# JavaScript

## Basics

### Basic Structure

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Basic File </**title**>

<**meta** charset = "UTF-8">

<**script** src = "wakeup.js"></**script**>

</**head**>

<**body**></**body**>

</**html**>

setTimeout(wakeUpUser, 2000);

**function** wakeUpUser(){

alert("Get Started With Javascript");

}

**Code 1: wakeupuser.js**

### Document Write

This method writes HTML expressions or JavaScript code to a document.

**var** count = 4;

**for**(**var** i = 0; i < count; i++){

document.write(i + "<br/>");

}

**Code 2: documentwrite.js**

This method is mostly used for testing: If it is used after an HTML document is fully loaded, it will delete all existing HTML. This method should be heavily avoided

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p>My first paragraph.</p>

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

<script>

function myFunction() {

document.write("Hello World");

}

</script>

</body>

</html>

**Code 3: documentwriteavoid.html**

Open an output stream, add some text, then close the output stream:

<!DOCTYPE html>

<**html**>

<**body**>

<**p**>Click It.</**p**>

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

<**script**>

**function** myFunction() {

document.open();

document.write("<h1>Hello World</h1>");

document.close();

}

</**script**>

</**body**>

</**html**>

**Code 4: documentwriteoutput.html**

Open a new window called "MsgWindow", and write some text into it:

<!DOCTYPE html>

<**html**>

<**body**>

<**p**>Click Here</**p**>

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

<**script**>

**function** myFunction() {

**var** myWindow = window.open("", "MsgWindow", "width=200,height=100");

myWindow.document.write("<p>This is 'MsgWindow'. I am 200px wide and 100px tall!</p>");

}

</**script**>

</**body**>

</**html**>

**Code 5: documentwritewindow.html**

### Console Log

The console.log() method writes a message to the console.

The console is useful for testing purposes.

### Debugger

The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function.

This has the same function as setting a breakpoint in the debugger.

If no debugging is available, the debugger statement has no effect.

**var** a = 3;

**var** b = 5;

console.log(a);

console.log(b);

**debugger**;

console.log(a + b);

**Code 6: console.js**

### Time Out

This setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds. Therefore, 1000 ms = 1 second. This function is only executed once.

Use the clearTimeout() method to prevent the function from running.

setTimeout(**function**(){

alert("Wake up!")

}, 3000);

**Code 7: timeout.js**

### Prompt

The prompt() method displays a dialog box that prompts the visitor for input.

**var** getInput = prompt("Give me your input bro!");

console.log(getInput);

**Code 8: prompt.js**

## Math

*/\*\**

*\* Give any decimal value between 0 to 1*

*\*/*

**var** random = Math.random();

console.log(random);

*/\*\**

*\* Give any decimal value between 0 to 5*

*\*/*

**var** random\_five = (5 \* Math.random());

console.log(random\_five);

*/\*\**

*\* Math Ceil*

*\*/*

console.log(Math.ceil(random\_five));

*/\*\**

*\* Math Floor*

*\*/*

console.log(Math.floor(random\_five));

*/\*\**

*\* Math Square Root*

*\*/*

**var** number = 9;

console.log(Math.sqrt(number));

*/\*\**

*\* Math PI*

*\*/*

radius = 3;

console.log("Area of the Circle: " + (Math.PI \* Math.pow(3,2)))

**Code 9: math.js**

## Function

Key Point: If we forget to declare a variable before using it, the variable will always be global (even if first time we use it is in a function.

radius = prompt("Please enter the radius of the Circle!");

**function** AreaOfCircle(radius){

**var** area = Math.PI \* Math.pow(radius, 2);

**return** area;

}

console.log(AreaOfCircle(radius));

**Code 10: function.js**

## Array

**var** fastfood = ["Pizza", "Burger", "Sandwich"];

**var** drinks = [];

*/\*\**

*\* Finding the number of items in an array*

*\*/*

console.log(fastfood.length);

*/\*\**

*\* For every fast food we should have a drink*

*\*/*

**for**(**var** i = 0; i < fastfood.length; i++){

**switch**(i){

**case** 0:

drinks.push("lemonade");

**break**;

**case** 1:

drinks.push("coke");

**break**;

**case** 2:

drinks.push("sprite");

**break**;

**default**: console.log("No Drink Bro!");

}

}

*/\*\**

*\* It's in array format*

*\*/*

console.log(drinks);

*/\*\**

*\* For each to convert to String*

*\*/*

drinks.forEach(**function**($item, $index){

console.log($item);

});

**Code 11: array.js**

### Sort

This method sorts the items of an array.

The sort order can be either alphabetic or numeric, and either ascending (up) or descending (down).

By default, this method sorts the values as strings in alphabetical and ascending order.

**compareFunction(Optional)**

A function that defines an alternative sort order. The function should return a negative, zero, or positive value, depending on the arguments

function(a, b){return a-b}

When the sort() method compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

**function** init(){

**var** numbers = [60, 50, 62, 58, 54, 54];

numbers.sort(compareNumbers);

console.log(numbers);

}

**function** compareNumbers(num1, num2){

**if**(num1 > num2){

**return** 1;

}**else** **if**(num1 === num2){

**return** 0;

}**else**{

**return** -1;

}

}

window.onload = init;

**Code 12: sort.js**

## Simple Object

**var** fiat = {

make: "Fiat",

model: "500",

year: 1957,

color: "Blue",

passengers: 2,

convertible: **false**,

mileage: 88000,

drive: **function**(){

console.log(**this**.make + " Running: Zoom Zoom");

}

};

fiat.drive();

**Code 13: object.js**

|  |  |
| --- | --- |
| **Object provided by JavaScript** | **Objects provided by the web browser** |
| Date 🡪 Manipulate Date, Time | Document: Manipulate HTML |
| Math 🡪 Math Functions | Window: Browser Related Properties |
| RegExp 🡪 Patterns in Strings | Console: Debugging/Display Messages |
| JSON 🡪 Exchange JS Objects with apps |  |

## Document Object Model

### Get Element By ID

🡪document.getElementById(“ “)

🡪[document].innerHTML

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Basic File </**title**>

<**meta** charset = "UTF-8">

<**script** src = "document.js"></**script**>

</**head**>

<**body**>

<**h1** id = "text">What's up bro</**h1**>

<**button** onclick = "changeMe()">Change</**button**>

</**body**>

</**html**>

**function** changeMe(){

**var** text = document.getElementById("text");

text.innerHTML = "Nothing Much!";

}

**Code 14: document.js**

### Set Attribute

🡪document.setAttribute(“class”, “something”)

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Basic File </**title**>

<**meta** charset = "UTF-8">

<**script** src = "attribute.js"></**script**>

<**style**>**.modify**{**color**: rgba(255, 121, 123, 0.4);}</**style**>

</**head**>

<**body**>

<**h1** id = "text">What's up bro</**h1**>

<**button** onclick = "changeMe()">Change</**button**>

</**body**>

</**html**>

**function** changeMe(){

**var** text = document.getElementById("text");

text.innerHTML = "Nothing Much!";

text.setAttribute("class", "modify");

}

**Code 15: attribute.js**

To make sure the JavaScript is not running before the page loads we will write,

function init() {} then assign it to window.onload = init;

### Get Element By Tag Name

document.getElementsByTagName(“”)

This selects all the tags and will get back all elements of that tag name

### Get Element By Class Name

document.getElementsByClassName(“”)

Pass this method the name of a class, we will get back all elements that have that class, as a Node List

### Get Element By Name

document.getElementsByName(“”)

This method retrieves elements that have a name attribute with a value that matches the name we pass it

### Get Element By Query Selector

document.querySelector(“”)

This method takes a selector(just like a CSS selector) and returns the first element that matches.

### Get Element By Query Selector All

document.querySelectorAll(“”)

This method also takes a selector, but returns all the elements that match, as a Node List

## Primitive Types

### Undefined

It is similar to [isset() from PHP] where it **tests** whether to see if **a variable (or property, or array item) has been given a value.**

### Null

It is intended to represent an **object** that isn’t there. It may just mean that **object doesn’t exist** yet and needs to be created, or object doesn’t exist and we can skip it.

### NaN

**Not a Numbers, to represent numeric result**. Use the function isNan()

Ex: 0 divided by 0 will give NaN because this has no definite answer.

