HTML

uses <tags>

**elements**

<h1>Highest Headings </h1>

<h6>Lowest Headings </h6>

<p> paragraph elements </p>

<**body**>body element <body> // this element covers the entire screen/background, can be styled in CSS

//if you give your body element font styles or colour, then they will be **inherited** by your other elements unless over-rided

<!-- Comment

-->

<ul>

<li>unordered list 1</li>

<li>unordered list 2</li>

</ul>

<od>

<li>ordered list 1</li>

<li>ordered list 2</li>

</od>

<input type=”text”> //**text field**

<input type=”text” placeholder=”placeholder text”>

<**form** action=”/URL-where-you-send-form-data”></form>

<form action=”/URL”><input type=”text” **required** placeholder=”text”</form> // this blocks from submitting until form is filled out

**buttons**

<button type=”submit”>submit</button> //submit button used with form

<**label**><input type=”**radio**” name=”name of radio buttons”>button1</label>

<**label**><input type=”**checkbox**” name=”name of checkboxes” **checked**> checkbox1 </label> //checked is checked by default

adding **properties**

<h1 **style**=”color:red”>inline style</h1>

<h2 **class**=”blue-class”>

<h2 **id** = “element id”> // can be used as handles

<h3 style = “**border-color**:red**”**>element</h3>

<h3 style = “**border-style**:solid**”**>element</h3>

<h3 style = “**border-width**:5px**”**>element</h3>

<h3 style = “**border-radius**:10px”>element</h3>

<h3 style = “**border-radius**:50%”>element</h3>

**importing font-families**

//place at top of page

<link href=”<https://fonts.googleapis.com/css?family=Lobster>” rel=”stylesheet” type=”text/css”>

**images**

<img src=”https://www.imagesource.com/image.jpg”>

resizing images <img src=”URL” style:500px> //can be done in CSS

<figcaption>text underneath image</figcaptionj>

**anchoring**

<a href=”URL”>link’s text</a>

<a href=”URL”><img src=”https://www.can\_embed\_image.com/image.jpg”>

resizing images <img src=”URL” style:500px></a>

<a href=”#”>dead link</a>

**div element containers**

<div>elements</div> // can use this to add properties to group of elements

**margin**

margin: 20px //amount of space between element border and surrounding elements

margin-bottom: -2px;

margin-top:

margin-left:

margin-right:

**padding**

padding: 10px; // amount of space between element and border

padding-top: 5px;

padding-right: 5px;

padding-bottom: 5px;

padding-left: 5px;

padding: 10px 20px 10px 20px; //top right bottom left

**Over-rides**

ids will always override styles

inline styles will override ids

!important in CSS will override all else

**Colors**

colours can be in Hex #000000;

there is abbreviated Hex code

rgb(0,0,0)

**Bootstrap – Responsive Design**

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css"/> // add to top of page

<div class=”container-fluid”> all other stuff so that browser can resize page to fit screen</div>

<img class=”img-responsive” [image source and other stuff]>

<h2 class=”text-center”> // centres text

<button class=”**btn**”>some text</button>

<button class=”btn **btn-block**”>fill entire screen</button>

<button class=”btn btn-block **btn-primary**”>gives it a great colour</button>

<button class=”btn btn-block **btn-info**”> gives a different secondary “info” colour</button>

<button class=”btn btn-block **btn-danger**”>gives a danger colour </button>

**Bootstrap Grid**

This grid helps organise the horizontal length of the screen into 12 units. You can assign elements size elements.

<div class=”col-xs-4”> //gives 4 size elements for extra small pages

<div class=”col-md-4”> // givesd 4 size elements for medium sized pages

<p>This helps change properties within<**span** class=”text-danger”>elements</span></p>

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css"/> //to include for font-awesome

**text input**’s can be given **form-control** classes which give it 100% width.

**icons**

<i class=”fa fa-info-circle”></i>

**row and wells**

<div class=”**row**”> //this creates columns

<div class=”col-xs-6”>

<div class=”**well**”>

//elements

</div>

</div>

<div class=”col-xs-6”>

<div class=”**well**” **id**=right-well> // can give ids to wells so that you can handle them

//elements

</div>

</div>

</div>

CSS Cascading Style Sheets

// place above HTML

<style>

h2 (color:red);

.blue-class {

color: blue;

