Table of Content

Click or tap to jump to any section;

`<head> ... </head>`

Used to specify metadata about the webpage. It includes the webpage's name, its dependencies (JS and CSS scripts), font usage, etc.

`<body> ... </body>`

Everything the user sees on a webpage is written inside this tag. It's a container for all the contents of the webpage.

Example

```
<html>
        <head>
                <title>My First Website</title>
        </head>
        <body>

        </body>
</html>
```

Document Information

`<base>`

Used to specify the base URL of your site. This tag makes linking to internal links on your site systematized. Remember that this tag can only be used once and only in the **<head>** tag.

`<meta>`

Metadata tag for the webpage. It can help highlight the page's author, keywords, original published date, etc.

`<style> ... </style>`

It can be used as an alternative to an external style sheet or complement it. Includes the webpage's appearance information.

`<script> ... </script>`

Used to add code snippets, typically in JavaScript, to make a web page dynamic. It can also be used to just link to an external script.

`<link>`

Used to link to scripts external to the webpage. Typically utilized for including stylesheets.

Example

```
<html>
        <head>
        <meta charset="utf-8">
        <base href="http://myfirstwebsite.tld" target="_blank" />
        <title>My Beautiful Website</title>
        <link rel="stylesheet" href="/css/master.css">
        <script type="text/javascript">
                var dummy = 0;
        </script>
        </head>
        <body>

        </body>
</html>
```

Document Structure

`<h1-h6> ... </h1-h6>`

Six different variations of writing a heading. **<h1>** has the largest font size, while **<h6>** has the smallest.

`<br>`

A single line break for webpages. It's used to write a new line.

`<hr>`

Similar to the **
** tag. In addition to switching to the next line, **<hr>** tag also draws a horizontal bar to indicate the end of the section.

`<span> ... </span>`

This tag injects inline elements, like an image, icon, or emoticon, without ruining the formatting or styling of the page.

`<div> ... </div>`

A web page's content is usually divided into blocks specified by the ** ** tag.

`<p> ... </p>`

Plain text is placed inside this tag.

Example

```
<div>
        <h1>Top 10 Greatest Films</h1>
        <p>These are considered the greatest movies of all time.</p>
        <hr>
                <h2>The Godfather</h2>
                <p>This 1972 classic stars Marlon Brando and Al Pacino.</p>
</div>
```

Text Formatting

`<strong> ... </strong>`

Makes text **bold**. Used to emphasize a word or phrase.

`<b> ... </b>`

Alternative to the **** tag. Generates **bold** text.

`<em> ... </em>`

Another emphasis tag, but displays text in *italics*.

`<i> ... </i>`

Used to display text in italics but does not emphasize it like the **** tag.

`<cite> ... </cite>`

A tag for citing the author of a quote.

`<ins> ... </ins>`

Denotes text that has been added to the webpage.

`<blockquote> ... </blockquote>`

Quotes often go into this tag. Is used in tandem with the **<cite>** tag.

`<pre> ... </pre>`

Pre-formatted, monospaced font text laid out, with whitespace inside the element, remained intact.

`<q> ... </q>`

Similar to the above tag, but for shorter quotes.

`<address> ... </address>`

A tag for specifying the author's contact details.

`<abbr> ... </abbr>`

Denotes abbreviations or acronyms.

`<dfn> ... </dfn>`

A tag dedicated to mark definitions.

`<code> ... </code>`

Used to display code snippets within a paragraph.

`<sup> ... </sup>`

Similar to the <sub> tag, but used to write a superscript. **Example**: a^x.

`<sub> ... </sub>`

Used to write a subscript. It's smaller font just below the mid-point of regular fonts. **Example**: a_x.

`<small> ... </small>`

Reduces text size. In HTML5, it often refers to redundant or invalid information.

Example

```
<p>Here's <strong>some bold text</strong> and here's <em>some text in italics</em> compared to regular text.</p>

<blockquote>
Anyone who has never made a mistake has never tried anything new.
<cite>-  Albert Einstein</cite>
</blockquote>

<pre>
Here's what pre-formatted text looks like.
</pre>

<p>A code snippet example: <code>Three. Two. Online</code></p>
```

Links

`<a href="mailto:"> ... </a>`

A tag dedicated to sending emails.

`<a href="#name"> ... </a>`

A variation of the **<a name="name">** tag. Used to navigate to the web page's ** ** section only.

`<a href=""> ... </a>`

An anchor tag. Primarily used to include hyperlinks.

`<a href="tel://###-###"> ... </a>`

An anchor tag for mentioning contact numbers. The numbers are clickable, which can be particularly beneficial for mobile users.

Images

`<img>`

A tag to display images on a web page.

`alt="text"`

A text is displayed when the user hovers the mouse over an image. It can be used to give additional details about the image.

`border=""`

Specifies the border thickness of the image. If not mentioned, the default value is 0.

`width=""`

Specifies image width in pixels or percentages.

`shape=""`

The shape of an area.

`<map name=""> ... </map>`

Name of the map associated between an image and a map.

`<area>`

Specifies image map area.

`coords=""`

Coordinates vital information about the shape. **Example:** vertices for rectangles, center or radius for circles.

`src="url"`

A URL or path where the image is located on your drive or the web.

`height=""`

Specifies image height in pixels or percentages.

`<map> ... </map>`

Denotes an interactive (clickable) image.

`align=""`

The relative alignment of an image. It can change parallel to other elements on a web page.

Example

```
<img src="https://upload.wikimedia.org/wikipedia/commons/8/83/Solar_system.jpg"
width="823" height="1024" alt="The Solar System"
usemap="#solarmap">

<map name="solarmap">

<area shape="circle" coords="572,322,100" href="solar-system.htm" alt="The Solar
System">

</map>
```

Lists

`<ol> ... </ol>`

A tag for an ordered or numbered list of items.

`<li> ... </li>`

An individual item as part of a list.

`<ul> ... </ul>`

Opposed to the **<ul>** tag. Used for an unordered list of items.

`<dt> ... </dt>`

A definition of a single term that is in-line with body content.

`<dl> ... </dl>`

A tag for a list of items with definitions.

`<dd> ... </dd>`

A description of the defined term.

Example

```
<ol>
        <li>Monday</li>
        <li>Tuesday</li>
        <li>Wednesday</li>
</ol>
<ul>
        <li>France</li>
        <li>Germany</li>
        <li>Italy</li>
</ul>
<dl>
<dt>Toyota</dt>
        <dd>A Japanese car brand</dd>
        <dt>Armani</dt>
        <dd>An Italian fashion brand</dd>
</dl>
```

Forms

`<form> ... </form>`

A parent tag for an HTML form.

`action="url"`

Where the form data will be submitted once the user fills it.

`method=""`

Specifies which HTTP method (POST or GET) will be used to submit the form.

`autocomplete`

Determines if the form has auto-complete enabled.

`accept-charsets`

Determines character encodings when a form is submitted.

`<fieldset> ... </fieldset>`

Identifies groups of all fields on the form.

`<input>`

Used to take input from the user. The input type is determined by a number of attributes.

`enctype=""`

Used only for the POST method. It dictates the data encoding scheme when a form is submitted.

`novalidate`

Determines whether a form should be validated before submission.

`<label> ... </label>`

Used to label a field in the form.

`<legend> ... </legend>`

Operates as a caption for the <fieldset> element.

`target`

After submission, the form response will be displayed wherever this refers to. Usually has the following values: _blank, _self, _parent, _top.

Input Type Attribute

`type=""`

Determines which type of input (text, dates, password) is requested from the user.

`name=""`

Specifies the name of the input field.

`value=""`

Specifies the value currently contained in the input field.

`width=""`

Determines the width of the input element in pixel values.

`size=""`

Determines the input element width (number of characters).

`maxlength=""`

Specifies the most input field characters allowed.

`required`

Makes an input field compulsory. The form cannot be submitted if a required field is left empty.

`height=""`

Determines the height of the input element in pixel values.

placeholder=""

This tag can be used to provide a hint to the user about the nature of the requested data.

pattern=""

Specifies a regular expression that can be used to look for patterns in the user's text.

min=""

The minimum value allowed for an **<input>** element.

max=""

The maximum value allowed for an **<input>** element.

autofocus

Forces focus on the input element when a web page loads completely.

disabled

Disables an input element. The user can no longer enter data.

<textarea> ... </textarea>

Applied for longer strings of input. It can be used to get a multi-sentence text from the user.

<select> ... </select>

Specifies a list of options that the user can choose from.

Select Attributes

name=""

A name for a particular list of options.

size=""

A total number of options given to the user.

multiple

States if the user can choose multiple options from the list.

required

Specifies if choosing an option(s) is necessary for form submission.

autofocus

Specifies that a drop-down list will automatically come into focus after the page loads.

<option> ... </option>

A tag for listing individual items on the list of options.

Option Attributes

value=""

A text visible to the user for any given option.

selected

Determines which option is selected by default when a form loads.

`<button> ... </button>`

Tag for creating a button for form submission.

Example

```
<form action="form_submit.php" method="post">
        <fieldset>
                <legend>Bio:</legend>
                First name:<br>
                <input type="text" name="first-name" value="John"
                placeholder="Please enter your first name here"><br>
                Last name:<br>
                <input type="text" name="last-name" value="Doe"
                placeholder="Please enter your last name here"><br><br>
                Favorite sport:<br>
                <select>
                        <option value="basketball">Basketball</option>
                        <option value="soccer">Soccer</option>
                        <option value="tennis">Tennis</option>
                </select>
                <textarea name="description"></textarea>
                <input type="submit" value="Submit">
        </fieldset>
</form>
```

Tables

`<table> ... </table>`

Marks a table on a webpage.

`<caption> ... </caption>`

A description of a table is placed inside this tag.

`<tfoot> ... </tfoot>`

Determines the footer of a table.

`<tbody> ... </tbody>`

The body of a table where the data is held.

`<tr> ... </tr>`

Denotes a single row in a table.

`<thead> ... </thead>`

Specifies information about specific columns of a table.

`<th> ... </th>`

The value of a heading of a table's column.

`<td> ... </td>`

A single cell of a table. Contains the actual value/data.

`<colgroup> ... </colgroup>`

Used for grouping columns together.

`<col>`

Denotes a column inside a table.

