Asynch and Await

Async/Await is a long anticipated JavaScript feature that makes working with asynchronous functions much more enjoyable and easier to understand. It is built on top of Promises and is compatible with all existing Promise-based APIs.

The name comes from async and await - the two keywords that will help us clean up our asynchronous code:

**Async - declares an asynchronous function (async function someName(){...}).**

* Automatically transforms a regular function into a Promise.
* When called async functions resolve with whatever is returned in their body.
* Async functions enable the use of await.

**Await - pauses the execution of async functions. (var result = await** someAsyncCall();).

* When placed in front of a Promise call, await forces the rest of the code to wait until that Promise finishes and returns a result.
* Await works only with Promises, it does not work with callbacks.
* Await can only be used inside async functions.

async **function** getData() {

**const** response = await fetch('test.json');

**const** body = await response.json()

console.log(body)

}

getData ();

Arrow Function

An **arrow function expression** has a shorter syntax than a [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function) and does not have its own [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this), [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments), [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), or [new. Target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target). These function expressions are best suited for non-method functions, and they cannot be used as constructors.

Example

**const** wishing = () => { console.log('good Morning');}

wishing ();

# One Line Function

It will not require curly braces.

**const** wishing = () => console.log('good Morning');

# Single Parameter

In Single parameter it doesn’t require any parenthesis

**const** wishing = name => {console.log(name+’gud morning’);}

wishing ();

Regular Expression

Regular expressions are a powerful way of searching and replacing inside a string.

In JavaScript regular expressions are implemented using objects of a built-in RegExp class and integrated with strings.

A regular expression also “regexp” or just “reg” consists of a *pattern* and optional *flags*.

In JavaScript, a regular expression is simply a type of object that is used to match character combinations in strings.

**There are two syntaxes to create a regular expression object.**

* **Regular Expression Literal** — This method uses slashes ( / ) to enclose the Regex pattern:

const reg = /hello/;

* **Regular Expression Constructor** — This method constructs the Expression for you:

var reg = new RegExp("hello");

# There are a number of methods we can use on our regex.

One basic method is **.test (),** which returns a Boolean

* Returns true: the string contains a match of the regex pattern
* Returns false: no match found

Example for regex

**var** myString= 'India is a big country'

**var** reg = /India/;

console.log (reg.test (myString));

#### Symbols

* **.**— (period) matches any single character, except for line breaks.
* **\***— Matches the preceding expression 0 or more times.
* **+**— Matches the preceding expression 1 or more times.
* **?**— Preceding expression is optional (Matches 0 or 1 times).
* **^**— Matches the beginning of the string.
* **$**— Matches the end of the string.

#### Character groups

* **\d**— Matches any single digit character.
* **\w**— Matches any word character (alphanumeric & underscore).
* **[XYZ]**— Character Set: Matches any single character from the character within the brackets. You can also do a range such as **[A-Z]**
* **[XYZ]+**— Matches one or more of any of the characters in the set.
* **[^A-Z]**— Inside a character set, the ^ is used for negation. In this example, match anything that is NOT an uppercase letter.

#### Flags:

* There are five optional flags. They can be used separately or together and are placed after the closing slash. Example: /[A-Z]/**g**I’ll only be introducing 2 here.
* **g** — Global search
* **i**— case insensitive search

#### Advanced

* **(x)** — Capturing Parenthesis: Matches x and remembers it so we can use it later.
* **(?:x)** — Non-capturing Parenthesis: Matches x and does not remembers it.
* **x(?=y)** — Lookahead: Matches x only if it is followed by y.