CH 11Further Functions

Functions are first-class objects in JavaScript.

Functions can be given properties by initializing them using dot notation. Ex: square.description = “ “

allow us to invoke the function as soon as its defined. This is useful when dealing with temp variables. Once a variable is declared in a scope, there’s no way to remove it, thus, IFFE are helpful in the sense that they can be used to create variables that disappear after the IFFE is finished.

(function() {

const name = 'Peter Parker'; // This might be obtained from a cookie in reality

const days = ['Sunday','Monday','Tuesday','Wednesday','Thursday', 'Friday','Saturday'];

const date = new Date(),today = days[date.getDay()];

console.log(`Welcome back ${name}. Today is ${today}`);

})();

<< 'Welcome back Peter Parker. Today is Tuesday'

JavaScript functions can redefine themselves.

function party(){

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

party = function(){

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

}

}

Callbacks can be used to allow JavaScript to be async

Promises are similar to callbacks but output the result / state

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

// initialization code goes here

if (success) {

resolve(value);

} else {

reject(error);

}

});

Async Functions:

Declare function as async with the keyword

Await is used to ensure the next line of code is not executed until line is finished

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;

}

}

Closure

A closure is a reference to a variable that was created inside the scope of another function, but is then kept alive and used in another part of the program.

(This means that by default, whenever a function is defined inside another function, the inner function will have access to any variables that are declared in the outer function's scope.) By returning the name of the function, the outer function can now have access to both what’s out of the inner function and what’s within it.

function outer() {

const outside = 'Outside!';

function inner() {

const inside = 'Inside!';

console.log(outside);

console.log(inside);

}

return inner;

}

Generators

Generators can be defined by placing a \* Infront of a function.

A generator returns a generator object that is used to create an iterator.

Yield is similar to return, except it holds on to the value it had before

function\* fibonacci(a,b) {

let [ prev,current ] = [ a,b ];

while(true) {

[prev, current] = [current, prev + current];

yield current;

}

}

AJAX

Fetch (AJAX)

const url = 'https:example.com/data';

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!') )

Fetch text() method

fetch(url)

.then( response => response.text() ); // transforms the text stream into a JavaScript string

.then( text => console.log(text) )

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

Example of API fetching

Due by Monday

Fix the menu bug, fix the extra blue space below header,

-name- (tutorial point how to build interface with Gmail) on contact us

-email-

-Comment-

Try implementing wood theme for the header banner

Activate.ps1

pip install virtualenv

.\venv/Scripts/activate.bat

Python3 app.py