Example

```
<table>
        <colgroup>
                <col span="2">
                <col>
        </colgroup>
        <tr>
                <th>Name</th>
                <th>Major</th>
                <th>GPA</th>
        </tr>
        <tr>
                <td>Bob</td>
                <td>Law</td>
                <td>3.55</td>
        </tr>
        <tr>
                <td>Alice</td>
                <td>Medicine</td>
                <td>3.61</td>
        </tr>
</table>
```

Objects and iFrames

`<object> ... </object>`

Used to embed an additional multimedia object into a web page. This can be an audio/video file, document (.pdf), etc.

`<iframe> ... </iframe>`

An inline block of content. It's used as a container for multimedia objects in a flexible manner. It floats inside a web page, meaning it's placed relative to other webpage items.

`height=""`

Determines object height in pixel values.

`width=""`

Determines object width in pixel values.

`type=""`

The type/format of the object's contents.

iFrame Attributes

`name=""`

The name of an iFrame.

`srcdoc=""`

Any HTML content to be displayed inside an iFrame.

`src=""`

The source URL/path of the multimedia object to be held inside an iFrame.

`<param />`

Used for iFrame customization. This includes additional parameters to go along with the content.

`height=""`

Determines the height of an iFrame.

`width=""`

Determines the width of an iFrame.

`<embed> ... </embed>`

Used to embed external objects, such as plugins.

Embed Attributes

`height=""`

Determines the height of an embedded item.

`width=""`

Determines the width of an embedded item.

`src=""`

The URL/path of an embedded item.

`type=""`

The type or format of an embedded content.

Example

```
<object width="1000" height="1000"></object>
<iframe src="other-web-page.html" width="500" height="500"></iframe>
<embed src="video-file.swf" width="500" height="500"></embed>
```

HTML5 New Tags

`<header> ... </header>`

Specifies the webpage header. It can also be used for objects inside the web page.

`<footer> ... </footer>`

Specifies the webpage footer. It can also be used for objects inside the web page.

`<main> ... </main>`

Marks the main content of a web page.

`<article> ... </article>`

Denotes an article.

`<aside> ... </aside>`

Denotes content displayed in a sidebar of a web page.

`<details> ... </details>`

Used for additional information. The user has the option to view or hide these details.

`<figure> ... </figure>`

A tag reserved for figures (diagrams, charts) in HTML5.

`<summary> ... </summary>`

Used as a heading for the **<details>** tag. It's always visible to the user.

`<dialog> ... </dialog>`

Used to create a dialog box.

`<section> ... </section>`

Specifies a particular section in a webpage.

`<figcaption> ... </figcaption>`

A description of the figure is placed inside.

`<mark> ... </mark>`

Used to highlight a particular portion of a text.

`<nav> ... </nav>`

Navigation links for the user in a web page.

`<menuitem> ... </menuitem>`

A particular item from a list or menu.

`<meter> ... </meter>`

Measures data within a given range.

`<progress> ... </progress>`

Typically used as a progress bar. This is used to track progress.

`<rp> ... </rp>`

Shows text for web browsers without **Ruby annotation** support.

`<ruby> ... </ruby>`

Describes a **Ruby annotation** for East Asian typography.

`<rt> ... </rt>`

Displays East Asian typography character details.

`<time> ... </time>`

A tag for formatting date and time.

`<wbr>`

A line-break within the content.

Collective Character Objects

`&#34; &quot;`

Quotation Marks (**"**)

`&#38; &amp;`

Ampersand (**&**)

`&#60; &lt;`

Less than sign (**<**)

`&#62; &gt;`

Greater than sign (**>**)

`   `

Non-breaking space ( )

`&#169; &copy;`

Copyright symbol (©)

`&#64; &Uuml;`

@ Symbol (**@**)

`&#149; &ouml;`

Small bullet (•)

`&#153; &ucirc;`

Trademark symbol (™)

LIST OF CONTENTS

Hostinger Tutorials

BACKGROUND

background	*background-image* *background-position* *background-size* *background-repeat* *background-attachment* *background-origin* *background-clip* *background-color*
background-attachment	scroll | fixed
background-break	bounding-box | each-box | continuous
background-clip	*length* *%* border-box | padding-box | content-box | no-clip
background-color	*color* transparent
background-image	*url* none
background-origin	border-box | padding-box | content-box
background-position	top left | top center | top right | center left | center center | center right | bottom left | bottom center | bottom right *x-% y-%* *x-pos y-pos*
background-repeat	repeat | repeat-x | repeaty | no-repeat
background-size	*length* *%* auto | cover | contain

TABLE

border-collapse	collapse | separate
border-spacing	*length length*
caption-side	top | bottom | left | right
empty-cells	show | hide
table-layout	table-layout auto | fixed

BORDER

border	*border-width* *border-style* *border-color*
border-break	*border-width* *border-style* *color* close
border-bottom	*border-bottom-width* *border-style* *border-color*
border-bottom-color	*border-color*
border-bottom-style	*border-style*
border-bottom-width	thin | medium | thick *length*
border-collapse	collapse | separate
border-color	*color*
border-image	*image* [*number / %* *border-width* stretch | repeat | round] none
border-left	*border-left-width* *border-style* *border-color*
border-left-color	*border-color*
border-left-style	*border-style*
border-left-width	thin | medium | thick *length*
border-right	*border-right-width* *border-style* *border-color*
border-right-color	*border-color*
border-right-style	*border-style*
border-right-width	thin | medium | thick *length*

BORDER

border-top	*border-top-width* *border-style* *border-color*
border-top-color	*border-color*
border-top-style	*border-style*
border-top-width	thin | medium | thick *length*
border-width	thin | medium | thick *length*
border-radius	*border-top-right-radius* *border-bottom-right-radius* *border-bottom-left-radius* *border-top-left-radius*
border-top-right-radius	*length*
border-bottom-right-radius	*length*
border-bottom-left-radius	*length*
border-top-left-radius	*length*
box-shadow	inset || [*length, length, length, length* || <*color*>] none
border-style	none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset

TRANSITIONS

transition	transition-property transition-duration transition-timing-function transition-delay
transition-delay	time
transition-duration	time
transition-property	none | all
transition-timing-function	ease | linear | ease-in | ease-out | ease-in-out | cubic-Bezier (number, number, number)

BOX MODEL

clear	left | right | both | none
display	none | inline | block | inlineblock | list-item | run-in | compact | table | inlinetable | table-row-group | table-header-group | tablefooter-group | table-row | table-column-group | tablecolumn | table-cell | tablecaption | ruby | ruby-base | ruby-text | ruby-base-group | ruby-text-group
float	left | right | none
height	auto *length* *%*
max-height	none *length* *%*
max-width	none *length* *%*
min-height	none | inherit *length* *%*
min-width	none | inherit *length* *%*
width	auto *%* *length*
margin	*margin margin-top* *margin-right* *margin-bottom* *margin-left*
margin-bottom	auto *length* *%*
margin-left	auto *length* *%*
margin-right	auto *length* *%*

BOX MODEL

margin-top	auto *length* *%*
padding	*padding padding-top* *padding-right* *padding-bottom* *padding-left*
padding-bottom	*length* *%*
padding-left	*length* *%*
padding-right	*length* *%*
padding-top	*length* *%*
marquee-direction	forward | reverse
marquee-loop	infinite *number*
marquee-play-count	infinite *integer*
marquee-speed	slow | normal | fast
marquee-style	scroll | slide | alternate
overflow	visible | hidden | scroll | auto | no-display | nocontent *overflow-x* *overflow-y*
overflow-style	auto | marquee-line | marquee-block
overflow-x	visible | hidden | scroll | auto | no-display | nocontent
overflow-y	visible | hidden | scroll | auto | no-display | nocontent
rotation	*angle*
rotation-point	*position (paired value offset)*
visibility	visibility visible | hidden | collapse

FONT

font	*font-style* *font-variant* *font-weight* *font-size/line-height* *font-family* caption | icon | menu | message-box | smallcaption | status-bar
font-family	*family-name* *generic-family* inherit
font-size	xx-small | x-small | small | medium | large | x-large | xx-large | smaller | larger | inherit *length* *%*
font-size-adjust	none| inherit *number*
font-stretch	normal | wider | narrower | ultra-condensed | extracondensed | condensed | semi-condensed | semiexpanded | expanded | extra-expanded | ultraexpanded | inherit
font-style	normal | italic | oblique | inherit
font-variant	normal | small-caps | inherit
font-weight	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit

COLOR

color	inherit *color*
opacity	inherit *number*

TEXT

direction	ltr | rtl | inherit
hanging-punctuation	none | [start | end | endedge]
letter-spacing	normal *length* *%*
punctuation-trim	none | [start | end | adjacent]
text-align	start | end | left | right | center | justify
text-align-last	start | end | left | right | center | justify
text-decoration	none | underline | overline | line-through | blink
text-emphasis	none | [[accent | dot | circle | disc] [before | after]?]
text-indent	*length* *%*
text-justify	auto | inter-word | interideograph | inter-cluster | distribute | kashida | tibetan
text-outline	none *color* *length*
text-shadow	none *color* *length*
text-transform	none | capitalize | uppercase | lowercase
text-wrap	normal | unrestricted | none | suppress
unicode-bidi	normal | embed | bidioverride
white-space	normal | pre | nowrap | prewrap | pre-line
white-space-collapse	preserve | collapse | preserve-breaks | discard
word-break	normal | keep-all | loose | break-strict | break-all
word-spacing	normal *length* *%*
word-wrap	normal | break-word

COLUMN

column-count	auto *number*
column-fill	auto | balance
column-gap	normal *length*
column-rule	*column-rule-width* *column-rule-style* *column-rule-color*
column-rule-color	*color*
column-rule-style	*border-style*
column-rule-width	thin | medium | thick *length*
columns	*column-width* *column-count*
column-span	1 | all
column-width	auto *length*

ANIMATIONS

animation	animation-name animation-duration animation-timing-function animation-delay animation-iteration-count animation-direction
animation-delay	time
animation-direction	normal | alternate
animation-duration	time
animation-iteration-count	inherit number
animation-name	none | IDENT
animation-play-state	running | paused
animation-timing-function	ease | linear | ease-in | ease-out | ease-in-out | cubic-Bezier (number, number, number, number)

