for customizing the layout of different elements. The web browser renders every element as a rectangular box according to the CSS box model.

Box-Model has multiple properties in CSS. Some of them are given below:

- **content**: This contains the actual data in the form of text, images, or other media forms and it can be sized using the width & height property.
- **padding**: This property is used to create space around the element, inside any defined border.
- **border**: This property is used to cover the content & any padding, & also allows setting the style, color, and width of the border.
- **margin**: This property is used to create space around the element ie., around the border area.
The following figure illustrates the **Box model** in CSS.



- **Content Area:** This area consists of content like text, images, or other media content. It is bounded by the content edge and its dimensions are given by content-box width and height.
- **Padding Area:** It includes the element's padding. This area is actually the space around the content area and within the border-box. Its dimensions are given by the width of the padding-box and the height of the padding-box.
- **Border Area:** It is the area between the box's padding and margin. Its dimensions are given by the width and height of the border.
- **Margin Area:** This area consists of space between the border and the margin. The dimensions of the Margin area are the margin-box width and the margin-box height. It is useful to separate the element from its neighbors

The position Property

The `position` property specifies the type of positioning method used for an element.
There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`

position: static;

HTML elements are positioned static by default.
Static positioned elements are not affected by the top, bottom, left, and right properties.

```
div.static {
  position: static;
  border: 3px solid #73AD21;
```

```
}
```

position: relative;

An element with `position: relative;` is positioned relative to its normal position.
Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

```
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
```

position: fixed;

An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.
A fixed element does not leave a gap in the page where it would normally have been located.

```
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
```

position: absolute;

An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

```
div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
```

position: sticky;

An element with `position: sticky;` is positioned based on the user's scroll position.
A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

```
div.sticky {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
  background-color: green;
  border: 2px solid #4CAF50;
}
```

CSS Colors

The color property in CSS is used to set the color of HTML elements. Typically, this property is used to set the background color or the font color of an element.
In CSS, we use color values for specifying the color. We can also use this property for the border-color and other decorative effects.
We can define the color of an element by using the following ways:

- RGB format.
- RGBA format.
- Hexadecimal notation.
- HSL.
- HSLA.
- Built-in color.

```
<style>
        h1{
            text-align:center;
        }
        #rgb{
          color:rgb(255,0,0);
        }
        #rgba{
          color:rgba(255,0,0,0.5);
        }
        #hex{
          color:#EE82EE;
        }
       #hsl{
          color:hsl(0,50%,50%);
        }
```

```
        #hsla{
          color:hsla(0,50%,50%,0.5);
        }
        #built{
          color:green;
        }
    </style>
```

CSS Pseudo-classes

A Pseudo class in CSS is used to define the special state of an element. It can be combined with a CSS selector to add an effect to existing elements based on their states. For Example, changing the style of an element when the user hovers over it, or when a link is visited. All of these can be done using Pseudo Classes in CSS.

Syntax:

```
selector: pseudo-class{
    property: value;
}
```

There are many Pseudo-classes in CSS but the ones that are most commonly used are as follows:

- **:hover Pseudo-class:** This pseudo-class is used to add a special effect to an element when our mouse pointer is over it.

The below example demonstrates that when your mouse enters the box area, its background color changes from yellow to orange.

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS transition-property property</title>
    <style>
        .box {
            background-color: yellow;
            width: 300px;
            height: 200px;
            margin: auto;
            font-size: 40px;
            text-align: center;
        }

        .box:hover {
            background-color: orange;
        }

        h1,
        h2 {
            color: green;
            text-align: center;
        }
    </style>
</head>

<body>
    <h1>Geeks For Geeks</h1>
```

```
    <h2>:hover Pseudo-class</h2>
    <div class="box">
        My color changes if you hover over me!
    </div>
</body>
</html
```

- **:active Pseudo-class:** This pseudo-class is used to select an element that is activated when the user clicks on it.

  The following example demonstrates that when you click on the box, its background color changes for a moment.

```
.box:active{
        background-color: orange;
    }
```

- **:focus Pseudo-class:** This pseudo-class is used to select an element that is currently focused by the user. It works on user input elements used in forms and is triggered as soon as the user clicks on it.

  In the following example, the background color of the input field which is currently focused changes.

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS transition-property property</title>
    <style>
        form{
            width: 300px;
            height: 200px;
            margin: 0 auto;
            text-align: center;
            line-height: 2rem;
        }

        label{
            width: 30%;
        }

        input{
            background-color: default;
            float: right;
        }

        input:focus{
            background-color: grey;
        }

        h1, h2{
            color: green;
            text-align: center;
        }
    </style>
</head>

<body>
    <h1>Geeks For Geeks</h1>
    <h2>:focus Pseudo-class</h2>
    <form>
```

```
            <label for="username">Username:</label>
            <input type="text" name="username"
                   placeholder="Enter your username" />
            <br>

            <label for="emailid">Email-Id:</label>
            <input type="email" name="emailid"
                   placeholder="Enter your email-id" />

            <label for="Password">Password:</label>
            <input type="password" name="Password"
                   placeholder="Enter your password" />
        </form>
    </body>
    </html>
```

- **:visited Pseudo-class:** This pseudo-class is used to select the links which have been already visited by the user.
  In the following example, the color of the link changes once it is visited.

