Documentation

Duke Resources Home   Duke Course Home   Coursera Course Home

HTML

Full list of HTML elements

As a reminder, in most browsers you can right-click on a page and select *View Source* to see the HTML code used to render the page.

Basic HTML

Tag	Description	Example
`<html>`	All content of your webpage must go inside `<html></html>` tags.	
`<head>`	Contains information about the webpage. The title tag goes inside `<head></head>` tags.	
`<title>`	Title of the webpage (what appears in the window/tab of your browser). The text itself does not appear on webpage.	
`<body>`	Everything that appears on the webpage should go between these tags	
`<p>`	Defines a paragraph (text with some space on the bottom and top).	`<p>This is a paragraph.</p>` This is a paragraph.

`<h1>`	Heading tag, bold and bigger text.
	You can use any number from `<h1>` to `<h6>` with `<h1>` being the largest heading and `<h6>` being the smallest.
		`<h3>larger heading</h3>`
		`<h6>smaller heading</h6>`
		
		larger heading
		
		smaller heading

`<b>`	Apply bold formatting to text	`<b>bold</b>` **bold**
`<em>`	Apply emphasis to text	`<em>emphasis</em>` *emphasis*
`<img>`	Inserts an image. • `src` is the link specifying the image to display (it is a required attribute)	`<img src="http://bit.ly/1QfkvVw" width="100px">`

	• `width` (and `height`) specifies the size of the image (it is an optional attribute) Unlike most other tags, this start tag does not have a corresponding end tag.	
[`<a>`]()	Links to another webpage. • `href` specifies the URL of the page to link to (it is a required attribute). There *must* be some text between the start and end tags to be the anchor of the link.	`<a href="https://www.duke.edu/">Duke University</a>` [Duke University]()
[`<div>`]()	Defines a section of the web page.	`<div><p>This paragraph is inside a div.</p></div>`

Lists

Tag	Description	Example
[`<li>`]()	List item. List items can go inside unordered list, `<ul>`, or ordered list, `<ol>` tags.	`<li>HTML</li>`
[`<ul>`]()	Unordered list, each item has a bullet point.	`<ul>` `<li>HTML</li>` `<li>CSS</li>` `</ul>` • HTML • CSS
[`<ol>`]()	Ordered list, each item has a number.	`<ol>` `<li>HTML</li>` `<li>CSS</li>` `</ol>` 1. HTML 2. CSS

Tables

Tag	Description	Example
[`<table>`]()	Defines a table. By default a table has no borders and is only as wide as the text it contains.	
[`<tr>`]()	Defines a table row (only has value within `<table>` tag). Table rows can contain either table data elements or table header cells.	

<td>	Table data element (standard table cell). Can contain many types of data including text, images, links, lists, or even a table.	`<table> <tr> <td> cell 1 </td> <td> cell 2 </td> </tr> </table>` cell 1 cell 2
<th>	Table header cell (a table cell with bold text).	`<table> <tr> <th> heading </th> </tr> <tr> <td> content </td> </tr> </table>` **heading** content

Input

Because the attributes used with input elements varies so much depending on the type of input you want to use, we have provided several specific examples of using different types of input.

Example	Description
`<input type = "button" value = "change" onclick = "alert('clicked button')">`	• `type` is button • `value` is text that appears on button • `onclick` is event handler, specifies to call alert function when button is clicked
`<input type = "color" value = "#001A57" id = "clr" onchange = "docolor()">`	• `type` is color picker • `value` is default color value • `id` lets us refer to input element in JavaScript • `onchange` is event handler, specifies to call docolor function when color is changed
`<input type = "range" min = "10" max = "100" value = "10" id = "sldr"`	• `type` is slider • `min` is minimum value, `max` is maximum value

`oninput = "dosquare()">`	• `value` is default value • `id` lets us refer to input element in JavaScript • `oninput` is event handler, specifies to call dosquare function when slider is changed
`<input type = "text" id = "finput">`	• `type` is text • `id` lets us refer to input element in JavaScript
`<input type = "file" multiple = "false" accept = "image/*" id = "finput" onchange = "upload()">`	• `type` is file • `multiple = "false"` indicates user cannot select multiple files • `accept = "image/*"` indicates user can only select image files • `value` is default value • `id` lets us refer to input element in JavaScript • `onchange` is event handler, specifies to call upload function when input changes

CSS

Full list of CSS properties
Mozilla color picker tool

This website challenges people to use CSS to make as many different stylized versions as possible using the same HTML code.