SPEECH

cue	*cue-before* *cue-after*
cue-before	*uri* [silent | x-soft | soft | medium | loud | x-loud] | none | inherit] *number* *%*
cue-after	*uri* [silent | x-soft | soft | medium | loud | x-loud] | none | inherit] *number* *%*
mark	*mark-before* *mark-after*
mark-before	*string*
mark-after	*string*
pause	*pause-before* *pause-after*
pause-before	none | x-weak | weak | medium | strong | x-strong | inherit *time*
pause-after	none | x-weak | weak | medium | strong | x-strong | inherit *time*
phonemes string	*string*
rest	rest-before rest-after
rest-before	none | x-weak | weak | medium | strong | x-strong | inherit *time*
rest-after	none | x-weak | weak | medium | strong | x-strong | inherit *time*
speak	none | normal | spell-out | digits | literal-punctuation | no-punctuation | inherit
voice-balance	left | center | right | leftwards | rightwards | inherit *number*
voice-duration	*time*

SPEECH

voice-family	inherit | [<specific-voice, age, generic-voice, number>]
voice-rate	x-slow | slow | medium | fast | x-fast | inherit *%*
voice-pitch	x-low | low | medium | high | x-high | inherit *number* *%*
voice-pitch-range	x-low | low | medium | high | x-high | inherit *number*
voice-stress	strong | moderate | none | reduced | inherit
voice-volume	silent | x-soft | soft | medium | loud | x-loud | inherit *number* *%*

TEMPLATE LAYOUT

box-align	start | end | center | base
box-direction	normal | reverse
box-flex	*number*
box-flex-group	*integer*
box-lines	single | multiple
box-orient	horizontal | vertical | inlineaxis | block-axis
box-pack	start | end | center | justify
box-sizing	content-box | padding-box | border-box | margin-box
tab-side	top | bottom | left | right

LIST & MARKERS

list-style	*list-style-type* *list-style-position* *list-style-image*
list-style-image	none *url*
list-style-position	Inside | outside
list-style-type	none | asterisks | box | check | circle | diamond | disc | hyphen | square | decimal | decimal-leadingzero | lower-roman | upperroman | lower-alpha | upper-alpha | lower-greek | lower-latin | upper-latin | hebrew | armenian | georgian | cjk-ideographic | hiragana | katakana | hiragana-iroha | katakana-iroha | footnotes
marker-offset	auto *length*

GRID POSITIONING

grid-columns	none | inherit *[length percentage relative length]*
grid-rows	none | inherit *[length percentage relative length]*

OUTLINE

outline	*outline-color* *outline-style* *outline-width*
outline-color	*color* invert
outline-offset	inherit *length*
outline-style	None | dotted | dashed | solid | double | groove | ridge | inset | outset
outline-width	thin | medium | thick *length*

3D / 2D TRANSFORM

backface-visibility	visible | hidden
perspective	none *number*
perspective-origin	*[[[percentage> | <length> | left | center | right] [<percentage> | <length> | top | center | bottom]?] <length>] | [[[left | center | right] | | [top | center | bottom]] <length>]*
transform	none | matrix | matrix3d | translate3d | translateX | translateY | translateZ | scale | scale3d | scaleX | scaleY | scaleZ | rotate | rotate3d | rotateX | rotateY | rotateZ | skewX | skewY | skew | perspective
transform-origin	*[[[<percentage> | <length> | left | center | right] [<percentage> | <length> | top | center | bottom]?] <length>] | [[[left | center | right] | | [top | center | bottom]] <length>]*
transform-style	flat | preserve-3d

HYPERLINK

target	*target-name* *target-new* *target-position*
target-name	current | root | parent | new | modal *string*
target-new	window | tab | none
target-position	above | behind | front | back

GENERATED CONTENT

bookmark-label	content *attr* *string*
bookmark-level	none *integer*
bookmark-target	self *uri* *attr*
border-length	auto *length*
content	normal | none | inhibit *uri*
counter-increment	none *identifier number*
counter-reset	none *identifier number*
crop	auto *shape*
display	normal | none | list-item
float-offset	*length length*
hyphenate-after	auto *integer*
hyphenate-before	auto *integer*
hyphenate-character	auto *integer*
hyphenate-lines	no-limit *integer*
hyphenate-resource	none *uri*
hyphens	none | manual | auto
image-resolution	normal | auto *dpi*
marks	[crop || cross] | none
move-to	normal | here *identifier*
page-policy	start | first | last
quotes	none *string string string string*

GENERATED CONTENT

string-set	*identifier* *content-list*
text-replace	none *[<string> <string>]+*

POSITIONING

bottom	auto *%* *length*
clip	*shape* auto
left	auto *%* *length*
position	static | relative | absolute | fixed
right	auto *%* *length*
top	auto *%* *length*
z-index	auto *number*

RUBY

ruby-align	auto | start | left | center | end | right | distribute-letter | distribute-space | lineedge
ruby-overhang	auto | start | end | none
ruby-position	before | after | right | inline
ruby-span	attr(x) | none

LINE BOX

alignment-adjust	auto | baseline | beforeedge | text-before-edge | middle | central | after-edge | text-after-edge | ideographic | alphabetic | hanging | mathematical *length* *%*
alignment-baseline	baseline | use-script | before-edge | text-beforeedge | after-edge | textafter-edge | central | middle | ideographic | alphabetic | hanging | mathematical
baseline-shift	baseline | sub | super *length* *%*
dominant-baseline	auto | use-script | nochange | reset-size | alphabetic | hanging | ideographic | mathematical | central | middle | text-after-edge | text-before-edge
drop-initial-after-align	*alignment-baseline*
drop-initial-after-adjust	central | middle | after-edge | text-after-edge | ideographic | alphabetic | mathematical length %
drop-initial-before-align	caps-height *alignment-baseline*
drop-initial-before-adjust	before-edge | text-beforeedge | central | middle | hanging | mathematical *length* *%*
drop-initial-value	initial integer
drop-initial-size	auto *integer* *%* *line*
inline-box-align	initial | last *integer*
line-height	normal *number* *length* *%*

LINE BOX

line-stacking	*line-stacking-strategy* *line-stacking-ruby* *line-stacking-shift*
line-stacking-strategy	inline-line-height | blockline-height | max-height | grid-height
line-stacking-ruby	exclude-ruby | include-ruby
line-stacking-shift	consider-shifts | disregardshifts
text-height	auto | font-size | text-size | max-size
vertical-align	Baseline | sub | super | top | text-top | middle | bottom | text-bottom *length* *%*

PAGED MEDIA

fit	fill | hidden | meet | slice
fit-position	[top | center | bottom] || [left | center | right] *length* *%*
image-orientation	auto *angle*
orphans	*integer*
page	auto *identifier*
page-break-after	auto | always | avoid | left | right
page-break-before	auto | always | avoid | left | right
page-break-inside	auto | avoid
size	auto | landscape | portrait *length*
windows	*integer*

UI

appearance	normal | inherit | [icon | window | desktop | workspace | document | tooltip | dialog | button | pushbutton | hyperlink | radiobutton | checkbox | menuitem | tab | menu | menubar | pull-down-menu | pop-up-menu | list-menu | radio-group | checkboxgroup | outline-tree | range | field | combo-box | signature | password]
cursor	auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | nresize | se-resize | sw-resize | s-resize | w-resize | text | wait | help *url*
icon	auto | inherit *url*
nav-index	auto | inherit *number*
nav-up	auto | inherit *<id> [current | root | <target-name>]*
nav-right	auto | inherit *<id> [current | root | <target-name>]*
nav-down	auto | inherit *<id> [current | root | <target-name>]*
nav-left	auto | inherit *<id> [current | root | <target-name>]*
resize	none | both | horizontal | vertical | inherit

PSEUDO-ELEMENT

::first-letter	Adds special style to the first letter of a text
::first-line	Adds special style to the first line of a text
::before	Inserts some content before an element
::after	Inserts some content after an element

PSEUDO-CLASS

:active	an activated element
:focus	an element while the element has focus
:visited	a visited link
:hover	an element when you mouse over it
:link	an unvisited link
:disabled	an element while the element is disabled
:enabled	an element while the element is enabled
:checked	an element (form element control) that is checked
:selection	an element that is currently selected of highlighted by the user
:lang	Allows the author to specify a language to use in a specified element
:nth-child(n)	an element that is the n-th sibling
:nth-last-child(n)	an element that is the n-th sibling counting from the last sibling
:first-child	an element that is the first sibling
:last-child	an element that is the last sibling
:only-child	an element that is the only child
:nth-of-type(n)	an element that is the n-th sibling of its type.
:nth-last-of-type(n)	an element that is the n-th sibling of its type counting from the last sibling
:last-of-type	an element that is the first sibling of its type
:first-of-type	an element that is the last sibling of its type
:only-of-type	an element that is the only child of that type
:empty	an element that has no children
:root	root element within the document
:not(x)	an element not represented by the argument 'x'
:target	a target element as specified by a target in a URI

UNITS

ABSOLUTE MEASUREMENT

%	percentage
cm	centimeter
in	inch
mm	millimeter
pc	pica (1p = 12 points)
pt	point (1pt = 1/72 inch)

RELATIVE MEASUREMENT

ch	width of the "0" glyph found in the font for the font size used to render
em	1em = current font size of current element
ex	x-height of the element's font
gd	the grid defined by 'layout-grid'
px	pixel of the viewing device
rem	the font size of the root element
vh	the viewport's height
vw	the viewport's width
vm	viewport's height or width, whichever is smaller of the two

ANGLES

deg	degrees
grad	grads
rad	radians
turn	turns

FREQUENCY

Hz	hertz
kHz	kilo-hertz

TIME

ms	milli-seconds
s	seconds

COLORS

color name	red, blue, green, dark green
rgb(x,y,z)	red = rgb(255,0,0)
rgb(x%,y%,z%)	red = rgb(100%,0,0)
rgba(x,y,z, alpha)	red = rgba(255,0,0)
#rrggbb	red = #ff0000 (or shorthand = #f00)
hsl(hue, saturation, lightness)	red = hsl(0, 100%, 50%)
hsla(hue, saturation, lightness, alpha)	red = hsl(0, 100%, 50%)
flavor	An accent color (typically chosen by the user) to customize the user interface of the user agent itself
currentColor	computed value of the 'currentColor' keyword is the computed value of the 'color' property