**font-size**: 30px;

**font-family**: Sans-serif, Helvetica; //changes to Helvetica if Sans-serif is not present

}

.image-class{ // remember can put more than one style in element

border-color:red;

border-width:5px;

border-style:solid;

}

#image-id{

background-color:green;

}

</style>

jQuery and JavaScript

this gives functionality to webpages and allows you to run JavaScript

give elements class=”**target**”

<script>

$(document).ready(function(){

// code here runs as soon as page is open

$("button").addClass("animated bounce"); // can target all **elements** of a type

$(".well").addClass("animated shake"); // can target elements by **class**

$("#target3").addClass("animated fadeOut"); // can target elements by **id**

$("#target1").addClass("btn-primary"); // can target same element by different handle

$("#target1").**css**("color", "red"); // can change **CSS** of an element

$("#target1").**prop**("disabled", true); // can access property of element

$("#target4").**html**("<em>#target4</em>"); // can affect text in html, here *emphasis*

$("#target4").**remove**(); // can **remove** element completely

$("#target2").**appendTo**("#right-well"); // can move elements to different wells

$("#target5").**clone().appendTo**("#left-well"); // can **copy** element to different wells

$("#target1").**parent**().css("background-color", "red"); // cantarget element of **parent**

$("#right-well").**children**().css("color","orange"); // can target **children** of element

$(".target:nth-child(2)").addClass("animated bounce"); // target **specific children** of element

$(".target:even").addClass("animated shake"); // can target **odd** or **even** elements

$("body").addClass("animated hinge"); // can handle entire page

});

</script>

JavaScript

comments: // or /\* stuff \*/

data types:

*undefined, null, boolean, string, symbol, number,* and *object.*

Declaring variable requires *var* keyword:

var ourName; // have variable *undefined* if not given initial variable.

Variables and functions are case sensitive and variables are typically in camelCase.

*Numbers*

operations are same as other languages.

I++ = i+1;

numbers can be decimal points, no need to do anything different.

% remainder operator is not modulus, it does not work with negative numbers.

MyVar += is myVar = myVar + num;

myVar -= is myVar = myVar – num;

myVar \*= is myVar = myVar \* num;

myVar /= is myVar/num;

*Strings Literals*

“Strings can be added together.”

if you want to include “ or ‘ inside your string, you can put a backslash before like this \”.

*Escape Sequences in JS*

| Code | Output |
| --- | --- |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | newline |
| \r | carriage return |
| \t | tab |
| \b | backspace |
| \f | form feed |

.length will get the length of string literal.

Indexing of string is through [x]; // starts from 0

strings are *immutable* meaning that once defined, they cannot be changed.

Deep copy string.slice();

*Arrays*

var name = [“stuff”, “stuff” ] // can contain different types of variables. // also has zero-based indexing.

Nested arrays var name=[[“dog”, 1],[2,4]];

arrays are *mutable*.

**Push()** = Adding value to end of array

**Pop()** = removing value from end of an array;

**Shift()** = removing first value from array;

**unShift()** = adding first value to array

local vs global:

global variables are just variables defined outside functions.

Local variables are just variables defined inside functions. Local variables with the same name as global variables will override the global variable.

*functions*

function functionName(input variables){

return variables;

}

*arguments object*

you can get the arguments input into a function by going arguments[x].

*if statements*

if (condition){

statements;

}

wasThatTrue and isItTrue are boolean conditions uses in if statements.

*== Equality operator*

JavaScript can compare over two different data types such as 1 or ‘1’.

=== *Strict Equality Operator*

JavaScript has an operator that compares data types as well.

*!= inequality operator.*

*!== strict inequality operator.*

&& AND

|| OR

remember to string your if and else if statements from smallest to biggest to capture the best resolution!

*Switch statements*

switch (num){

case value1:

statement1;

break;

case value2:

statement2;

break;

…

case valueN:

statementN;

break;

default:

statement;

}

a cool trick using switch statements:

switch(val) {  
case 1:  
case 2:  
case 3:  
result = "1, 2, or 3";  
break;  
case 4:  
result = "4 alone";  
}

refactoring trick!

don’t need if or else statements if return [equality statement] does the same.

*Objects*

similar to arrays but instead of using indexes to access and modify data, you access data through *properties*.