Ex: var b = “food” + 100

Ex: var c = Math.sqrt(-9)

## Strings

### Length

The length property holds the number of characters in the string. It’s quite handy for iterating through the characters of the string

var\_string.length

### Index Of

This method takes a string as an argument and returns the index of the first characters of the first occurrence of that argument in that string

var\_string.indexOf(string, starting\_point)

### Substring

Give the substring method two indices and it will extract and return the string contained within them.

var\_string.substring(starting\_point, ending\_point)

### Split

The split method takes a character that acts as a delimiter, and breaks the string into parts based on the delimiter.

var\_string.split(delimiter)

### Upper Case

Returns a string all lowercase character changed to uppercase characters

var\_string.toUpperCase()

### Trim

Removes all the white spaces in a string

var\_string.trim()

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Basic String Manipulation </**title**>

<**meta** charset = "UTF-8">

<**script**>

**function** init(){

*/\*\**

*\* Use Get Element By ID (Always to get value)*

*\*/*

**var** content = document.getElementById("patrick");

*/\*\**

*\* Use Text text content to retrieve the value*

*\*/*

**var** content = content.textContent;

*/\*\**

*\* Gives the length of the string*

*\*/*

**var** len = content.length;

*/\*\**

*\* Removes the white space*

*\*/*

**var** trimmed\_content = content.trim();

*/\*\**

*\* Upper Case All the string*

*\*/*

**var** upper\_content = trimmed\_content.toUpperCase();

console.log(content + ", length: " + len);

console.log("Trimmed Content: " + trimmed\_content + ", Upper Content: " + upper\_content);

*/\*\**

*\* Look for certain character*

*\*/*

**for**(**var** i = 0; i < len; i++){

**if**(content.charAt(i) == "S"){

console.log("S is for Superstar");

}

}

*/\*\**

*\* In which index does the star is found*

*\*/*

console.log(content.indexOf("Star"));

*/\*\* Gives the string at a given range*

*\*/*

**var** data = document.getElementById("data");

data = data.textContent;

**var** val = data.substring(6,9);

console.log(val);

*/\*\**

*\* Split the string*

*\*/*

data = data.split("|");

console.log(data);

combine = data.join(",")

console.log(combine);

}

window.onload = init;

</**script**>

</**head**>

<**body**>

<**h1** id = "patrick"> Patrick Star </**h1**>

<**p** id = "data">MySQL|PHP|JavaScript|HTML</**p**>

</**body**>

</**html**>

**Code 16: string.html**

### Lower Case

Returns a string all uppercase character changed to lowercase characters

var\_string.toLowerCase()

### Replace

Finds the substrings and replaces them with another string

var\_string.replace(delimiter, replace)

### Slice

Returns a new string that has part of the original string removed

var\_string.slice(start, end)

### Last Index Of

Just like indexOf(), but finds the last, not the first occurrence

lastIndexOf(string, start)

### Concatenation

Joins strings together

var\_string.concat(string1, string2, … )

### Match

Searches for matches in a string using regular expression. No Apostrophe inside the function. Returns null when no match found

var\_string.match(/string/modifier)

Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

|  |  |
| --- | --- |
| Modifier | Description |
| i | Perform case-insensitive matching |
| g | Perform a global match (find all matches rather than stopping after the first match) |
| m | Perform multiline matching |

Regular Expression Patterns

Brackets are used to find a range of characters:

|  |  |
| --- | --- |
| Expression | Description |
| [abc] | Find any of the characters between the brackets |
| [0-9] | Find any of the digits between the brackets |
| (x|y) | Find any of the alternatives separated with | |

Metacharacters are characters with a special meaning

|  |  |
| --- | --- |
| Metacharacter | Description |
| \d | Find a digit |
| \s | Find a whitespace character |
| \b | Find a match at the beginning or at the end of a word |

Quantifiers define quantities:

|  |  |
| --- | --- |
| Quantifier | Description |
| n+ | Matches any string that contains at least one n |
| n\* | Matches any string that contains zero or more occurrences of n |
| n? | Matches any string that contains zero or one occurrences of n |

Operators

|  |  |
| --- | --- |
| = | Assigns a value to a variable |
| == | Compares Object reference, Check values to see if equal |
| === | Compares values and types to see if equal |

<**script**>

**var** name = "Cupertino Spaniola";

*/\*\**

*\* Lower Case*

*\*/*

console.log(name.toLowerCase());

*/\*\**

*\* Replace*

*\*/*

console.log(name.replace("Span", "Ran"));

*/\*\**

*\* Slice*

*\*/*

console.log(name.slice(4, 6));

*/\*\**

*\* Match*

*\*/*

**var** name2 = "pan.peteroto";

**if**(name2.match(/oto/gi)){

console.log("Found it!");

}

**else**{

console.log("Where is he?");

}

*/\*\**

*\* Test*

*\*/*

**var** email = "peter.pan@gmail.com";

**var** regex = /[A-Za-z.\_]n?@[A-Za-z]n?.[a-z]n?[a-z]n?[a-z]n?/;

**if**(regex.test(email)){

console.log("Valid Email Address");

}

**else**{

console.log("Invalid Email Address");

}

</**script**>

**Code 17: string2.html**

## Handling Events

Whenever there’s an event, there is an opportunity for our code to handle it.

### On Click

The onclick event occurs when the user clicks on an element.

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>Event Handler</**title**>

<**meta** charset = "UTF-8"/>

<**script**>

**function** init(){

**var** event = document.getElementById("handler");

event.onclick = handleEventHandler;

**var** event\_form = document.getElementById("submit");

event\_form.onclick = handleEventForm;

}

**function** handleEventHandler(){

**var** handler = document.getElementById("handler");

handler.innerHTML = "Fire";

}

**function** handleEventForm(){

**var** fullname = document.getElementById("fullname");

**var** value = fullname.value;

console.log(value);

}

window.onload = init;

</**script**>

</**head**>

<**body**>

<**h1** id = "handler">Event Handler</**h1**>

<**input** type = "text" name = "text" id = "fullname"/><**br**/>

<**button** id = "submit">Confirm</**button**><**br**/><**br**/>

</**body**>

</**html**>

**Code 18: eventhandler.html**

### Handle Key Press

This event occurs when the user presses a key (on the keyboard).

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>Handle Key Press</**title**>

<**meta** charset = "UTF-8">

<**script**>

**function** init(){

**var** element = document.getElementById("clickme");

element.onkeypress = handleevent;

}

**function** handleevent(){

element = document.getElementById("fire");

element.innerHTML = "Shoot";

}

window.onload = init;

</**script**>

</**head**>

<**body**>

<**input** type = "submit" id = "clickme" name = "submit"/>

<**h1** id = "fire">Fire</**h1**>

</**body**>

</**html**>

**Code 19: handlekeypress.html**

**Selecting Tags (Multiple)**

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>Tag Name</**title**>

<**meta** charset = "UTF-8">

</**head**>

<**script**>

**function** init(){

**var** superheroes = document.getElementsByTagName("p");

**for**(**var** i = 0; i < superheroes.length; i++){

superheroes[i].onclick = eventHandler;

}

}

**function** eventHandler(){

**var** text = document.getElementById("fire");

text.innerHTML = "Boom!";

}

window.onload = init;

</**script**>

<**body**>

<**h1** id = "fire"></**h1**>

<**p**> Superman </**p**>

<**p**> Batman </**p**>

<**p**> Spiderman </**p**>

</**body**>

</**html**>

**Code 20: tagname.html**

### Event Object

function eventHandler(eventObject){

var target = eventObject.target;

}

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>Event Object</**title**>

<**meta** charset = "UTF-8"/>

<**script**>

**function** init(){

**var** words = document.getElementsByTagName("p");

**for**(**var** i = 0; i < words.length; i++){

words[i].onclick = eventHandler;

}

}

**function** eventHandler(eventObject){

**var** word = eventObject.target;

**var** id = word.id

**if**(word.hasAttribute("class")){

word.removeAttribute("class");

}

**else**{

word.setAttribute("class", "color");

}

console.log(id);

}

window.onload = init;

</**script**>

<**style**>

**.color**{

**color**: rgba(120, 120, 110, 0.7);

}

</**style**>

</**head**>

<**body**>

<**p** id = "zero">Zero</**p**>

<**p** id = "one">One</**p**>

<**p** id = "two">Two</**p**>

</**body**>

</**html**>

**Code 21: eventobject.html**

### Click

Get tis even when we click (or tap in the page)

### Load

The event we get when the browser has completed

### Mouse Move

When we move our mouse over an element

### Key Press

The event is generated every time we press a key

### Unload

This event is generated when we close the browser window, or navigate away from a web page.

### Mouse Over

When we put our mouse over an element, we will generate this event

### Mouse Out

We will generate this event when we move our mouse off an event

### Resize

When we resize our browser window, this event is generated

### Drag Start

If we drag an element in the page, we will generate this event

### Touch Start

On touch devices, we will generate a touch start event when we touch and hold an element

### Play

We will get this event when we will play <audio> or <video> play button

### Pause

We will get this event when we will pause <audio> or <video> play button

### Drop

We will get this event when we drop an element we have been dragging

### Touch End

We will get this even when we stop touching

### Set Time Out

setTimeout(function(){ alert("Hello"); }, 3000);

This setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds. Therefore, 1000 ms = 1 second. This function is only executed once.

Use the clearTimeout() method to prevent the function from running.

### Set Interval

This calls a function or evaluates an expression at specified intervals (in milliseconds).

This method will continue calling the function until clearInterval() is called, or the window is closed.

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Local Time </**title**>

<**meta** charset = "UTF-8">

<**script**>

**function** init(){

setInterval(update,1000);

}

**function** update(){

**var** showtime = document.getElementById("date");

**var** date = **new** Date();

**var** localTime = date.toLocaleTimeString();

showtime.innerHTML = localTime;

}

window.onload = init;

</**script**>

<**style**>

</**style**>

</**head**>

<**body**>

<**p** id = "date"><**date**>Show time</**date**></**p**>

</**body**>

</**html**>

**Code 22: setinterval.html**

## First Class Functions

Start thinking about function as values, jut like numbers, strings, booleans, or objects. The ting that really makes a function value different from these other values is that we can invoke it.

**Steps**

* Assign the values to a variable (or store it in a data structure like an array or object)
* Pass the value to a function
* Return the value from a function

**First Class:** A value that can be treated like any other value in a programming language, including the ability to be assigned to a variable, passed as an argument, and returned from a function.

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> First Class Functions </**title**>

<**meta** charset = "UTF-8">

<**script**>

**function** init(){

**var** telephone = callMe;

**var** answer = telephone("Peter");

console.log(answer);

}

**function** callMe(someone){

**return** (someone + " called me!");

}

window.onload = init;

</**script**>

</**head**>

<**body**>

</**body**>

</**html**>

**Code 23: firstclass.html**

### Nesting Function [Anonymous Function]

Used the anonymous function concept which is

setTimeout(function(){

alert(“Anonymous function!”);

},1000);

The precedence at which the compiler starts to read a code, the compiler reads the functions first then the variables.

**function** init(){

**var** fly = **function**(num){

**var** sound = "Flying Sound";

**function** wingFlapper(){

console.log(sound);

}

**for**(**var** i = 0; i < num; i++){

wingFlapper();

}

};

**function** quack(num){

**var** sound = "Quack";

**var** quacker = **function**(){

console.log(sound);

}

**for**(**var** i = 0; i < num; i++){

quacker();

}

}

quack(4);

fly(4);

}

window.onload = init;

**Code 24: nestinganonymous.js**

### Lexical Scope

Lexical just means we can determine the scope of a variable by reading the structure of the code, as opposed to waiting until the code runs to figure it out.

All local variables are stored in an environment. Java script functions are always evaluated in the same scope environment in which they were defined. Within a function, if we want to determine where a variable is coming from, search in its enclosing functions, from the most nested to the least.

### Closure

A closure is a function together with a referencing environment. If a variable in the function body isn’t defined locally, and it’s not a global, we can bet it’s from a function that it is nested in, and available in environment. A closure results when we combine a function that has free variables with an environment that provides variable bindings for all those free variables.

Therefore, for global variables we do not need to declare it is global inside the function because the variable is present in the environment.

**var** count = 0;

**function** init(){

console.log(counter());

console.log(counter());

console.log(counter());

}

**function** counter(){

count += 1;

**return** count;

}

window.onload = init;

**Code 25: closure.js**

Another way to create a closure is to pass a function to a function.

**function** init(){

makeTimer("Cooking Time Over!", 3000);

}

**function** makeTimer(message, time){

setTimeout(**function**(){alert(message)}, time);

}

window.onload = init();

**Code 26: closure2.js**

## Advanced Object Construction

Object constructors and functions are closely related. (Only in JavaScript)

**function** init(){

**var** fluffy = **new** Dog("Fluffy", "Poodle", 30);

**var** fido = **new** Dog("Fido", "Mixed", 38);

**var** spot = **new** Dog("Spot", "Chihuahua", 10);

**var** dogs = [fluffy, fido, spot];

dogs.forEach(**function**(dog){

console.log("Dog Name: " + dog.name);

console.log("Dog Breed: " + dog.breed);

console.log("Dog Weight: " + dog.weight);

console.log("==========================");

});

}

**function** Dog(name, breed, weight){

**this**.name = name;

**this**.breed = breed;

**this**.weight = weight;

}

window.onload = init;

**Code 26: basicobject.js**

Even constructed objects can have their own independent property

*/\*\* Insert Many type of same object into another \*/*

*/\*\**

*\**

*\*/*

**function** init(){

*/\*\**

*\* Created New Modded Dogs*

*\*/*

**var** fluffy\_mod = **new** Dog(fluffy);

**var** fido\_mod = **new** Dog(fido);

**var** spot\_mod = **new** Dog(spot);

*/\*\**

*\* Modded Dogs Array to see the results*

*\*/*

**var** mod\_dogs = [fluffy\_mod, fido\_mod, spot\_mod];

mod\_dogs.forEach(**function**(mod\_dog){

console.log("Dog Name: " + mod\_dog.name);

console.log("Dog Breed: " + mod\_dog.type);

console.log("Dog Weight: " + mod\_dog.weight);

console.log("Dog Running: " + mod\_dog.run);

console.log("==========================");

});

*/\*\**

*\* Constructed Objects with their individual property*

*\*/*

fido\_mod.owner = "Bob";

**delete** fido\_mod.weight;

**for**(**var** key **in** fido\_mod){

console.log(fido\_mod[key]);

}

}

*/\*\**

*\* Individual Dogs Objects*

*\*/*

**var** fluffy = {

name :"Fluffy",

type :"Poodle",

weight: 30

};

*/\*\**

*\* Individual Dogs Objects*

*\*/*

**var** fido = {

name :"Fido",

type :"Mixed",

weight: 38

};

*/\*\**

*\* Individual Dogs Objects*

*\*/*

**var** spot = {

name :"Spot",

type :"Chihuahua",

weight: 10

};

*/\*\**

*\* Dog Object (Inheritance)*

*\*/*

**function** Dog(param){

**this**.name = param.name;

**this**.type = param.type;

**this**.weight = param.weight;

**this**.run = **this**.name + " running ... running!";

}

window.onload = init;

**Code 26: objectliterals.js**

### Object Inside Object

**function** init(){

**var** trump = **new** HumanResource(**new** Teacher("Donald Trump", 4000));

**for**(**var** property **in** trump){

console.log(trump[property]);

}

}

**function** HumanResource(employee){

**this**.name = employee.name;

**this**.salary = employee.salary + 2000;

**this**.fire = "I can fire you!";

}

**function** Teacher(name, salary){

**this**.name = name;

**this**.salary = salary;

}

window.onload = init

**Code 27: inheritance.js**

### Date Objects

**function** init(){

*/\*\**

*\* Create Date Object*

*\*/*

**var** now = **new** Date();

console.log(now.toString());

console.log(now.getFullYear());

console.log(now.getDay());

}

window.onload = init;

**Code 27: others.js**

## Array Objects

How modulo division work

27 / 16 = 1, remainder 11

=> 27 mod 16 = 11

30 / 3 = 10, remainder 0

=> 30 mod 3 = 0

35 / 3 = 11, remainder 2

=> 35 mod 3 = 2

**[array].every()**

Every method takes a function and tests each value of the array to see if the function returns true or false when called on that value. If the function returns true for all the array items, then the result of every method is true.