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS transition-property property</title>
    <style>
        body{
            text-align: center;
        }

        h1, h2{
            color: green;
        }

        a:visited{
            color: red;
        }
    </style>
</head>

<body>
    <h1>Geeks For Geeks</h1>
    <h2>:visited Pseudo-class</h2>

    <p>
        <a href=" https://www.geeksforgeeks.org/"
target="_blank">
            My color changes once you visit this link
        </a>
    </p>
</body>
</html>
```

What is JavaScript ?

- JavaScript is a dynamic computer programming language.
- It is lightweight and most commonly used as a part of web pages,
- whose implementations allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented

capabilities

HISTORY

- JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java.

- JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**.
- The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

  The ECMA-262 Specification defined a standard version of the core JavaScript language.
  - JavaScript is a lightweight, interpreted programming language.
  - Designed for creating network-centric applications.
  - Complementary to and integrated with Java.
  - Complementary to and integrated with HTML.
  - Open and cross-platform

Client-Side JavaScript

- Client-side JavaScript is the most common form of the language.
- The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
- The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts.
- For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.
- The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.
- JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are –

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features –

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

JavaScript Identifiers

- javaScript identifiers are the name that we give to variables, objects, functions, arrays, classes, etc.
- We must use a unique name so as to identify them.
- We then use the identifier to refer to the variable, functions, etc elsewhere in the program.
- There are certain rules & restrictions that we must follow when we name an identifier.

In the example below, the message is the name we have given to the variable. The variable holds the string “hello world”. Hence the message is Identifier.

```
var message;
message=”hello world”
```

JavaScript Identifiers Rules

When you name any identifier, you need to follow these rules.

1. The identifier name must be unique within the scope.
2. The first letter of an identifier should be a
   - upper case letter
   - Lower case letter
   - underscore
   - dollar sign
3. The subsequent letters of an identifier can have
   - upper case letter
   - Lower case letter
   - underscore
   - dollar sign
   - numeric digit

4. We cannot use reserved keywords. You can find the list of keywords for here.
5. They are case-sensitive. For Example, `sayHello` is different from `SayHello`
6. Cannot use spaces in a identifier name.
7. Avoid using the JavaScript Reserved Keywords as identifier name

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

JavaScript Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

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

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

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
<<	left shift	5 << 1	0101 << 1	1010	10

>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)
{
    code to be executed
}
```

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

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

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

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

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.
There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

```
var arrayname=[value1,value2.....valueN];
```

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
var arrayname=new Array();
```

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.
The example of creating object by array constructor is given below.

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

JavaScript Function

JavaScript function is a set of statements that are used to perform a specific task. It can **take one or more input** and **can return output** as well. Both, taking input and returning an output are optional.
Using functions avoids repetition of the code as we need to write the code only once and then the code can be called anywhere using function name. A function can be helpful in the following scenarios.

- For instance, suppose you want to add some numbers and display the results on a web page. In that case, you can define the code for adding the numbers in a function and call the function whenever needed.
- For repetitive tasks like displaying a message whenever a web page is loaded into the browser, we can use functions.

functions can be **parameterized** and **non-parameterized** which means they may or may not take any input. A function that does not take any parameters/inputs is known as **non-parametrized function**.

we can categorize functions into two category:

1. User defined Function
2. Built-in Function

Creating a User-defined Function:

In JavaScript, function is defined by using the function keyword followed by the **function name** and parentheses ( ), to hold params(inputs) if any. A function can have **zero or more parameters separated by commas**.
The **function body is enclosed within curly braces** just **after the function declaration part**(function name and params), and at the end of the function body, we can have a return statment to return some output, if we want.
Following is the syntax for JavaScript user-defined functions:

```
functionfunction_name(parameter-1,parameter-2,...parameter-n)
{
        // function body
}
```

Calling a User-defined Function:

After creating a function, we can call it anywhere in our script. The syntax for calling a function is given below:

```
function_name(val-1,val-2,...,val-n);
```

JavaScript Built-In Functions

Functions that are provided by JavaScript itself as part of the scripting language, are known as **built-in functions**. JavaScript provides a rich set of the library that has a lot of built-in functions. Some examples of the built-in functions
are : alert(), prompt(), parseInt(), eval() etc.

JavaScript Errors

Throw, and Try...Catch...Finally

- The `try` statement defines a code block to run (to try).
- The `catch` statement defines a code block to handle any error.
- The `finally` statement defines a code block to run regardless of the result.
- The `throw` statement defines a custom error.

```
try {
 Block of code to try
}
catch(err) {
 Block of code to handle errors
}
```

JavaScript Throws Errors

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

```
throw "Too big";    // throw a text

throw 500;          // throw a number
```

If you use `throw` together with `try` and `catch`, you can control program flow and generate custom error messages.

The finally Statement

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

Syntax

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

The Error Object

JavaScript has a built in error object that provides error information when an error occurs.
The error object provides two useful properties: name and message.

Error Object Properties

Property	Description
name	Sets or returns an error name
message	Sets or returns an error message (a string)

Error Name Values

Six different values can be returned by the error name property:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred