Call and Apply Methods

A function can use an unspecific object called this and use that objects properties in its body

const me = { fname: ‘Dean’, lname: ‘Carver’};

function sayMyName() {

console.log(`I’m ${this.fname} ${this.lname}. Hello!`);

}

Memoization

Allows results of a function call to be stored in an array cache

IIFE

A function that is invoked upon definition

(function() {

…

} ()

This is especially good when using strict mode, incase code from someone else doesn’t fit within strict mode (function() { ‘use strict’; /\* the rest of ALL of the code goes here})();

Temporary variables

Similar syntax to IIFE. Allows a variable to be accessible outside of its scope and does not exist outside of its scope

(()=>{

Const temp = 3;

})();

Console.log(temp) // temp not defined

Redefine functions

Functions can be redefined, even within themselves

function party(){

console.log('Wow this is amazing!');

party = function(){

console.log('Been there, got the T-Shirt');

}

}

The first time this function is executed, it will run Wow this is amazing, and anytime after Been there got the T-shirt

Recursive Functions

A function that calls itself until a certain condition is met

Promises

const promise = new Promise( (resolve, reject) => {

// initialization code goes here

if (success) {

resolve(value);

} else {

reject(error);

}

});

These are used to avoid callback hell. Used for async functions. .then is used to deal with a successful ‘resolved’ promise .then (“success”, “failure”) .catch is used for a failed promise

Async functions

async function loadGame(userName) {

try {

const user = await login(userName);

const info = await getPlayerInfo (user.id);

// load the game using the returned info

}

catch (error){

throw error;

}

}

Allows to make a function behave asynchronously by using the async and await key words. In this example getPlayerInfo will not execute until login finishes

Closures

Closures are functions that return another function, but with the inner function continuing to have access to the ‘outer’ functions elements

function closure() {

const a = 1.8;

const b = 32;

return c => c \* a + b;

}

toFahrenheit(30);

<< 86

Pure functions

Three rules

1. 1+ arguments
2. No side effects – variables and other data are not changed, only used
3. Return value

Currying functions

A function that returns another function when not all of the parameters of the main function are supplied, this allows the omitted parameters to be supplied later

This can also allow easier use later if one parameter is being repeated frequently, the ‘copied’ or ‘curried’ function can be reused for the second parameter.

function multiplier(x,y) {

if (y === undefined) {

return function(z) {

return x \* z;

}

} else {

return x \* y;

}

}

Ajax

Asynchronous JavaScript And XML

Regularly used for client-server communication

Used for Single Page Applications(SPA)

FETCH API

fetch(pokeapi.co/type/3)

.then( //show all those pokemon)

.catch( // code that runs if the server returns an error )

Response

The actual response for the browser or api

fetch(url)

.then((response) => {

if(response.ok) {

return response;

}

throw Error(response.statusText);

})

.then( response => // do something with response )

.catch( error => console.log('There was an error!') )