**function** init(){

**var** array = **new** Array();

array.push(53, 234, 1234, 10, 1, 90 , 52, 92);

*/\*\**

*\* Normal Array*

*\*/*

console.log(array);

*/\*\**

*\* Reversed Array*

*\*/*

array.reverse();

console.log(array);

**var** findOdds = array.every(**function**(number){

**return** ((number%2) !== 0)

});

console.log(findOdds);

}

window.onload = init;

**Code 28: array.js**

## Prototypes

They are similar to abstract or interface. JavaScript object can inherit properties and behavior from other objects. JavaScript uses what is known as prototypal inheritance, and the object we are inheriting behavior from is called the prototype.

**function** init(){

**var** fido = **new** Dog("Fido", "Mixed", 38);

**var** fluffy = **new** Dog("Fluffy", "Poodle", 30);

**var** spot = **new** Dog("Spot", "Chihuahua", 10);

Dog.prototype.run = **function**(){

console.log("Run!");

};

console.log("Name: " + fido.name);

console.log("Breed: " + fido.breed);

console.log("Weight: " + fido.weight);

console.log("Run: " + fido.run);

}

**function** Dog(name, breed, weight){

**this**.name = name;

**this**.breed = breed;

**this**.weight = weight;

}

window.onload = init;

**Code 28: prototypes.js**

### Arguments

**function** printArgs(){

**for**(**var** i = 0; i < arguments.length; i++){

console.log(arguments[i]);

}

}

printArgs("sshh", "pssss", "isss");

**Code 28: arguments.js**

## Handling Operations

try{

}

catch(error){

console.log(“Error! ” + error.message);

}

## Add Event Listener

window.addEventListener(“load”, init, false)

window.onload = function(){

var div = document.getElementById(“clickme”)

div.addEventListener(“click”,handleClick, false);

};

function handleClick(e){

var target = e.target;

alert(“You clicked on “ + target.id);

target.removeEventListener(“click”, handleClick, false);

}

## Recursion

Recursion is based on mathematics 🡪 an algorithm to compute the Fibonacci number series. The Fibonacci number series is 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

function Fibonacci(n){

if((n === 0) || (n === 1)){

return 1;

}

else{

return (fibonacci(n-1) + fibonacci(n-2));

}

for(var i = 0; i < 10; i++){

console.log(“The Fibonacci of “ + i + “is “ + Fibonacci(i));

}

## Server-Side JavaScript

http.createServer(function(request, response){

response.writeHead(200, {“Content-Type”: “text/plain”});

response.write(“Hello World”);

response.end();

}).listen(8888);

## JavaScript Terms

### Window Object

window.innerWidth 🡪 Browser’s window width

window.innerHeight 🡪 Browser’s window height

window.close() 🡪 Closes the browser window

window.setTimeout() 🡪 Time out for window

window.setInterval() 🡪 Time interval for window

window.print() 🡪 Printing the page to the printer

window.confirm() 🡪 Similar to prompt, have okay or cancel button

window.history 🡪 Contains browsing history

window.location 🡪 URL of the current page, can be set to direct the browser to load a new page.

### Regular Expression Constructor

new RegExp(/^\d{3}-?\d{4}$/);

### JavaScript Object Notation (JSON)

var fidoString = ‘{“name”: “Fido”, “breed”: “Mixed”, “weight”: 38}’;

We are using single quotes around the JSON string. We have to use single quotes because the string contains double quotes, so JavaScript will get confused otherwise. This way, JavaScript knows this is one long string that contains other strings

var fido = JSON.parse(fidoString)

Passing the string to convert it to an Object

var fido = {name: “Fido”, breed: “Mixed”, weight: “38”}

var fidoString = JSON.stringify(fido);

Passing the object to convert it to a string

|  |  |
| --- | --- |
| **Debugging**  console.log()  alert()  prompt()  typeof  undefined  NULL  isNaN()  **Math**  Math.floor()  Math.PI  Math.random()  **Array**  [array].length()  [array].push()  [array].sort(*compareNumbers*)  [array].toString()  [array].reverse()  [array].join()  [array].forEach(function\_name)  **Events**  window.onload = init  window.onunload  window.resize  var = eventObject.target  var.id, var.src, var.hasAttribute()  var.removeAttribute(),  var.setAttribute()  [document].onclick  [document].onkeypress  [document].onmousemove  [document].onmouseover  [document].onmouseout  [document].pause  [document].onkeypress  [document].drop  [document].touchend  [document].pause  [document].play  [document].dragstart | **Timer**  setInterval()  setTimeout(function, time)  **Boolean False**  undefined  null  isNaN()  **String**  [string].indexOf(var, starting)  [string].split()  [string].trim()  [string].toUpperCase()  [string].toLowerCase()  [string].replace(delimiter, replace)  CharAt()  [string].substring(start, end)  [string].slice(start, end)  [regex[.test(string)  [string].concat(string1, string2,…)  **Date Object**  [new Date()].getFullYear()  [new Date()].getDay()  [new Date()].toString()  [new Date()].getTime()  **HTML Document**  document.getElementById()  document.getElementsByTagName()  document.write()  [HTML Document].innerHTML  [HTML Document].textContent  [HTML Document].setAttribute()  [HTML Document].value  [HTML Document].getElementByClassName  [HTML Document].getElementByName  [HTML Document].querySelector  [HTML Document].querySelectorAll  [HTML Document].createElement(“”)  [HTML Document].appendChild()  **Array Object**  new Array()  [array].every() |

# JavaScript Advanced

## Document Object Model

**createElement(element)**

This method creates an Element Node with the specified name.

**createTextNode(element)**

This method creates a Text Node with the specified text.

**[element].appendChild(element)**

This method appends a node as the last child of a node.

var h = document.createElement("H1")  // Create a <h1> element  
var t = document.createTextNode("Hello World"); // Create a text node  
h.appendChild(t); // Append the text to <h1>

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Add Songss </**title**>

<**meta** charser = "UTF-8"/>

<**script** src = "playlist.js"></**script**>

</**head**>

<**body**>

<**form**>

<**input** type = "text" id = "songTextInput" size = "40" placeholder = "Song Name">

<**input** type = "button" id = "addButton" value = "Add Song"/>

</**form**>

<**ul** id = "playlist">

</**ul**>

</**body**>

</**html**>

**function** init(){

**var** button = document.getElementById("addButton");

button.onclick = handleButtonClick;

}

**function** handleButtonClick(){

**var** textInput = document.getElementById("songTextInput");

**var** songName = textInput.value;

**var** list = document.createElement("li");

list.innerHTML = songName;

**var** unorderedList = document.getElementById("playlist");

unorderedList.appendChild(list);

}

window.onload = init;

**Code 1: playlist.js**

## Method Chaining

**movie.showtimes.length 🡪 Method Chaining**

**function** init(){

getShows(movie2);

}

**function** getShows(movie){

**for** (**var** i = 0; i < movie.showtimes.length; i++){

console.log("Next Show for " + movie.title + ": " +

movie.showtimes[i]);

}

}

**var** movie = {

title: "Plan 9 from Outer Space",

genre: "Cult Classic",

rating: 5,

showtimes: ["03:00 pm", "07:00 pm", "11:00 pm"]

};

**var** movie2 = {

title: "Forbidden Planet",

genre: "Classic Sci-Fi",

rating: 5,

showtimes: ["05:00 pm", "09:00 pm"]

};

window.onload = init;

**Code 2: methodchaining.js**

## Geolocation

Geolocation is not considered a first-class member of existing HTML5 standard, but that said, it is a standard of the W3C widely supported and pretty much everyone includes Geolocation in the list of important HTML5 APIs.

Geolocation API determines our location

**GPS**

Global Positioning System, supported by many newer mobile devices, provides extremely accurate location information based on satellites. Location data may include altitude, speed and heading information. To use it, device has to be able to see the sky, and it can take a long time to get a location. GPS can also be hard on our batteries.

**IP Address**

Location information based on our IP addresses uses an external database to map the IP address to a physical location. the advantage of this approach is that it can work anywhere, however, often IP addresses are resolved to locations such as ISP’s local office. This method as being reliable to the city or sometimes neighborhood level.