*object layout*

var objectName ={

“prop1”: “stuff”,

“prop2”: 2,

“prop3”: 1,

“prop4”: [“Water”, “Dogs”]

};

accessing properties: *objectName.prop1*;

or *objectName[“prop1”]*;

Here is an example of using a variable to access a property:

var someProp = "propName";  
var myObj = {  
propName: "Some Value"  
}  
myObj[someProp]; // "Some Value"

Here is one more:

var myDog = "Hunter";  
var dogs = {  
Fido: "Mutt", Hunter: "Doberman", Snoopie: "Beagle"  
}  
var breed = dogs[myDog];  
console.log(breed);// "Doberman"

can **overwrite properties** by going *objectName.prop1 = “NEW THING”;*

can **add properties** by going *objectName.newprop = “NEW Thing!”*;

can **delete properties** by going *delete objectName.prop1*;

can **check if property exist** by.hasOwnProperty() returns true or false.

You can also **nest objects** together

creating objects using **constructor** functions:

var Car = function() {  
  this.wheels = 4;  
  this.engines = 1;  
  this.seats = 5;  
};

making **instances** of objects with constructor function:

var myCar = new Car();

another cool example:

// Setup

function phoneticLookup(val) {

var result = "";

// Only change code below this line

var lookup ={

alpha: "Adams",

bravo: "Boston",

charlie: "Chicago",

delta: "Denver",

echo: "Easy",

foxtrot: "Frank"

}

result = lookup[val];

// Only change code above this line

return result;

}

// Change this value to test

phoneticLookup("charlie");

[JavaScript Object Notation](http://www.json.org/) or JSON is a related data interchange format used to store data.

{  
"artist": "Daft Punk",  
"title": "Homework",  
"release\_year": 1997,  
"formats": [   
"CD",  
"Cassette",  
"LP"  
],  
"gold": true  
}

**Unique Objects**

var Car = function(wheels, seats, engines) {  
  this.wheels = wheels;  
  this.seats = seats;  
  this.engines = engines;  
};

Objects can have their ownfunctions called **methods**.

var Car = function() {

// this is a **private** variable

var speed = 10;

// these are **public** methods

this.accelerate = function(change) {

speed += change;

};

this.decelerate = function() {

speed -= 5;

};

this.getSpeed = function() {

return speed;

};

};

**map method**  is used to iterate through an array.

OldArray.map(function(val)){ //multiplies all values in array by 4

return val\*4;

}

**reduce method** is used to operate cumulatively on each value of array.

var singleVal = array.reduce(function(previousVal, currentVal) {  
  return previousVal - currentVal;  
}, 0);

**filter method** is used to filter out unwanted variables.

array = array.filter(function(val) {  
  return val !== 5;  
});

**sort method** is used to sort array, it changes the original array and typically you should do a compare function inside of it too so that it knows how you want to sort it.

var array = [1, 12, 21, 2];  
array.sort(function(a, b) {  
  return a - b;  
});

**reverse method** is a method that just reverses an array, it alters the original array.

var myArray = [1, 2, 3];  
myArray.reverse();

**concat method** is a method that joins an array to the end of the first array.

newArray = oldArray.concat(otherArray);

**split method** is a method to split strings into many strings and put them into an array.

Var array=string.split(‘s’);

**join method** is a method that takes an array of strings and then joins them together to form one string. You can place an argument between each string.

Var salad = veggies.join(“ and “);

*for loops*

for([initialization];[condition]; [final-expression]){

statements;

}

*while* loops

while(condition){

statements;

}

**Math**

Random Numbers Generator 0 to 1: *Math.random()*

floor fractions using: *Math.floor()*

console.log("Hi World"); // prints to console

**Finding Occurrences of Words in Strings**

Regular expressions are used to find certain words or patterns inside of strings.

For example, if we wanted to find the word the in the string The dog chased the cat, we could use the following regular expression: /the/gi

Let's break this down a bit:

/ is the start of the regular expression.

the is the pattern we want to match.

/ is the end of the regular expression.

g means global, which causes the pattern to return all matches in the string, not just the first one.

i means that we want to ignore the case (uppercase or lowercase) when searching for the pattern.

andCount = testString.match(expression).length;

for **numbers** use expression: var expression = /\d+/g; // the + symbol means look for more than one.

For **whitespace** use expression: var expression = /\s+/g;

For any **non-whitespace** use expression: var expression = /\S/g;