Common CSS Properties and Values

Property	Example Values	Use with	Example
`color`	blue rgb(0,0,255) #0000FF	text: paragraphs, links, list elements, table cells, headings	`h1 { color: rgb(0,0,255); }`
`font-size`	12pt 16px 100%	text	`p { font-size: 14pt; }`
`text-align`	left right center justify	text	`td { text-align: center; }`

[background-color]()	blue rgb(0,0,255) #0000FF	table, table cell, page backgrounds	`body { background-color: #00FF00; }`
[vertical-align]()	top middle bottom	table cells	`th { vertical-align: top; }`
[float]()	left right	images	`img { float: right; }`
[width]()	100px	tables, table cells, images	`img { width: 80px; }`
[height]()	100px	tables, table cells, images	`td { height: 10px; }`
[border-width]()	5px	tables, table cells, images	`table { border-width: 2px; }`
[border-style]()	solid dotted dashed	tables, table cells, images	`table { border-style: solid; }`
[border-color]()	blue rgb(0,0,255) #0000FF	tables, table cells, images	`table { border-color: red; }`
[border]()	5px 10px dotted 5px dashed green	tables, table cells, images	`table { border: 2px solid red; }`
[border-collapse]()	collapse	table	`table { border-collapse: collapse; }`

Course Specific JavaScript Functions

SimplePixel

For these examples, assume

- `pix1` is a pixel at coordinate (100, 200) representing the color Duke blue, with RGBA values of (0, 26, 87, 255)
- `pix2` is a pixel at coordinate (300, 400) representing the color white, with RGBA values of (255, 255, 255, 255)

Function name	Description	Example
`getX()`	returns the pixel's x–coordinate within the image	`pix1.getX()` is 100
`getY()`	returns the pixel's y–coordinate within the	`pix1.getY()` is 200

	image	
`getRed()`	returns the value of the pixel's red component (always in the range 0–255)	`pix1.getRed()` is 0
`getGreen()`	returns the value of the pixel's green component (always in the range 0–255)	`pix1.getGreen()` is 26
`getBlue()`	returns the value of the pixel's blue component (always in the range 0–255)	`pix1.getBlue()` is 87
`getAlpha()`	returns the value of the pixel's alpha, or transparency, component (always in the range 0–255)	`pix1.getAlpha()` is 255
`setRed(newR)`	changes the value of the pixel's red component to `newR` (if `newR` is not in the range of 0–255 it is changed to be in that range)	`pix1.setRed(255)` changes the color to (255, 26, 87, 255)
`setGreen(newG)`	changes the value of the pixel's green component to `newG` (if `newG` is not in the range of 0–255 it is changed to be in that range)	`pix1.setGreen(255)` changes the color to (0, 255, 87, 255)
`setBlue(newB)`	changes the value of the pixel's blue component to `newB` (if `newB` is not in the range of 0–255 it is changed to be in that range)	`pix1.setBlue(255)` changes the color to (0, 26, 255, 255)
`setAlpha(newA)`	changes the value of the pixel's alpha, or transparency, component to `newA` (if `newA` is not in the range of 0–255 it is changed to be in that range)	`pix1.setAlpha(100)` changes the color to (0, 26, 87, 100)
`setAllFrom(otherPixel)`	changes the value of all of the pixel's components (its red, green, blue, and alpha) to match `otherPixel`'s values	`pix2.setAllFrom(pix1)` changes the color of pix2 to (0, 26, 87, 255)

SimpleImage

For these examples, assume the variable `logo` has the value of the image "devil.png" below. It is 100 pixels wide and 85 pixels tall.

Duke Blue Devil

Function name	Description	Example
`new SimpleImage(filename)`	creates a	`new SimpleImage("devil.png")` is

	`SimpleImage` to represent the image in `filename`	
`new SimpleImage(width, height)`	creates a `SimpleImage` whose dimensions are `width` by `height`. All the pixels in this image are black (0, 0, 0, 255)	`new SimpleImage(100, 100)` is
`new SimpleImage(fileInputElement)`	creates a `SimpleImage` to represent the image selected by the user using the `fileInputElement` given from the web page	`var input = document.getElementById("fileLoader");` `var img = new SimpleImage(input);` is assuming the user selected that image from their computer.
`getWidth()`	returns the image's width, or number of pixels in the X direction	`logo.getWidth()` is 100
`getHeight()`	returns the image's height, or number of pixels in the Y direction	`logo.getHeight()` is 85
`getPixel(x,y)`	returns the pixel in this image at the coordinate (`x`, `y`)	`logo.getPixel(0, 0)` is the pixel (255, 255, 255, 255)
`setPixel(x,y,pixel)`	copies the RGBA values from the given pixel into pixel at the (x,y) coordinates given	`logo.setPixel(50, 42, pix2)` changes the color to white
`setSize(width, height)`	resizes the image to be	`logo.setSize(300, 85)` is

	`width` by `height`. The image is scaled to fit into the new dimensions.	