**Cell Phone**

Cell phone triangulation figures out our location based on our distance from one or more cell phone towers. More the towers, the more accurate our location will be. This method can be fairly accurate and works indoors (unlike GPS); it also can be much quicker than GPS. If we are in the middle of nowhere with only one cell tower, our accuracy is going to suffer.

**WIFI**

WIFI positioning uses one or more WIFI access points to triangulate our location. This method can be very accurate, works indoors and is fast. It requires us to be stationary

**navigator.geolocation**

This allows us to use the geographical position of a user

**navigator.geolocation.getCurrentPosition(displayLocation, displayerror)**

This method is used to return the user's position.

**displayLocation(position)**

position.coords.latitude;

position.coords.longitude;

Helps us to get the latitude and longitude

**displayError(error)**

**error.code 🡪 0 => Unknown error, 1 => User Permission Denied,**

**3 => Position Unavailable, 4 => Request Timed Out**

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Geolocation </**title**>

<**meta** charset = "UTF-8"/>

<**script** src = "geolocation.js"></**script**>

</**head**>

<**body**>

<**div** id = "myLocation">

</**div**>

</**body**>

</**html**>

**function** getMyLocation(){

**if**(navigator.geolocation){

navigator.geolocation.getCurrentPosition(displayLocation, displayError);

}**else**{

console.log("Geolocation Not Supported");

}

}

**function** displayLocation(position){

**var** latitude = position.coords.latitude;

**var** longitude = position.coords.longitude;

**var** div = document.getElementById("myLocation");

div.innerHTML = "Latitude: " + latitude + "<br/>" +

"Longitude: " + longitude;

}

**function** displayError(error){

**var** errorTypes = {

0 : "Unknown error",

1 : "Permission denied by user",

2 : "Position is not available",

3 : "Request timed out"

};

**var** errorMessage = errorTypes[error.code];

**if**(error.code == 0 || error.code == 1){

errorMessage = errorMessage + " " + error.message;

}

**var** div = document.getElementById("muLocation");

div.innerHTML = errorMessage;

}

window.onload = getMyLocation;

**Code 3: geolocation.js**

### Computing Distance Between Two Geolocation Points

Therefore,

This uses the **‘haversine’ formula** to calculate the great-circle distance between two points – that is, the shortest distance over the earth’s surface – giving an ‘as-the-crow-flies’ distance between the points (ignoring any hills they fly over).

**Haversine Derivation**:

a = sin²(Δφ/2) + cos φ1 ⋅ cos φ2 ⋅ sin²(Δλ/2)

c = 2 ⋅ atan2( √a, √(1−a) )

**d = R ⋅ c**

where φ is latitude, λ is longitude, R is earth’s radius (mean radius = 6,371km);

note that angles need to be in radians to pass to trigonometric functions

var R = 6371e3; // metres

var φ1 = lat1.toRadians();

var φ2 = lat2.toRadians();

var Δφ = (lat2-lat1).toRadians();

var Δλ = (lon2-lon1).toRadians();

var a = Math.sin(Δφ/2) \* Math.sin(Δφ/2) +

Math.cos(φ1) \* Math.cos(φ2) \* Math.sin(Δλ/2) \* Math.sin(Δλ/2);

var c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

var d = R \* c;

**function** getGeo(){

**if**(navigator.geolocation){

navigator.geolocation.getCurrentPosition(displayLocation, displayError);

}**else**{

**var** div = document.getElementById("distance");

div.innerHTML = "Geolocation Not Supported!";

}

}

**function** displayLocation(pos){

**var** div = document.getElementById("distance");

latitude = pos.coords.latitude;

longitude = pos.coords.longitude;

div.innerHTML = "Longitude: " + latitude + "<br/>" +

"Longitude: " + longitude;

**var** lat2 = 47.624;

**var** long2 = -122.52088;

**var** distance = computingDistance(latitude, longitude, lat2, long2);

**var** computed = document.getElementById("computed");

computed.innerHTML = distance;

}

**function** displayError(err){

**var** errorTypes = {

0: "Unknown error",

1: "User Permission Denied",

2: "Position Unavailable",

3: "Request Timed Out",

};

**var** errorMessage = error.types[err.code];

**if**(err.code == 0 || err.code == 2){

errorMessage = errorMessage + " " + err.message;

}

**var** div = document.getElementById("distance");

div.innerHTML = errorMessage;

}

**function** computingDistance(lat1, long1, lat2, long2){

**var** radius = 6371;

**var** latitude1 = degreesToRadian(lat1);

**var** longitude1 = degreesToRadian(long1);

**var** latitude2 = degreesToRadian(lat2);

**var** longitude2 = degreesToRadian(long2);

**var** delta\_lat = degreesToRadian((lat2 - lat1));

**var** delta\_long = degreesToRadian((long2 - long1));

**var** a = Math.pow(Math.sin(delta\_lat/2), 2) +

Math.cos(latitude1/2) \* Math.cos(latitude2/2) \* Math.pow(Math.sin(delta\_long/2),2);

**var** c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

**var** d = radius \* c;

**return** d;

}

**function** degreesToRadian(deg){

**return** ((Math.PI \* deg)/180);

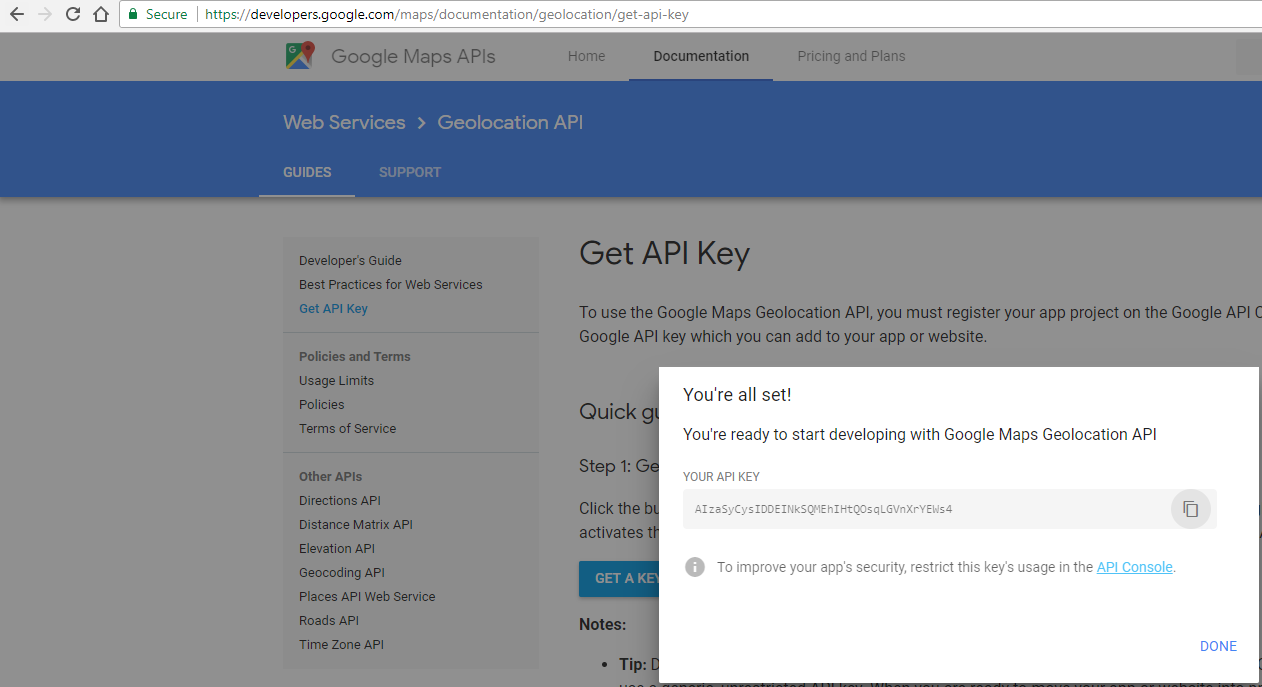
}

window.onload = getGeo;

**Code 4: computedDistance.js**

### Google Map

To use the google map, we need to generate a key and must have a Gmail account. Then generate a key for the google map from the website.

**Fig 1: googlegeneratekey.js**

Generate Key for the localhost

**AIzaSyD3sDQtedEfmXmqSmxL-3Fay7RM5NHWfPw**

Add this script

<script src=[https://maps.googleapis.com/maps/api/js?key= AIzaSyD3sDQtedEfmXmqSmxL-3Fay7RM5NHWfPw&callback=initMap](https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap) async defer></script>

For the map to appear in the html, we definitely to mention some of the properties in CSS

<**style**>

#map{**height**: 100%;}

**html**, **body** {**height**: 100%;**margin**: 0;**padding**: 0;}

</**style**>

Passing the Geolocation Coordination values in the Google Map function

google.maps.LatLng(latitude, longitude);

🡨 🡪 🡨 🡪

**google.maps** 🡺 It precedes all the methods of the google maps API.

**.LatLng** 🡺 This is the constructor, which takes out latitude and longitude, and returns a new object that holds them both.

**var** mapOptions = {

zoom: 19,

center: LatLng,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

The **zoom** option can be specified 0 to 21. Bigger numbers correspond to being zoomed in more. 10 is about city sized.

The **center** option can be specified where the location can be, our location has been specified

The **mapTypeId** option can be SATELLITE or HYBRID etc. …

**var** divMap = document.getElementById("map");

map = **new** google.maps.Map(divMap, mapOptions);

Creating the **map** over here

**var** title = "Eastern";

**var** content = "Latitude: " + coords.latitude + ", Longitude: " + coords.longitude;

**var** marker = **new** google.maps.Marker({

position: LatLng,

map: map,

title: 'Eastern!',

clickable: **true**

});

Creates a **marker** over here, Clickable is set to true, because we want to be able to display an info window when it is clicked, window options let us see information when clicked.

**var** infoWindowOptions = {

content: content,

position: LatLng

};

**var** infoWindow = **new** google.maps.InfoWindow(infoWindowOptions);

google.maps.event.addListener(marker, "click", **function**(){

infoWindow.open(map);

});

}

<!DOCTYPE html>

<html>

<head>

<title>Google Map</title>

<meta charset = "UTF-8">

<style>

#map{height: 100%;}

html, body {height: 100%;margin: 0;padding: 0;}

</style>

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

<script

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyD3sDQtedEfmXmqSmxL-3Fay7RM5NHWfPw"

async defer>

</script>

</head>

<body>

<div id = "distance"></div>

<div id = "computed"></div>

<div id = "map"></div>

</body>

</html>

**function** getGeo(){

**if**(navigator.geolocation){

navigator.geolocation.getCurrentPosition(displayLocation, displayError);

}**else**{

**var** div = document.getElementById("distance");

div.innerHTML = "Geolocation Not Supported!";

}

}

**function** showMap(coords){

**var** LatLng = **new** google.maps.LatLng(coords.latitude, coords.longitude);

**var** mapOptions = {

zoom: 19,

center: LatLng,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

**var** divMap = document.getElementById("map");

map = **new** google.maps.Map(divMap, mapOptions);

**var** title = "Eastern";

**var** content = "Latitude: " + coords.latitude + ", Longitude: " + coords.longitude;

**var** marker = **new** google.maps.Marker({

position: LatLng,

map: map,

title: 'Eastern!',

clickable: **true**

});

**var** infoWindowOptions = {

content: content,

position: LatLng

};

**var** infoWindow = **new** google.maps.InfoWindow(infoWindowOptions);

google.maps.event.addListener(marker, "click", **function**(){

infoWindow.open(map);

});

}

**function** displayLocation(pos){

**var** div = document.getElementById("distance");

latitude = pos.coords.latitude;

longitude = pos.coords.longitude;

div.innerHTML = "Longitude: " + latitude + "<br/>" +

"Longitude: " + longitude;

**var** lat2 = 47.624;

**var** long2 = -122.52088;

**var** distance = computingDistance(latitude, longitude, lat2, long2);

**var** computed = document.getElementById("computed");

computed.innerHTML = distance;

showMap(pos.coords);

}

**function** displayError(err){

**var** errorTypes = {

0: "Unknown error",

1: "User Permission Denied",

2: "Position Unavailable",

3: "Request Timed Out",

};

**var** errorMessage = error.types[err.code];

**if**(err.code == 0 || err.code == 2){

errorMessage = errorMessage + " " + err.message;

}

**var** div = document.getElementById("distance");

div.innerHTML = errorMessage;

}

**function** computingDistance(lat1, long1, lat2, long2){

**var** radius = 6371;

**var** latitude1 = degreesToRadian(lat1);

**var** longitude1 = degreesToRadian(long1);

**var** latitude2 = degreesToRadian(lat2);

**var** longitude2 = degreesToRadian(long2);

**var** delta\_lat = degreesToRadian((lat2 - lat1));

**var** delta\_long = degreesToRadian((long2 - long1));

**var** a = Math.pow(Math.sin(delta\_lat/2), 2) +

Math.cos(latitude1/2) \* Math.cos(latitude2/2) \* Math.pow(Math.sin(delta\_long/2),2);

**var** c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

**var** d = radius \* c;

**return** d;

}

**function** degreesToRadian(deg){

**return** ((Math.PI \* deg)/180);

}

window.onload = getGeo;

**Code 5: googlemap.js**

Using Google Map we can manipulate controls like pan control, Switch between Map and Satellite view and even the street view control. We can also use the direction services. Overlays can also be changed like heat map, check traffic, traffic overlay, custom overlays, our photos, custom markers.

### Tracking Movemnet

**getCurrentPosition(successHandler, errorHandler, positionOptions)**

The success handler (or callback) is called when a location is determined, and it is passed a position object.

The error handler is called when the browser can’t determine its location.

The position handler allows us to fine-tune the behavior of geolocation.

Coordinates

latitude

longitude

accuracy

altitude

altitudeAccuracy

heading

speed

Position

coords

timestamp

**Fig 2: Position 🡪 Coordinate (UML Diagram)**

Position Object contains the coords object(property) and the timestamp (property). Timestamp property contains the time the position object was created. This can be useful for knowing how old the location is.

Latitude, Longitude and accuracy are guaranteed to stay in the device. The rest may or may not be supported depending on our device.

**function** displayLocation(pos){

**var** div = document.getElementById("distance");

latitude = pos.coords.latitude;

longitude = pos.coords.longitude;

accuracy = pos.coords.accuracy;

div.innerHTML = "Longitude: " + latitude + "<br/>" +

"Longitude: " + longitude + "<br/>" +

"Accuracy: " + accuracy;

**var** lat2 = 47.624;

**var** long2 = -122.52088;

**var** distance = computingDistance(latitude, longitude, lat2, long2);

**var** computed = document.getElementById("computed");

computed.innerHTML = distance;

showMap(pos.coords);

}

**Code 6: updatedgooglemapwithaccuracy.js**

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> Tracking Movement </**title**>

<**meta** charset = "UTF-8"/>

<**script** src = "trackmovement.js"></**script**>

<**script**

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyD3sDQtedEfmXmqSmxL-3Fay7RM5NHWfPw"

async defer>

</**script**>

<**style**>

#map{**height**: 50%;}

**html**, **body** {**height**: 100%;**margin**: 0;**padding**: 0;}

</**style**>

</**head**>

<**body**>

<**form**>

<**input** type = "button" id = "watch" value = "Watch me">

<**input** type = "button" id = "clearWatch" value = "Clear Watch">

</**form**>

<**div** id = "location"></**div**>

<**div** id = "map"></**div**>

</**body**>

</**html**>

**var** watchId = **null**;

**var** map;

**function** getMyLocation(){

**if**(navigator.geolocation){

**var** watchButton = document.getElementById("watch");

watchButton.onclick = watchLocation;

**var** clearWatchButton = document.getElementById("clearWatch");

clearWatchButton.onclick = clearWatch;

}

**else**{

console.log("Geolocation is not supported!");

}

}

**function** watchLocation(){

watchId = navigator.geolocation.watchPosition(displayLocation, displayError);

}

**function** clearWatch(){

**if**(watchId){

navigator.geolocation.clearWatch(watchId);

watchId = **null**;

}

}

**function** displayLocation(pos){

**var** div = document.getElementById("location");

latitude = pos.coords.latitude;

longitude = pos.coords.longitude;

accuracy = pos.coords.accuracy;

div.innerHTML = "Longitude: " + latitude + "<br/>" +

"Longitude: " + longitude + "<br/>" +

"Accuracy: " + accuracy;

**if**(map == **null**){

showMap(pos.coords);

}

}

**function** displayError(err){

**var** errorTypes = {

0: "Unknown error",

1: "User Permission Denied",

2: "Position Unavailable",

3: "Request Timed Out",

};

**var** errorMessage = error.types[err.code];

**if**(err.code == 0 || err.code == 2){

errorMessage = errorMessage + " " + err.message;

}

**var** div = document.getElementById("location");

div.innerHTML = errorMessage;

}

**function** showMap(coords){

**var** LatLng = **new** google.maps.LatLng(coords.latitude, coords.longitude);

**var** mapOptions = {

zoom: 19,

center: LatLng,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

**var** divMap = document.getElementById("map");

map = **new** google.maps.Map(divMap, mapOptions);

**var** title = "Eastern";

**var** content = "Latitude: " + coords.latitude + ", Longitude: " + coords.longitude;

**var** marker = **new** google.maps.Marker({

position: LatLng,

map: map,

title: 'Eastern!',

clickable: **true**

});

**var** infoWindowOptions = {

content: content,

position: LatLng

};

**var** infoWindow = **new** google.maps.InfoWindow(infoWindowOptions);

google.maps.event.addListener(marker, "click", **function**(){

infoWindow.open(map);

});

}

window.onload = getMyLocation;

**Code 7: trackingmovements.js**

**watchPosition(successHandler, errorHandler)**

Our app calls the successHandler. watchPosition sits in the background and constantly monitors our positions. When our position changes, watchPosition calls our success handler function to report our new position. watchPosition continues to monitor our position (and report it to our success handler) until we clear it calling clearWatch

### Current Position with options

var positionOptions = {

enableHighAccuracy: false,

timeout: Infinity,

maximumAge: 0

}

**enableHighAccuracy:** We can control the accuracy of the geolocation

**timeout:** How long it gets to determine the user’s location

**maximumAgeL** How old the location can be. If the browser has a location that was determined sixty seconds go, and maximumAge is set to 90000 (90 seconds), then a call to getCurrentPosition would return the existing, cached position( the browser would not try to get a new one). But if the maximumAge was set to 30 seconds, the browser would be forced to determine a new position

## Asynchronous JavaScript and XML(AJAX)

Always check the size or length of the SQL object, it might hamper the when displaying with json\_encode()

Request object is created over here

**var request = new XMLHttpRequest();**

This sets up a request for us, using an HTTP GET request, which is the standard means of retrieving HTTP data. It also sets up the request to use the URL stored in our URL variable

**request.open(“GET”, url);**

200 is the browsers answer from the remote web service (when it is successful)

**request.onload** is same as **request.onreadystatechange** (request.onload is not supported by old browsers)

request.onload = function(){

if(request.status == 200){

//Do something

}

};

**request.sent(null);**

**JSON.stringify(array)** 🡪 turns array into an object 🡪 JSON Format

**JSON.parse(object)** 🡪 turns object into an array 🡪 JSON Format

**We check for request.status == 200 and request.readyState() == 4**

<!DOCTYPE html>

<html>

<head>

<title> AJAX </title>

<meta charset = "UTF-8"/>

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

</head>

<body>

<ul id = "sales">

</ul>

</body>

</html>

<?php

$server\_name = "localhost";

$username = "root";

$pass = "";

$db = "riskyjobs";

$connection = mysqli\_connect($server\_name, $username, $pass, $db)

**or** **die** ("Server Denied");

$query = "SELECT \* FROM riskyjobs";

$data = mysqli\_query($connection, $query)

**or** **die**("Query Denied");

$array\_data = **array**();

**while**($row = mysqli\_fetch\_array($data)){

$id = $row['job\_id'];

$title = $row['title'];

$city = $row['city'];

$state = $row['state'];

$zip = $row['zip'];

$company = $row['company'];

$date\_posted = $row['date\_posted'];

$array\_data[] = ["id" => "**$id**",

"title" => "**$title**",

"city" => "**$city**",

"state" => "**$state**",

"zip" => "**$zip**",

"company" => "**$company**",

"date\_posted" => "**$date\_posted**"

];

}

**echo**(json\_encode($array\_data));

?>

**function** init(){

**var** url = "ajax.php";

**var** request = **new** XMLHttpRequest();

request.open("GET", url);

*// request.onload = function(){*

request.onreadystatechange = **function**(){

**if**(request.status == "200" && request.readyState == "4"){

displayJSON(request.responseText);

}

};

request.send(**null**);

}

**function** displayJSON(data){

**var** dataObject = JSON.parse(data);

**var** ul = document.getElementById("sales");

**for**(**var** i = 0; i < dataObject.length; i++){

*// console.log("ID: " + dataObject[i].id);*

*// console.log("Title: " + dataObject[i].title);*

*// console.log("City: " + dataObject[i].city);*

*// console.log("State: " + dataObject[i].state);*

*// console.log("Zip: " + dataObject[i].zip);*

*// console.log("Company: " + dataObject[i].company);*

*// console.log("Date Posted: " + dataObject[i].date\_posted);*

*// console.log("==========================================");*

**var** li = document.createElement("li");

li.setAttribute("class", "title");

li.innerHTML = dataObject[i].title;

ul.appendChild(li);

}

}

window.onload = init;

**Code 8: ajax.js**

### JSON with Padding

We can’t retrieve data from a domain that is different from the domain the page itself was served from. Therefore, the **XMLHttpRequest** cross-origin request would fail. JSONP is a way to retrieve JSON objects by uing the <script> tag. It’s also a way of retrieving data and this avoids the same-origin security issues. **JSONP** stands for JavaScript Object Notation with Padding.

**Steps**

* We include a <script> element. The source for this script is the URL of a web service that is going to supply us with JSON data.
* The browser encounters the <script> element in the page and sends an HTTP request to the src URL
* **The server treats request and sends back the JSON string. Since the server sends back the JSON string, it first wraps it in a function call, like a call.**
* This time when the JSON response is parsed and interpreted, it is wrapped in a function call. And so that function is called, and the object created from the JSON string is passed to it.

[**http://localhost/ajax.php?callback=update**](http://localhost/ajax.php?callback=update)

### Cross Origin Resource Sharing

**header('content-type: application/json; charset=utf-8');**

**header("access-control-allow-origin: \*");**

To let everyone to see my data in JSON format, we need to flag to open. It is a specification that enables a truly open access across domain-boundaries. CORS defines how browsers and servers communicate when accessing sources across origins using HTTP headers to allow both the browser and the server to know enough about each other to determine if the request or response should succeed or fail.

<?php

$server\_name = "localhost";

$username = "root";

$pass = "";

$db = "riskyjobs";

$connection = mysqli\_connect($server\_name, $username, $pass, $db)

**or** **die** ("Server Denied");

$query = "SELECT \* FROM riskyjobs";

$data = mysqli\_query($connection, $query)

**or** **die**("Query Denied");

$array\_data = **array**();

**while**($row = mysqli\_fetch\_array($data)){

$id = $row['job\_id'];

$title = $row['title'];

$city = $row['city'];

$state = $row['state'];

$zip = $row['zip'];

$company = $row['company'];

$date\_posted = $row['date\_posted'];

$array\_data[] = ["id" => "**$id**",

"title" => "**$title**",

"city" => "**$city**",

"state" => "**$state**",

"zip" => "**$zip**",

"company" => "**$company**",

"date\_posted" => "**$date\_posted**"

];

}

**echo** "update(".json\_encode($array\_data).")";

?>

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> JSON P Basics </**title**>

<**meta** charset = "UTF-8"/>

<**script** src = "jsonp.js"></**script**>

</**head**>

<**body**>

<**h1**> Json P Data</**h1**>

<**ul** id = "data"></**ul**>

<**script** src = "jsonp.php?callback=update"></**script**>

</**body**>

</**html**>

window.onload = **function**(){};

**function** update(data){

console.log(data);

**var** ul = document.getElementById("data");

**for**(**var** i = 0; i < data.length; i++){

**var** li = document.createElement("li");

li.setAttribute("class", "title");

li.innerHTML = data[i].title;

ul.appendChild(li);

}

}

**Code 9: jsonp.js**

If we want to update the data after a certain interval of time without refreshing the page. Then we need to create the script tag manually through JavaScript.