SELECTOR TYPES

Name	Info	Example
Universal	Any element	* { font: 10px Arial; }
Type	Any element of that type	h1 { text-decoration: underline; }
Grouping	Multiple elements of different types	h1, h2, h3 { font-family: Verdana; }
Class	Multiple elements of different types when you don't want to affect all instances of that type	.sampleclass { textdecoration: underline; }
Id	A single element type when you don't want to affect all instances of that type	#sampleid { textdecoration: underline; }
Descendant	An element that is below (in the document tree) another element—no matter how many levels below	#gallery h1 { textdecoration: underline; }
Child	An element that is directly below (in the document tree) another element	#title > p { font-weight: bold; }
Adjacent Sibling	All elements that share the same parent and elements are in the same immediate sequence	h1 + p { font-style: italic; }
General Sibling	All elements that share the same parent and elements are in the same sequence (not necessarily immediate)	h1 ~ p { font-style: italic; }
Attribute	An element with that matches the attribute listed	E[selected] - att whatever the value E[att="val"] - att with a specific value E[rel~="next"] - att with a value is a whitespace separated list *[lang|="en"] - att value either being exactly "val" or beginning with "val" immediately followed by "-" E[att^="val"] - att value that begins with the prefix "val" E[att$="val"] - att value that end with the suffix "val" E[att*="val"] - att value contains at least one instance of the substring "val"

Contents

JavaScript Tutorial: Basics to Advanced Concepts

JavaScript Tutorial: Basics to Advanced Concepts

(.....Continued)

21. JavaScript Memory Allocation and Event Loop
22. JavaScript Error Handling
23. ES5 Vs ES6: What is the Difference

Let's get Started

Introduction to JavaScript:

JavaScript is a programming language that, together with HTML and CSS, is one of the core technologies of the World Wide Web. Over 97% of websites employ JavaScript on the client-side for web page behaviour, typically combining third-party libraries. When it comes to web development, JavaScript is one of the most popular languages owing to its **features and capabilities**.

This article includes a JavaScript cheat sheet as well as rich JavaScript documentation and how-tos to make it easy for our readers to work with JavaScript. The purpose of the cheat sheet is to give you some quick, correct, and ready to use code snippets for common circumstances. In a nutshell, this article makes **JavaScript** simple for both novice and professional coders.

JavaScript Tutorial: Basics to Advanced Concepts

1. JavaScript Fundamentals

- **Writing JavaScript code in an HTML page**: We must enclose the JavaScript code within the <script> tag in order to include it on an HTML page just as the example shown below:

```
<script type = "text/javascript">
//JavaScript coding can be done inside this tag
</script>
```

With this information, the browser can correctly make out that the code is written in JavaScript and execute the code.

- **Inclusion of external JavaScript files in an HTML file**: An external JavaScript file can also be written separately and included within our HTML file. That way, different types of code can be kept isolated from one another, resulting in better-organised files. For instance, if our JavaScript code is written in the file script.js, we can include it in our HTML file in the following way:

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

- **Usage of Comments in JavaScript coding:** Comments are extremely useful in programming because they can assist others to understand what's going on in your code or they can help you if you have forgotten something. Keep in mind that they must be correctly identified so that the browser does not attempt to execute them. There are two alternatives available in JavaScript in which we can add comments:
  - **Single-line comments:** If you want to include a single line comment, start it with "//".
  - **Multi-line comments**: If you want to write a comment that spans multiple lines, you can wrap it in /* and */ to prevent it from being executed.

2. Javascript Variables

Variables in JavaScript are simply names of storage locations. In other words, they can be considered as stand-in values that we can use to perform various operations in our JavaScript codes. JavaScript allows the usage of variables in the following three ways:

- **var**: The most commonly used variable in JavaScript is var. It can be redeclared and its value can be reassigned, but only inside the context of a function. When the JavaScript code is run, variables defined using var are moved to the top. An example of a variable declared using the "var" keyword in JavaScript is shown below:

```
var x = 140; //  variable x can be reassigned a new value and also redeclared
```

- **const**: const variables in JavaScript cannot be used before they appear in the code. They can neither be reassigned values, that is, their value remains fixed throughout the execution of the code, nor can they be redeclared. An example of a variable declared using the "const" keyword in JavaScript is shown below:

```
const x = 5; // variable x cannot be reassigned a new value or redeclared
```

- **let**: The let variable, like const, cannot be redeclared. But they can be reassigned a value. An example of a variable declared using the "let" keyword in JavaScript is shown below:

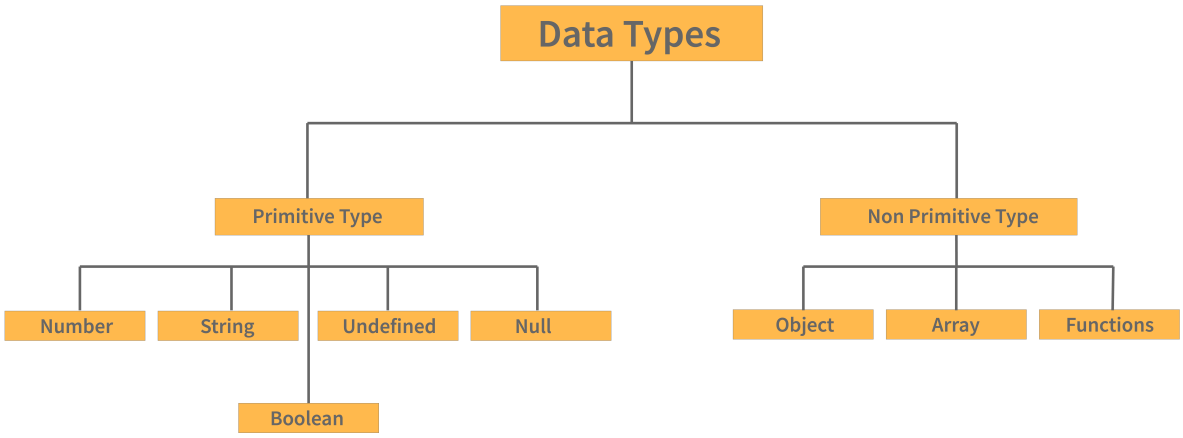
```
let x = 202; // variable x cannot be redeclared but can be reassigned a new value
```

3. Javascript Data Types

Different types of values and data can be stored in JavaScript variables. To assign values to JavaScript variables, you use the equals to "=" sign operator. The various data types in JavaScript are as follows:

- Numbers: These are just numerical values. They can be real numbers or integers. An example of the numbers data type is shown below: `var id = 100`
- Variables: The variable data type, as the name suggests, does not have a fixed value. An example of the variables data type is shown below: `var y`
- Text (strings): String data types of JavaScript are a combination of multiple characters. An example of the string data type is shown below: `var demoString = "Hello World"`
- Operations: Operations in JavaScript can also be assigned to JavaScript variables. An example of these is shown below: `var sum = 20 + 30 + 29`
- Boolean values: Boolean values can be true or false. An example of the boolean data type is shown below: `var booleanValue = true`
- Constant numbers: As the name suggests, these data types have a fixed value. An example of the constant data type is shown below: `const g = 9.8`
- Objects: JavaScript objects are containers for named values called properties. They possess their own data members and methods. An example of the objects data type is shown below:

```
var name = {name:"Jon Snow", id:"AS123"}
```



> Note: It is important to remember that variables in JavaScript are case-sensitive. As a result, "small" and "Small" will be treated as separate variables in JavaScript.

4. JavaScript Operators

You can use variables to conduct a variety of tasks using JavaScript operators. The various types of operators in JavaScript are as follows:

- **Fundamental Operators**: These operators are used to perform basic operations like addition, multiplication, etc. in JavaScript. A list of all the Fundamental operators in JavaScript is as follows:
  - **+**: The Addition Operator is used to add two numbers
  - **-**: The Subtraction Operator is used to subtract two numbers
  - *****: The MultiplicationOperator is used to multiply two numbers
  - **/**: The Division Operator is used to divide two numbers
  - **(...)**: In general, operations within brackets are executed earlier than that outside. This grouping operator surrounds an expression or sub-expression with a pair of parentheses to override the conventional operator precedence, allowing operators with lower precedence to be evaluated before operators with higher precedence. It does exactly what it says: it groups the contents of the parentheses.
  - **%**: The Modulus operator is used to get the remainder when an integer number is divided by another integer number.
  - **++**: The Increment operator is used to increase the value of numbers by one.
  - **--**: The Decrement operator is used to decrease the value of numbers by one.
- **Bitwise Operators**: All the operations dealing with the bits of the numbers can be performed by the bitwise operators in JavaScript. A list of all the Bitwise operators in JavaScript is as follows:
  - **&**: The bitwise AND operator returns a 1 in every bit position where both operands' corresponding bits are 1.
  - **|**: The bitwise OR operator (|) returns a 1 in each bit position where either or both operands' corresponding bits are 1.
  - **~**: The bitwise NOT operator reverses the operand's bits. It turns the operand into a 32-bit signed integer, just like other bitwise operators.
  - **^**: The bitwise XOR operator () returns a 1 in each bit position where the corresponding bits of both operands are 1s but not both.
  - **<<**: The left shift operator shifts the first operand to the left by the provided number of bits. Extra bits that have been relocated to the left are discarded. From the right, zero bits are shifted in.
  - **>>**: The right shift operator (>>) moves the first operand to the right by the provided number of bits. Extra bits that have been relocated to the right are discarded. The leftmost bit's copies are shifted in from the left. The sign bit (the leftmost bit) does not change since the new leftmost bit has the same

5. JavaScript If-Else Statements

The if-else statements are simple to comprehend. You can use them to set conditions for when your code runs. If specific requirements are met, something is done; if they are not met, another action is taken. The switch statement is a concept that is comparable to if-else. The switch however allows you to choose which of several code blocks to run. The syntax of if-else statements in JavaScript is given below:

```
if (check condition) {
   // block of code to be executed if the given condition is satisfied
} else {
   // block of code to be executed if the given condition is not satisfied
}
```

Loops in JavaScript:

Most programming languages include loops. They let you run code blocks as many times as you like with different values. Loops can be created in a variety of ways in JavaScript:

- **the for loop**: The most frequent method of creating a loop in JavaScript. Its syntax is shown below:

```
for (initialization of the loop variable; condition checking for the loop; updation aft
   // code to be executed in loop
}
```

- **the while loop**: Establishes the conditions under which a loop will run. Its syntax is shown below:

```
// Initialization of the loop variable is done before the while loop begins
while(condition checking for the loop){
// 1. code to be executed in loop
// 2. updation of the loop variable
}
```

- **the do-while loop**: Similar to the while loop, but it runs at least once and checks at the end to see whether the condition is met to run again. Its syntax is shown below:

```
// Initialization of the loop variable is done before the do-while loop begins
do{
// 1. code to be executed in loop
// 2. updation of the loop variable
}while(condition checking for the loop);
```

There are two statements that are important in the context of loops:

- **the continue statement**: Skip parts of the loop if certain conditions are met.
- **break statement:** Used to stop and exit the cycle when specific conditions are met.