`values()`	returns all the pixels in the image, starting in the upper-left corner and moving down to the lower-right corner, providing a way to access each pixel in turn	`for (var pixel of logo.values()) {`
		`// modify pixel`
		`}`

`drawTo(canvas)`	draws the image to canvas, for drawing images on web pages	`var canvas = document.getElementById("canvas");`
		`logo.drawTo(canvas);`

Printing

Function name	Description	Example
`print(something)`	displays `something` in the main "See It" area of the page	`print(image)` shows the image
`debug(something)`	displays `something` in the small area at the bottom of the "See It" area of the page	`debug(x)` shows the value of the variable `x`

Standard JavaScript

Arithmetic Operations

Operator	Description	Example
`+`	addition	`4 + 5` is 9
`-`	subtraction	`9 - 5` is 4
`*`	multiplication	`3 * 5` is 15
`/`	division	`6 / 3` is 2 `6 / 4` is 1.5
`%`	mod/remainder	`5 % 3` is 2

Comparing Two Numbers

Operator	Description	Example
`==`	is equal to	`3 == 3` is true
`!=`	is not equal to	`3 != 3` is false
`>=`	is greater than or equal to	`4 >= 3` is true
`<=`	is less than or equal to	`4 <= 3` is false
`>`	is strictly greater than	`4 > 3` is true
`<`	is strictly less than	`3 < 3` is false

Combining Comparisons – Logic Operators

For these examples, assume the variable `x` has the value 5.

Operator	Description	Example
`||`	returns true if at least one part of the comparison is true	`(x < 3 || x > 7)` is false `(x < 3 || x < 7)` is true
`&&`	returns true only if both parts of the comparison are true	`(x > 3 && x < 7)` is true `(x > 3 && x > 7)` is false
`!`	reverses the boolean value of a comparison	`(! x == 5)` is false

Math Functions

Function name	Description	Example
`abs(x)`	returns absolute value of `x`	`Math.abs(-3.6)` is 3.6 `Math.abs(3.2)` is 3.2
`ceil(x)`	returns `x`, rounded upwards to the nearest integer	`Math.ceil(3.6)` is 4 `Math.ceil(3.2)` is 4
`floor(x)`	returns `x`, rounded downwards to the nearest integer	`Math.floor(3.6)` is 3 `Math.floor(3.2)` is 3
`round(x)`	returns `x`, rounded x to the nearest integer	`Math.round(3.6)` is 4 `Math.round(3.2)` is 3

Random Functions

Function name	Description	Example

`random()`	returns a random number between 0 and 1	`Math.random()` might return 0.523 `Math.random()` might return 0.983

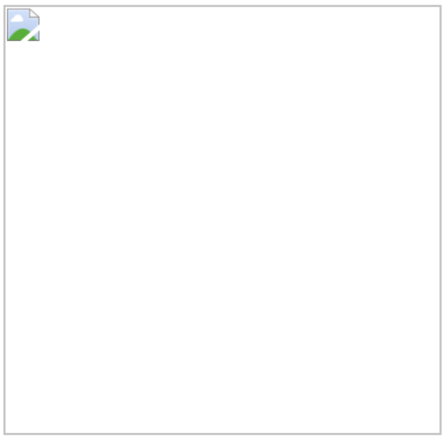
Background Information

What is a Pixel?

Digital images are made of pixels. A *pixel* is a small point of colored light. When you look at a computer monitor, the image you see is actually made of a grid of these tiny dots of light. They are so small and so close together that it looks like one continuous picture. To get an idea of how small a pixel is, the monitor that I happen to be using as I write this has a *resolution* of 1440 x 900 (read as "1440 by 900"). That means that there are 1,440 pixels across the top and 900 pixels down one side, for a total of almost 1.3 million pixels.

This is what pixels might look like if they were magnified. This is an example of a 5x4 image because it is 5 pixels wide and 4 pixels tall.