**[parent\_element].replacechild(newChild, oldChild)**

This method replaces the children

**[element].remove()**

This method removes the element

<!DOCTYPE html>

<**html**>

<**head**>

<**title**> JSON P Basics </**title**>

<**meta** charset = "UTF-8"/>

<**script** src = "jsonp.js"></**script**>

</**head**>

<**body**>

<**h1**> Json P Data</**h1**>

<**div** id = "info"></**div**>

</**body**>

</**html**>

<?php

$server\_name = "localhost";

$username = "root";

$pass = "";

$db = "riskyjobs";

$connection = mysqli\_connect($server\_name, $username, $pass, $db)

**or** **die** ("Server Denied");

$query = "SELECT \* FROM riskyjobs";

$data = mysqli\_query($connection, $query)

**or** **die**("Query Denied");

$array\_data = **array**();

**while**($row = mysqli\_fetch\_array($data)){

$id = $row['job\_id'];

$title = $row['title'];

$city = $row['city'];

$state = $row['state'];

$zip = $row['zip'];

$company = $row['company'];

$date\_posted = $row['date\_posted'];

$array\_data[] = ["id" => "**$id**",

"title" => "**$title**",

"city" => "**$city**",

"state" => "**$state**",

"zip" => "**$zip**",

"company" => "**$company**",

"date\_posted" => "**$date\_posted**"

];

}

**echo** "update(".json\_encode($array\_data).")";

?>

window.onload = **function**(){

setInterval(handleRefresh, 3000);

};

**function** handleRefresh(){

**var** url = "jsonp.php?callback=update";

**var** newScript = document.createElement("script");

newScript.setAttribute("src", url);

newScript.setAttribute("id", "jsonp");

**var** oldScript = document.getElementById("jsonp");

**var** head = document.getElementsByTagName("head")[0];

**if**(oldScript == **null**){

head.appendChild(newScript);

}**else**{

head.replaceChild(newScript, oldScript);

}

}

**function** update(data){

console.log(data);

**var** div = document.getElementById("info");

**var** ulExists = document.getElementById("data");

**if**(ulExists){

ulExists.remove();

}

**var** ul = document.createElement("ul");

ul.setAttribute("id", "data");

**for**(**var** i = 0; i < data.length; i++){

**var** li = document.createElement("li");

li.setAttribute("class", "title");

li.innerHTML = data[i].title;

ul.appendChild(li);

}

div.appendChild(ul);

}

**Code 9: jsonpmodified.js**

We can also modify the URL by adding data and time to add uniqueness

**var url = “**[**http://localhost/jsonp.php?callback=update**](http://localhost/jsonp.php?callback=update)**” + “&random=” +**

**(newDate().getTime());**

## Canvas

**var selectedIndex = shape.selectedValue;**

**var shape = shape[selectedIndex].value;**

Getting the selected value from the form

**var context = canvas.getContext("2d");**

Needs 2D context from the canvas to draw

**context.fillStyle = “lightblue”**

fillStyle is a property, not a method. So, we set it, rather than call it. We can set it to a color by using the CSS color format which is color names (Ex: maroon, lightblue) or values like #cccff or rgb(0, 173, 239)

### Rectangle

**context.fillRect(initialPosX, initialPosY, width, height);**

The first two parameters are the x, y position of the rectangle

The last two parameters are the width and height of the rectangle

### Triangle

If we want to create a triangle, there is not create triangle method, therefore, we need to look into other canvas API built-in functions

**context.beginPath()**

Starting a new path

**context.moveTo(100, 50)**

This pin points the location where the line should start

**context.lineTo(250, 75)**

This method traces a path from the pencil’s current location to another point on the canvas.

Therefore, if the pencil’s initial point was {context.moveTo(100,50)} then it created a line between (100, 50) 🡪 (250, 75)

**Another context.lineTo(125, 30)** will create a line from (250, 75) 🡪 (125, 30)

Therefore, we have a line from (100, 50) 🡪 (250, 75) then another line from (250, 75) 🡪 (125, 30)

**context.closePath()**

This connects the starting point of the path to the last point in the current path

Therefore, (125, 30) 🡪 (250, 75)

### Circle

**context.beginPath()**

Starts a new path

**context.arc(intialPosX, initialPosY, radius, start∠, end∠, true)**

The first two parameters are the initial position

The third parameter is the radius

The fourth and fifth parameter is the starting and ending angle in radians

The fifth parameter is the direction whether it is clockwise (false), counterclockwise (true)

**context.fill()**

It is used to fill the color in the path

## JavaScript Terms

≪Document Object Model≫

**createElement(element)**

This method creates an Element Node with the specified name.

**createTextNode(element)**

This method creates a Text Node with the specified text.

**[element].appendChild(element)**

This method appends a node as the last child of a node.

**[parent\_element].replacechild(newChild, oldChild)**

This method replaces the children.

**[element].remove()**

This method removes the element

≪Geolocation≫

**navigator.geolocation**

This allows us to use the geographical position of a user

**navigator.geolocation.getCurrentPosition(displayLocation, displayerror)**

This method is used to return the user's position.

**getCurrentPosition(successHandler, errorHandler, positionOptions)**

The success handler (or callback) is called when a location is determined, and it is passed a position object. The error handler is called when the browser can’t determine its location.

The position handler allows us to fine-tune the behavior of geolocation

**displayLocation(position)**

position.coords.latitude;

position.coords.longitude;

position.coords.accuracy;

**watchPosition(successHandler, errorHandler)**

Our app calls the successHandler. watchPosition sits in the background and constantly monitors our positions. When our position changes, watchPosition calls our success handler function ro report our new position. watchPosition continues to monitor our position (and report it to our success handler) until we clear it calling clearWatch

≪Google Map≫

**new google.maps.Map(document, mapOptions)**

This creates and returns map object

**new google.maps.LatLng(latitude, longitude);**

google.maps 🡺 It precedes all the methods of the google maps API.

.LatLng 🡺 This is the constructor, which takes out latitude and longitude, and returns a new object that holds them both.

**new google.maps.Marker(markerOptions)**

This constructor creates marker object

**new google.maps.InfoWindow(infoWindowOptions)**

This constructor creates Info Window

**google.maps.event.addListener(marker, “click”, function)**

This adds listened to google Map

**infoWindow.open(map)**

It causes the window in google map to open

≪AJAX≫

request = new XMLHttpRequest();

request.open(“GET”, url);

request.onload = function(){

if(request.status == 200){

//do something

var data = request.responseText;

}

};

request.send(null);

**request.status == 200**

**request.readyState == 4**

**request.onload** is same as **request.onreadystatechange**

**JSON.stringify($array)**

Turns array into an object 🡪 JSON Format

**JSON.parse($object)**

Turns object into an array 🡪 JSON Format

≪Canvas≫

**var context = canvas.getContext("2d");**

Needs 2D context from the canvas to draw

**context.fillRect(intialPosX, initialPosY, width, height);**

The first two parameters are the x, y position of the rectangle

The last two parameters are the width and height of the rectangle

**context.fillStyle = “lightblue”**

It is a property fills out the color of the shapes.

**context.beginPath()**

Starting a new path

**context.moveTo(100, 50)**

This pin points the location where the line should start

**context.lineTo(250, 75)**

This method traces a path from the pencil’s current location to another point on the canvas.

Therefore, if the pencil’s initial point was {context.moveTo(100,50)} then it created a line between (100, 50) 🡪 (250, 75)

**Another context.lineTo(125, 30)** will create a line from (250, 75) 🡪 (125, 30)

Therefore, we have a line from (100, 50) 🡪 (250, 75) then another line from (250, 75) 🡪 (125, 30)

**context.closePath()**

This connects the starting point of the path to the last point in the current path

Therefore, (125, 30) 🡪 (250, 75)

**context.beginPath()**

Starts a new path

**context.arc(intialPosX, initialPosY, radius, start∠, end∠, true)**

The first two parameters are the initial position

The third parameter is the radius

The fourth and fifth parameter is the starting and ending angle in radians

The fifth parameter is the direction whether it is clockwise (false), counterclockwise (true)

**context.fill()**

It is used to fill the color in the path

≪Form≫

**var selectedIndex = shape.selectedValue;**

**var shape = shape[selectedIndex].value;**

Getting the selected value from the form