6. JavaScript Arrays

Arrays are the next item on our JavaScript cheat sheet. Arrays are used in a variety of programming languages. They are a method of categorising variables and attributes. Arrays can be defined as a collection of objects of the same type. In JavaScript, here's how one can make an array of cars:

```
var cars = {"Mercedes", "Tesla","Volvo"};
```

Now that we understand how to make arrays, we can perform a bunch of operations on them. Let us take a look at some JavaScript methods which can be used to perform various types of operations on arrays:

- **pop():** This method is used for removing the last element of an array.
- **push():** This method is used for adding a new element at the very end of an array.
- **concat():** This method is used for joining various arrays into a single array.
- **reverse():** This method is used for reversing the order of the elements in an array.
- **shift():** This method is used for removing the first element of an array.
- **slice():** This method is used for pulling a copy of a part of an array into a new array.
- **splice()**: This method is used for adding elements in a particular way and position.
- **toString():** This method is used for converting the array elements into strings.
- **unshift():** This method is used for adding new elements at the beginning of the array.
- **valueOf():** This method is used for returning the primitive value of the given object.
- **indexOf():** This method is used for returning the first index at which a given element is found in an array.
- **lastIndexOf():** This method is used for returning the final index at which a given element appears in an array.
- **join():** This method is used for combining elements of an array into one single string and then returning it.
- **sort():** This method is used for sorting the array elements based on some condition.

7. JavaScript Functions

JavaScript Functions can be defined as chunks of code written in JavaScript to perform a single task. A function in JavaScript looks like this:

```
function nameOfTheFunction(parameterOne, parameterTwo, parameterThree, parameterFour,..
   // Job or Task of the function
}
```

The code above consists of the "function" keyword and a name, as you can see. The parameters of the function are enclosed in brackets, while the function's task code and output is enclosed in curly brackets. You can make your own, but there are a few default functions to make your life easier. Although we will be discussing various methods throughout this cheat sheet, let us discuss in brief two important types of JavaScript functions in this section:

- **Functions For Throwing Data As Output**: The output of data is a common application for functions. You have the following options for outputting data:
  - **prompt()**: This function is used for creating a dialogue box for taking input from the user.
  - **alert()**: This function is used for outputting information in an alert box in the browser window
  - **console.log()**: This function is used for writing data to the browser's console and is used for the purpose of debugging code by developers.
  - **document.write()**: This function is used for writing straight to our HTML document
  - **confirm()**: This function is used for opening up a yes or no dialogue box and for returning a boolean value depending upon the user's click
- **Global Functions:** Every browser that can run JavaScript has a set of global functions built-in. Some of them are as follows:
  - **parseFloat():** This function is used for parsing the argument passed to it and returning a floating-point number.
  - **parseInt():** This function is used for parsing the argument passed to it and returning an integral number.
  - **encodeURI():** This function is used for encoding a URI into a UTF-8 encoding scheme.
  - **decodeURI():** This function is used for decoding a Uniform Resource Identifier (URI) made by encodeURI() function or similar functions.
  - **encodeURIComponent():** This function is used for the same purpose as encodeURI() only for URI components.
  - **decodeURIComponent():** This function is used for decoding a URI component.
  - **isNaN():** This function is used for determining if a given value is Not a Number or not.
  - **Number():** This function is used for returning a number converted from what is passed as an argument to it.
  - **eval():** This function is used for evaluating JavaScript programs presented as strings.
  - **isFinite():** This function is used for determining if a passed value is finite or not.

8. Scope and Scope Chain in JavaScript

1. Scope: The accessibility or visibility of variables in JavaScript is referred to as scope. That is, which sections of the program can access a given variable and where the variable can be seen. There are usually three types of scopes:

- **Global Scope:** The global scope includes any variable that is not contained within a function or block (a pair of curly braces). Global scope variables can be accessed from anywhere in the program. An example showing the global scope of a variable is given below:

```
var hello = 'Hello!';
function sayHello() {
 console.log(hello);
}
// 'Hello!' gets logged
sayHello();
```

- **Local or Function Scope:** Variables declared inside a function are local variables. They can only be accessed from within that function; they are not accessible from outside code. An example showing local scope of a variable is given below:

```
function sayHello() {
 var hello = 'Hello!';
 console.log(hello);
}
// 'Hello!' gets logged
sayHello();
```

console.log(hello); // Uncaught ReferenceError: hello is not defined

- **Block Scope**: Unlike var variables, let and const variables can be scoped to the nearest pair of curly brackets in ES6. They can't be reached from outside that pair of curly braces, which means they can't be accessed from the outside. An example showing the block scope of a variable is given below:

```
{
 let hello = 'Hello!';
 var language = 'Hindi';
 console.log(hello); // 'Hello!' gets logged
}
console.log(language); // 'Hindi!' gets logged
console.log(hello); // Uncaught ReferenceError: hello is not defined
```

2. Scope Chain: When a variable is used in JavaScript, the JavaScript engine searches the current scope for the variable's value. If it can't find the variable in the inner scope, it will look into the outer scope until it finds it or reaches the global scope.

If it still can't identify the variable, it will either return an error or implicitly declare the variable in the global scope (if not in strict mode). Let us take into consideration the following example:

```
let a = 'a';
function foo() {
 let b = 'b';
 console.log(b); // 'b' gets logged
 console.log(a); // 'a' gets logged
 randomNumber = 33;
 console.log(randomNumber);  // 33 gets logged
}
foo();
```

When the function foo() is called, the JavaScript engine searches for the 'b' variable in the current scope and finds it. Then it looks for the 'a' variable in the current scope, which it can't find, so it moves on to the outer scope, where it finds it (i.e global scope).

After that, we assign 33 to the 'randomNumber' variable, causing the JavaScript engine to search for it first in the current scope, then in the outer scope.

If the script isn't in strict mode, the engine will either create a new variable called randomNumber and assign 33 to it, or it will return an error (if not in strict mode).

As a result, the engine will traverse the scope chain till the time when a variable is found.

9. JavaScript Hoisting

Prior to executing the code, the interpreter appears to relocate the declarations of functions, variables, and classes to the top of their scope using a process known as Hoisting in JavaScript. Functions can be securely utilised in code before they have been declared thanks to hoisting. Variable and class declarations are likewise hoisted, allowing them to be referenced prior to declaration. It should be noted that doing so can result in unforeseen mistakes and is not recommended. There are usually two types of Hoisting:

- **Function Hoisting**: Hoisting has the advantage of allowing you to use a function before declaring it in your code as shown in the code snippet given below. Without function hoisting, we would have to first write down the function display and only then can we call it.

```
display("Lion");
function display(inputString) {
 console.log(inputString); // 'Lion' gets logged
}
```

- **Variable Hoisting:** You can use a variable in code before it is defined and/or initialised because hoisting works with variables as well. JavaScript, however, only hoists declarations, not initializations! Even if the variable was initially initialised then defined, or declared and initialised on the same line, initialization does not occur until the associated line of code is run. The variable has its default initialization till that point in the execution is reached (undefined for a variable declared using var, otherwise uninitialized). An example of variable hoisting is shown below:

```
console.log(x) // 'undefined' is logged from hoisted var declaration (instead of 7)
var x // Declaration of variable x
x = 7; // Initialization of variable x to a value 7
console.log(x); // 7 is logged post the line with initialization's execution.
```

10. JavaScript Closures

A closure is a function that has been bundled together (enclosed) with references to its surroundings (the lexical environment). In other words, a closure allows an inner function to access the scope of an outside function. Closures are formed every time a function is created in JavaScript, during function creation time. An example of closures in Javascript is given below:

```
function subtractor(subtractingInteger) {
 return function(a) {
   return a - subtractingInteger;
 };
}
var subtract2 = subtractor(2);
var subtract5 = subtractor(5);
console.log(subtract2(5));  // 3 is logged
console.log(subtract5(5)); // 0 is logged
```

In this example, we have developed a function subtractor(subtractingInteger) that takes a single parameter subtractingInteger and returns a new function. Its return function accepts only one input, a, and returns the difference of a and subtractingInteger. The function 'subtractor' is essentially a function factory. It creates functions that have the ability to subtract a specified value from their arguments. The function factory creates two new functions in the example above: one that subtracts 2 from its argument and one that subtracts 5 from its arguments. Both subtract2 and subtract5 are closures. They have the same function body definition, but they hold lexical surroundings that are distinct. subtractingInteger is 2 in subtract2's lexical environment, but subtractingInteger is 5 in subtract5's lexical environment.

11. JavaScript Strings

As mentioned earlier, Strings are nothing but a combination of characters that can be used to perform a variety of tasks. JavaScript provides so many methods for Strings alone that it makes sense to cover Strings as a standalone topic in this cheat sheet. Let us now look at the various escape sequences in JavaScript and the methods which JavaScript provides for strings:

- **Escape Sequences or Escape Characters:** An escape character is a character in computers and telecommunications that causes the following characters in a character sequence to take on a different meaning. Metacharacters include escape characters, which are a subset of metacharacters. In general, whether something is an escape character or not is determined by the context. For instance, Strings in JavaScript are delimited by single or double-quotes. You must use special characters in a string if you want to utilise quote marks. A few of the escape characters allowed by JavaScript are as follows:
  - **\'** — Single quotes
  - **\"** — Double quotes
  - **\t** — Horizontal tab
  - **\v** — Vertical tab
  - \\ — Backslash
  - **\b** — Backspace
  - **\f** — Form feed
  - **\n** — Newline
  - **\r** — Carriage return
- **String methods:** As mentioned earlier, JavaScript provides a lot of methods to manipulate its Strings. Let us take a look at some of them:
  - **toLowerCase()** — This method is used for converting strings to lower case
  - **toUpperCase()** — This method is used for converting strings to upper case
  - **charAt()** — This method is used for returning the character at a particular index of a string
  - **charCodeAt()** — This method is used for returning to us the Unicode of the character at a given index
  - **fromCharCode()** — This method is used for returning a string made from a particular sequence of UTF-16 code units
  - **concat()** — This method is used for concatenating or joining multiple strings into a single string
  - **match()** — This method is used for retrieving the matches of a string against a pattern string which is provided
  - **replace()** — This method is used for finding and replacing a given text in the string
  - **indexOf()** — This method is used for providing the index of the first appearance of a given text inside the string
  - **lastIndexOf()** — This method is similar to the indexOf() methods and only differs in the fact that it searches for the last occurrence of the character

12. Document Object Model (DOM) in JavaScript

The Document Object Model (DOM) is the structure of a webpage's code. There are many different ways to build and alter HTML elements with JavaScript (called nodes).