Each pixel has a color value. We need a way to represent colors so that we can tell the computer which color to make each pixel. There are many color representations, but JavaScript uses a scheme called the *RGBA color model*. Basically, it means that a color is represented by four numbers:



- R (the amount of red light)
- G (the amount of green light)
- B (the amount of blue light) and
- A (called "alpha", this number tells how transparent the color should be)

So each color is represented by these four numbers ("Everything's a number", remember?). Moreover, **each of these number slots must have a value between 0 and 255**. You may be wondering whether or not only 256 possibilities for each slot is enough to make all the many colors that we might want. If you have 256 possibilities for each of the R, G, and B values, (ignoring the transparency number for now) then the total number of colors available is over 16 million! It is estimated that the human eye can only detect 10 million different colors, so there's no worry that you won't be able to make any color you want.

Since the computer uses light to make a picture, the RGBA model is an *additive color model*. This means that the more the medium is added, the closer the color gets to white. Contrast this with using paint as a medium. That is a *subtractive color model* because the more paint you add, the further you get from white. So it should come as no surprise then that a color with R, G, and B values all 0 is black (no light) and a color with R, G, B values all 255 is white (pure light). Think of these numbers as knobs that you can turn up or down. If you turn on the red and blue lights and leave off the green light, you have shades of purple (R = 150, G = 0, B = 150, for example). The more you turn up the light, the higher the number goes and the brighter the color gets. How will you know what values to use for R, G, and B to make the color you want? If you search for "RGB color chart" on the Internet you will find lots of sites with palettes of colors and their corresponding values.

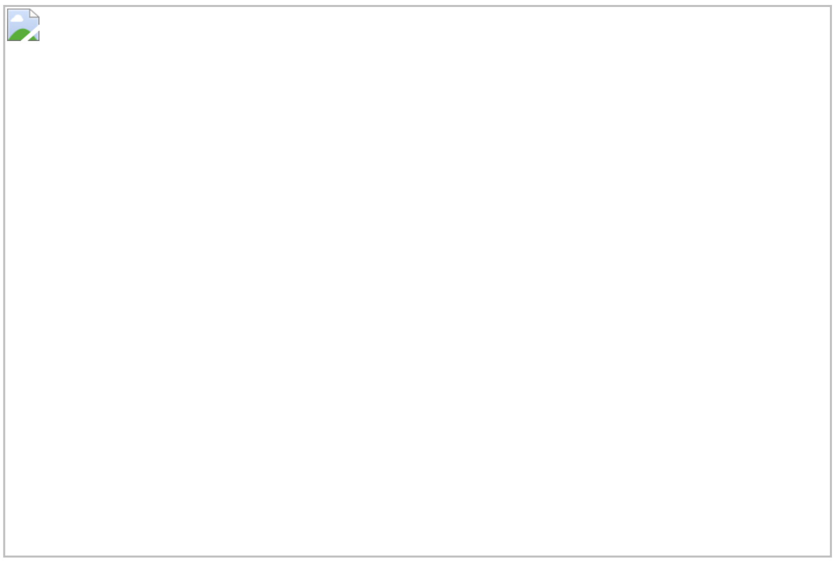
Transparency: Alpha Channel

The last value, called alpha, is also a number whose value must be between 0 and 255. This time the value of the number does not change the color's hue, but rather the transparency of the color. If a pixel has an alpha value of 0, it is completely transparent, or invisible. If it has a value of 255, it is completely opaque. Using this transparency value allows you to have layers of color on the screen, or shapes that can be partially seen through other shapes.

Image Coordinate System

Finally, it is important to be able to distinguish one pixel from another. We do this by referring to each pixel's location on the screen or image. Each pixel gets an *X and Y value*, where (0,0) is the top left corner of the screen. (This can take some getting used to as it's not the way a Cartesian Plane is drawn in math class!)

The X value refers to how far right the pixel is, and the Y value refers to how far down the pixel is. For my computer monitor, which is 1440 x 900, the top left corner is (0,0), the top right is (1439,0), the bottom left is (0,899), the bottom right is (1439,899), and the middle is (720, 450).



All of the discussion so far has been generic knowledge of a computer representation of images and colors. In other words, the same information would probably apply if you were programming in a language other than JavaScript. The rest of this page gives you details that are specific to JavaScript and the problems you are asked to solve.

HTML

- Basic HTML
- Lists
- Tables
- Input

CSS

- CSS Properties and Values

Course Specific Functions

- SimplePixel
- SimpleImage
- Printing

Standard JavaScript

- Arithmetic Operations
- Comparing Two Numbers
- Combining Comparisons
- Math Functions
- Random Functions

Background Information

- What is a Pixel?
- Transparency: Alpha Channel
- Image Coordinate System