- **Node Properties:** Let us first take a look at some of the properties of a JavaScript DOM node:
  - **attributes** — Gets a live list of all the characteristics associated with an element.
  - **baseURI** — Returns an HTML element's absolute base URL.
  - **childNodes** — Returns a list of the child nodes of an element.
  - **firstChild** — Returns the element's first child node.
  - **lastChild** — An element's final child node
  - **nextSibling** — Returns the next node in the same node tree level as the current node.
  - **nodeName** —Returns a node's name.
  - **nodeType** — Returns the node's type.
  - **nodeValue** — Sets or returns a node's value.
  - **ownerDocument** — This node's top-level document object.
  - **parentNode** — Returns the element's parent node.
  - **previousSibling** — Gets the node that comes before the current one.
  - **textContent** — Sets or returns a node's and its descendants' textual content.
- **Node Methods:** Let us now take a look at some of the methods provided by JavaScript to manipulate these nodes in the DOM:
  - **appendChild()** — Adds a new child node as the last child node to an element.
  - **cloneNode()** is a function that duplicates an HTML element.
  - **compareDocumentPosition()** — Compares two elements' document positions.
  - **getFeature()** returns an object that implements the APIs of a feature.
  - **hasAttributes()** — If an element has any attributes, it returns true; otherwise, it returns false.
  - **hasChildNodes()** — If an element has any child nodes, it returns true; otherwise, it returns false.
  - **insertBefore()** — Adds a new child node to the left of an existing child node.
  - **isDefaultNamespace()** returns true if a given namespaceURI is the default, and false otherwise.
  - **isEqualNode()** — Determines whether two elements are the same.
  - **isSameNode()** — Determines whether two elements belong to the same node.
  - **isSupported()** — Returns true if the element supports the provided feature.

13. JavaScript Data Transformation

Data Transformation in JavaScript can be done with the usage of higher-order functions. Higher-order functions are those functions in JavaScript which can accept one or more functions as inputs and return a function as the result. All higher-order functions that take a function as input are map(), filter(), and reduce(). Let us now take a look at how these functions can be used. One thing to note over here is that because all of these functions are part of the JavaScript Array prototype, they can be used directly on an array.

map() method: The map method applies a function to each array element. The callback function receives each element of the array and returns a new array of the same length. This method can be used to conduct the same operation/transformation on each element of an array and return a new array with the modified values of the same length. An example of the usage of the map() method is given below:

```
var arr = [10,20,30];
var triple  = arr.map(x => x * 3);
triple; // [30,60,90]
```

filter() method: Using the filter() method, items that do not meet a criterion are removed from the array. The callback function receives every element of the array. If the callback returns true on each iteration, the element will be added to the new array; otherwise, it will not be added. An example of the usage of the filter() method is given below:

```
var arr = [13,40,47];
var odd = arr.filter(x => x % 2);
odd; // [13,47]
```

reduce() method: The reduce() method can combine the items of an array into a single value. When using reduce, we must declare the accumulator's beginning value (final result). Each iteration, we do some operation inside the callback, which is then added to the accumulator. An example of the usage of the reduce() method is given below:

```
var arr = [10,20,30];
var counter = 0;
let answer = arr.reduce((accumulator, value) => value + accumulator, counter);
console.log(answer) // answer = 10 + 20 + 30 = 60
```

14. JavaScript Regular Expressions

Regular expressions can be defined as search patterns that can be used to match string character combinations. Text search and text to replace procedures can both benefit from the search pattern. Let us look at how JavaScript allows Regular Expressions:

- **Pattern Modifiers**: Following are the pattern modifiers that are allowed in JavaScript:
  - **e** — This is used for evaluating replacement
  - **i** — This is used for performing case-insensitive matching
  - **U** — This is used for ungreedy pattern
  - **g** — This is used for performing global matching
  - **m** — This is used for performing multiple line matching
  - **s** — This is used for treating strings as a single line
  - **x** — This is used for allowing comments and whitespace in the pattern
- **Metacharacters:** Following are the metacharacters that are allowed in JavaScript:
  - **.** — This is used for finding a single character, except newline or line terminator
  - **\w** — This is used for finding Word characters
  - **\W** — This is used for finding Non-word characters
  - **\s** — This is used for finding Whitespace characters
  - **\S** — This is used for finding Non-whitespace characters
  - **\b** — This is used for finding matches at the beginning or at the end of a word
  - **\B** — This is used for finding matches not at the beginning or at the end of a word
  - **\0** — This is used for finding NULL characters
  - **\n** — This is used for finding a new line character
  - **\f** — This is used for finding a Form feed character
  - **\r** — This is used for finding a Carriage return character
  - **\t** — This is used for finding a Tab character
  - **\v** — This is used for finding a Vertical tab character
  - **\d** — This is used for finding digits
  - **\D** — This is used for finding non-digit characters
  - **\xxx** — This is used for finding characters given by an octal number xxx
  - **\xdd** — This is used for finding characters given by a hexadecimal number dd
  - **\uxxxx** — This is used for finding the Unicode character given by a hexadecimal number XXXX
- **Brackets:** You can group parts of a regular expression together by putting them inside round brackets or parentheses. You can use this to apply a quantifier to the entire group or to limit the alternation to a specific area of the regex. For

15. Numbers and Mathematics in JavaScript

JavaScript provides various properties and methods to deal with Numbers and Maths. Let us have a quick look at those:

- **Numbers Properties:**
  - **MAX VALUE** — The maximum numeric value that JavaScript can represent.
  - **NaN** — The "Not-a-Number" value is NaN.
  - **NEGATIVE INFINITY** – The value of Infinity is negative.
  - **POSITIVE INFINITY** – Infinity value that is positive.
  - **MIN VALUE** — The smallest positive numeric value that JavaScript can represent.
- **Numbers Methods:**
  - **toString()** — Returns a string representation of an integer.
  - **toFixed()** — Returns a number's string with a specified number of decimals.
  - **toPrecision()** — Converts a number to a string of a specified length.
  - **valueOf()** — Returns a number in its original form.
  - **toExponential()** — Returns a rounded number written in exponential notation as a string.
- **Maths Properties:**
  - **E** — Euler's number is E.
  - **SQRT1_2** — 1/2 square root
  - **SQRT2** stands for square root of two.
  - **LOG2E** — E's base 2 logarithm
  - **LN2** — The natural logarithm of 2 is LN2.
  - **LN10** — The natural logarithm of ten is LN10.
  - **LOG10E** — E's base ten logarithm
  - **PI** — PI stands for Pianistic Integer.
- **Maths Methods:**
  - **exp(x)** — Ex's value
  - **floor(x)** — x's value rounded to the nearest integer.
  - **log(x)** — The natural logarithm (base E) of x is log(x).
  - **abs(x)** — Returns the value of x in its absolute (positive) form.
  - **acos(x)** — In radians, the arccosine of x.
  - **asin(x)** — In radians, the arcsine of x.
  - **pow(x,y)** — x to the power of y
  - **random()** — Returns a number between 0 and 1 at random.
  - **round(x)** — Rounds the value of x to the nearest integer.
  - **sin(x)** — The sine of x is sin(x) (x is in radians)
  - **sqrt(x)** — x's square root
  - **tan(x)** — The angle's tangent

16. JavaScript Date Objects

Dates are extremely important to deal with while creating most types of applications. JavaScript also allows you to deal with and change dates and times. The next section of the JavaScript cheat sheet is how to work on dates:

- **Setting Dates:** Dates can be set using the following three ways:
  - **Date()** — Returns a new date object that contains the current date and time.
  - **Date(1993, 6, 19, 5, 12, 50, 32)** — We can create a custom date object with the pattern as Year, month, day, hour, minutes, seconds, and milliseconds are represented by the numbers. Except for the year and month, we can omit anything we like.
  - **Date("1999-12-22")** — Date as a string declaration
- **Getting the values of Time and Date:** The following methods can be used for getting the date and time values in JavaScript:
  - **getDate()** returns the month's day as a number (1-31)
  - **getTime()** — Get the milliseconds since January 1, 1970
  - **getUTCDate()** returns the month's day (day) in the supplied date in universal time (also available for day, month, full year, hours, minutes etc.)
  - **getMilliseconds()** — This function returns the milliseconds (0-999)
  - **getMinutes()** — Returns the current minute (0-59)
  - **getMonth()** returns the current month as a number (0-11)
  - **parse** — It returns the number of milliseconds since January 1, 1970 from a string representation of a date.
  - **getDay()** returns a number representing the weekday (0-6)
  - **getFullYear()** returns the current year as a four-digit value (yyyy)
  - **getHours()** — Returns the current hour (0-23)
  - **getSeconds()** — Returns the second number (0-59)
- **Setting a Part of the Dates:** We can set a part of the dates in JavaScript using the following methods:
  - **setDate()** — Returns the current date as a number (1-31)
  - **setFullYear()** — This function sets the year (optionally month and day)
  - **setMonth()** — This function sets the month (0-11)
  - **setSeconds()** — This function sets the seconds (0-59)
  - **setTime()** — This function is used to set the time (milliseconds since January 1, 1970)
  - **setMinutes()** — This function sets the minutes (0-59)
  - **setUTCDate()** — According to universal time, sets the day of the month for a given date (also available for day, month, full-year, hours, minutes etc.)
  - **setHours()** — Changes the time (0-23)
  - **setMilliseconds()** — This function sets the milliseconds (0-999)

17. JavaScript Browser Objects

JavaScript is also capable of taking note of the user's browser activity and incorporating its properties into the code, in addition to HTML elements.

- **Given below is a list of Window properties that JavaScript can take into account:**
  - **history** — Provides the window's History object.
  - **innerHeight** — The content area of a window's inner height.
  - **innerWidth** — The content area's inner width.
  - **closed** — Returns true or false depending on whether or not a window has been closed.
  - **pageXOffset** — The number of pixels offset from the centre of the page. The current document has been horizontally scrolled.
  - **pageYOffset** — The number of pixels offset from the centre of the page. The document has been vertically scrolled.
  - **navigator** — Returns the window's Navigator object.
  - **opener** — Returns a reference to the window that created the window.
  - **outerHeight** — A window's total height, including toolbars and scrollbars.
  - **outerWidth** — A window's outside width, including toolbars and scrollbars.
  - **defaultStatus** — Changes or restores the default text in a window's status bar.
  - **document** — Returns the window's document object.
  - **frames** — All <iframe> elements in the current window are returned by frames.
  - **length** — Determine how many iframe> elements are in the window.
  - **location** — Returns the window's location object.
  - **name** — Sets or retrieves a window's name.
  - **parent** — The current window's parent window is called parent.
  - **screen** — Returns the window's Screen object.
  - **screenLeft** — The window's horizontal coordinate (relative to the screen)
  - **screenTop** — The window's vertical coordinate.
  - **self** — Returns the window that is currently open.
  - **status** — Changes or restores the text in a window's status bar.
  - **top** — Returns the browser window that is currently at the top of the screen.
  - **screenX** — Identical to screenLeft, but required by some browsers
  - **screenY** — Identical to screenTop, but required by some browsers
- **Given below are the JavaScript methods which can work on the user's browser window:**
  - **alert()** — Shows a message and an OK button in an alert box.
  - **setInterval()** — Calls a function or evaluates an expression at intervals defined by the user.

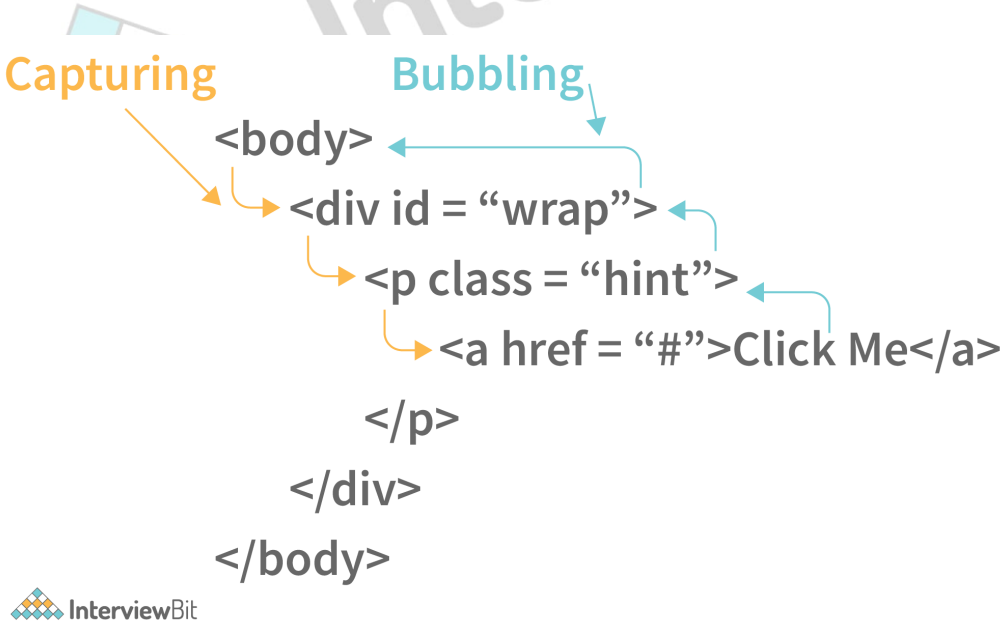
18. JavaScript Events

Things that can happen to HTML components and are executed by the user in JavaScript are called events. These events can be detected by the programming language, which can then be used to activate code actions. Without them, no JavaScript cheat sheet would be complete. Let us take a look at some of the events supported by JavaScript:

- **Mouse Events:**
  - **onclick** – When a user clicks on an element, an event is triggered.
  - **oncontextmenu** — When a user right-clicks on an element, a context menu appears.
  - **ondblclick** — When a user double-clicks on an element, it is called ondblclick.
  - **onmousedown** — When a user moves their mouse over an element, it is called onmousedown.
  - **onmouseenter** — The mouse pointer is moved to a certain element.
  - **onmouseleave** — The pointer leaves an element.
  - **onmousemove** — When the pointer is over an element, it moves.
  - **onmouseover** — When the cursor is moved onto an element or one of its descendants, the term onmouseover is used.
  - **onmouseout** — When the user moves the mouse cursor away from an element or one of its descendants, it is called onmouseout.
  - **onmouseup** — When a user releases a mouse button while hovering over an element, it is known as onmouseup.
- **Form Events:**
  - **onblur** — When an element loses focus, it is called onblur.
  - **onchange** — A form element's content changes. (for the input>, select>, and textarea> elements)
  - **onfocus** – An aspect is brought into focus.
  - **onfocusin** — When an element is ready to become the centre of attention.
  - **onfocusout** —The element is about to lose focus.
  - **oninput** — When a user inputs something into an element, it's called oninput.
  - **oninvalid** — If an element isn't valid, it's called oninvalid.
  - **onreset** — The state of a form is reset.
  - **onsearch** — A user enters text into a search field (for input="search">).
  - **onselect** — The user chooses some text (input> and textarea>).
  - **onsubmit** — A form is filled out and submitted.
- **Drag Events**:
  - **ondrag** — When an element is dragged, it is called ondrag.
  - **ondragend** — The element has been dragged to its final position.
  - **ondragenter** — When a dragged element enters a drop target, it is called ondragenter.
  - **ondragleave** — When a dragged element departs the drop target, it is

19. JavaScript Event Propagation

Event propagation is a technique that governs how events propagate or travel through the DOM tree to reach their destination, as well as what happens to them once they arrive. Consider the following scenario: you have been given a click event handler to a hyperlink (i.e. <a> element) that's nested inside a paragraph (i.e. <p> element). The handler will now be executed if you click on that link. However, if you set the click event handler to the paragraph containing the link instead of the link, the handler will be triggered regardless of whether the link is clicked. Because events go up and down the DOM tree to reach their target, they don't merely affect the target element that triggered the event. This is known as event propagation.



When an event is fired on an element with parent elements, the above picture shows how the event travels through the DOM tree at different stages of the event propagation. Event propagation in current browsers is divided into two phases: capturing and bubbling.

- **The Capturing Phase**: In the capturing phase, events propagate from the Window down through the DOM tree to the target node. For example, if the user clicks a hyperlink, that click event would pass through the <html> element, the <body> element, and the <p> element containing the link. Also if any ancestor (i.e. parent, grandparent, etc.) of the target element and the target itself has a specially registered capturing event listener for that type of event, those listeners are executed during this phase.
- **The Bubbling Phase:** From the target element up to the Window, the DOM tree visits all of the target element's ancestors one by one. When a user hits a hyperlink, the click event passes via the <p> element containing the link, the <body> element, the <html> element, and the document node, for example. Additionally, if the target element or any of its ancestors have event handlers for that sort of event, those handlers are run during this phase. By default, all event handlers in current browsers are registered at the bubbling phase.

20. Asynchronous JavaScript

A number of Web API features now use asynchronous code for running, especially those that access or fetch a resource from external devices, for instance, retrieving files from the network, accessing some database and returning data to it, accessing a video stream from a webcam, or broadcasting the display to a VR headset. There are two ways in which asynchronous coding can be done in JavaScript:

- **Async Callbacks:** When invoking a function, async callbacks are functions that are passed as arguments and begin executing code in the background. When the background code is finished, it runs the callback function to notify you that the work is complete or that anything interesting has occurred. Callbacks are a little out of date these days, but they're still utilised in a lot of older but still popular APIs. The second parameter of the addEventListener() method is an example of an async callback:

```
buton.addEventListener('click', () => {
 alert('Button has been clicked!');
 let paraElement = document.createElement('p');
 paraElement.textContent = 'A new paragraph.';
 document.body.appendChild(paraElement);
});
```

The first parameter specifies the type of event to be listened for, while the second specifies a callback function to be called when the event occurs. When we give a callback function as an input to another function, we are merely passing the function's reference; the callback function isn't immediately performed. It is asynchronously "called back" (thus the name) somewhere within the containing function's body. When the time comes, the contained function is in charge of calling the callback function.

- **Promises:** Promises are a new async programming paradigm that you'll see in current Web APIs. The get() API, which is essentially a newer, more efficient version of XMLHttpRequest, is a nice example. Let's take a look at an example from our post Fetching data from the server:

```
fetch(items.json').then(function(res) {
 return res.json();
}).then(function(json) {
 let item = json;
 initialise(item);
}).catch(function(e) {
 console.log('Fetch error: ' + e.message);
});
```

Here, fetch() takes a single argument: the URL of a network resource we wish to fetch and a promise is returned. A promise is an object that represents whether the async operation succeeded or failed. In a sense, it reflects a transitional condition. In essence, it's the browser's way of saying, "I promise to respond as quickly as I can," hence the name "promise."

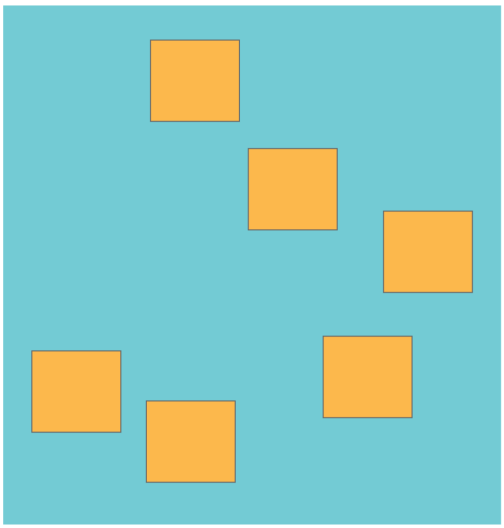
> Note: Pyramid of Doom, also known as Callback Hell, is an anti-pattern found in asynchronous computer code. It is a slang phrase for a large number of interconnected "if" statements or functions. A few callbacks appear harmless if you do not expect your application logic to become too sophisticated. However, as your project's requirements grow, you will quickly find yourself with layers upon layers of nested callbacks. The usage of callbacks makes writing and maintaining code more challenging. It also makes it more difficult to identify the application's flow, which is a debugging roadblock, hence the problem's well-known nickname: Callback Hell. An example of callback hell is given below:

```
fs.readdir(source, function (error, file) {
 if (error) {
   console.log(Problem occurred while finding file: ' + error)
 } else {
   file.forEach(function (filename, fileIndex) {
     console.log(filename)
     gm(source + filename).size(function (error, value) {
       if (error) {
         console.log(Problem occurred while identifying file size: ' + error)
       } else {
         console.log(filename + ' : ' + value)
         aspect = (values.width / values.height)
         widths.forEach(function (width, widthIndex) {
           height = Math.round(width / aspect)
           console.log('resizing ' + filename + 'to ' + height + 'x' + height)
           this.resize(width, height).write(dest + 'w' + width + '_' + filename, functi
             if (error) console.log('Problem occurred while writing file: ' + error)
           })
         }.bind(this))
       }
     })
   })
 }
})
```

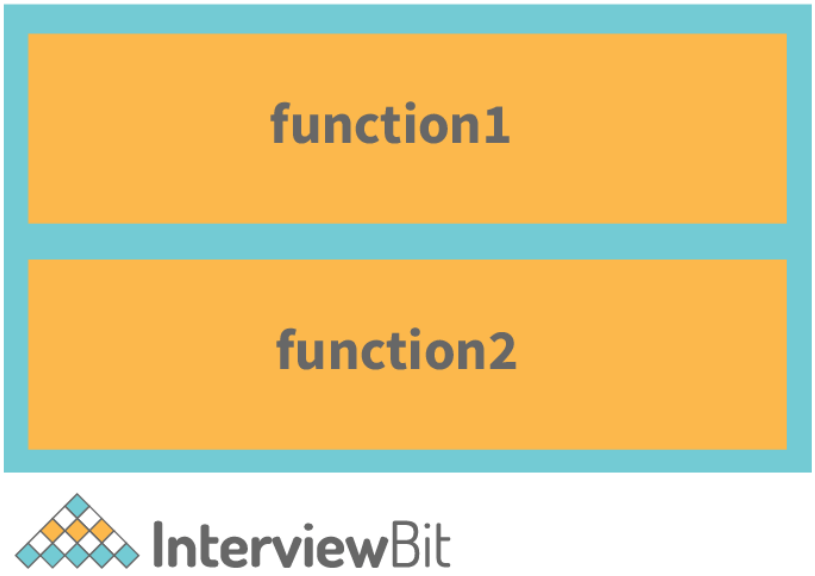
21. JavaScript Memory Allocation and Event Loop

In JavaScript, memory allocation is done in the following regions:

- **Heap memory:** Data is stored in random order and memory is allocated accordingly.



- **Stack memory:** Memory that is allocated in stacks. The majority of the time, it's employed for functions.



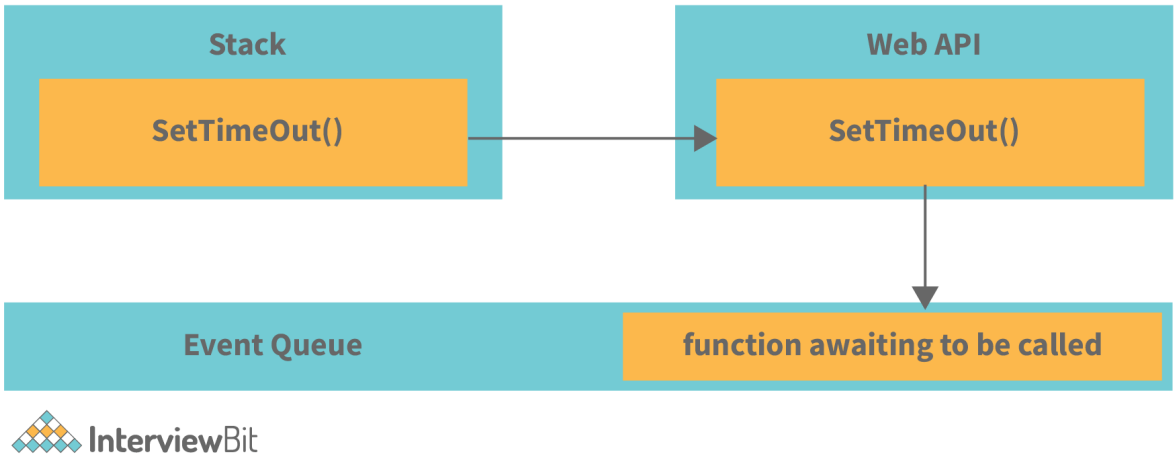
The function stack is a function that maintains track of all other functions that are running at the same time. An example to illustrate it is as follows:

```
function second() {
console.log("Second")
}
function First() {
second()
}
function foo() {
first()
}
foo()
```

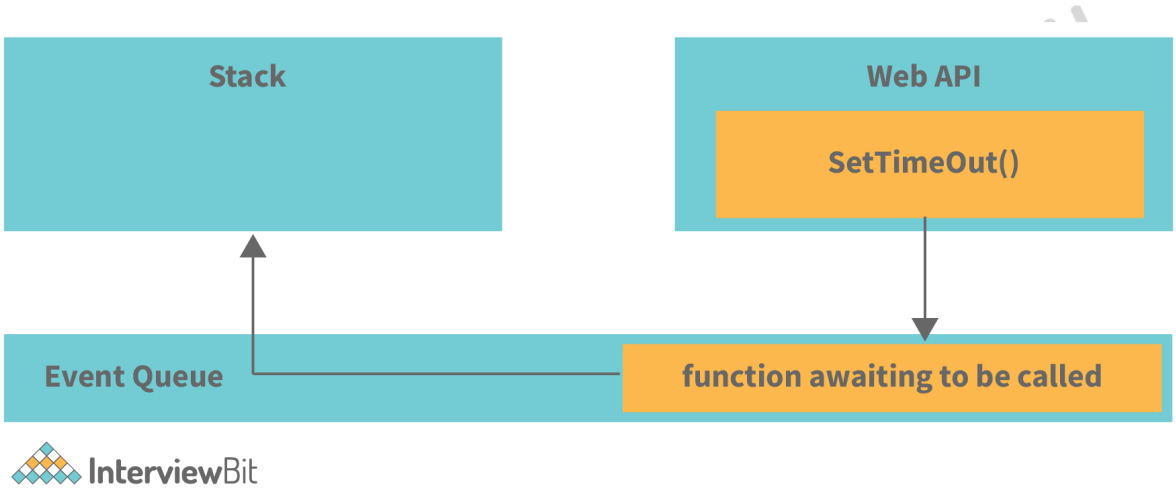
The order in which functions are executed, that is. when they are popped out of the stack once their purpose is completed, is as follows:

1. console.log
2. second
3. first
4. foo

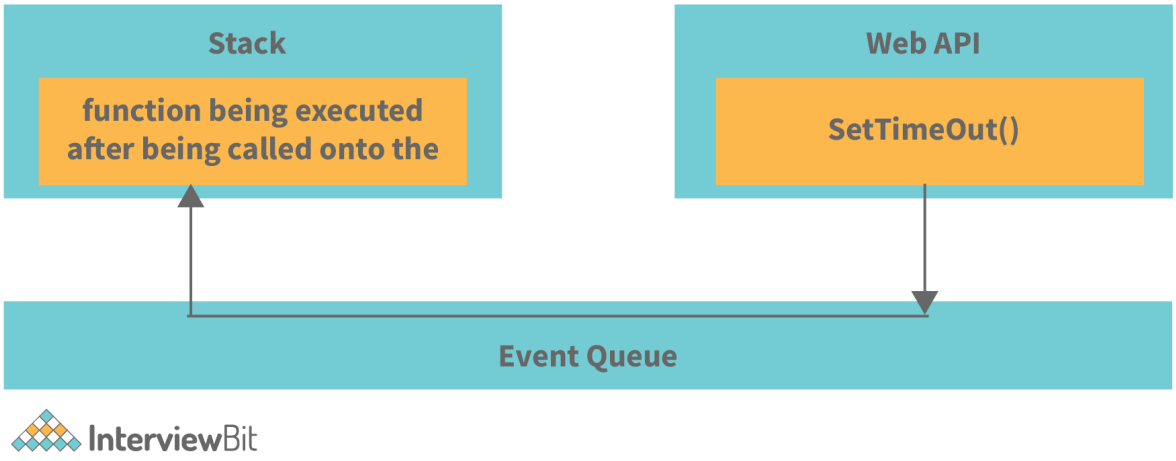
- **Event loop:** An event loop is something that pulls various things like methods, etc. out of the queue and places it onto the function execution stack whenever the function stack becomes empty. The event loop is the trick to making JavaScript appear multithreaded even if it is only single-threaded. The following illusion clearly explains how the event loop works:



The callback function in the event queue has not yet started and is waiting for its time to be added to the stack when SetTimeOut() is called and the Web API waits. The function is loaded onto the stack when the function stack becomes empty, as seen below:



The event loop is used to take the first event from the Event Queue and place it on the stack, which in this case is the callback function. If this function is called from here, it will call other functions within it.



22. JavaScript Error Handling

Various types of errors occur when we are coding in JavaScript. There are a few options for dealing with them:

- **try** — We can define a code block for testing errors using the try block.
- **catch** — We can set up a block of code to execute in the event of an error using the catch statement.
- **throw** — Instead of the typical JavaScript errors, we can also create custom error messages using the throw statement.
- **finally** — JavaScript also allows us to run our code regardless of the outcome of try and catch.

JavaScript possesses its own inbuilt error object which has the following properties:

- **name** — It is used for setting or returning an error name.
- **message** — It is used for setting or returning the error message as a string.

There are six types of ways in which the error property can return its name. They are as follows:

- **EvalError** — It indicates that an error has occurred within the eval() method.
- **RangeError** — It indicates that some number is “out of range”.
- **ReferenceError** — It indicates that an illegal reference was occurring.
- **SyntaxError** — It indicates that a syntax error was occurring.
- **TypeError** — It indicates that a type error was occurring.
- **URIError** — It indicates that an encodeURI() error was occurring.

23. ES5 Vs ES6: What is the Difference

ECMAScript 5 (ES5P): Because it was launched in 2009, ES5 is also known as ECMAScript 2009. It has functions where developers concentrate on how items are created. In ES5, you must use the function keyword and return to define the function, just as you would in any other JavaScript language.

ECMAScript 6 (ES6): Because it was launched in 2015, ES6 is also known as ECMAScript 2015. Its class allows developers to create an object with the new operator and an arrow function if they don't need to use the function keyword to specify the function, and they can also avoid using the return keyword to get the computer value.