FUNCTIONS & ASYNCHRONOUS PROGRAMMING

FUNCTON DECLARATION

- Function declared using the 'function' keyword with a name

```
function name(parameters) {
        // code to be executed
 }
```

FUNCTION EXPRESSION

- Function assigned to a variable with a function name

```
const name = function(parameters) {
         // code to be executed
 };
```

ARROW FUNCTION

- Shorter syntax for the functions

```
const name = (parameters) => {
        // code to be executed
};
```

IMMEDIATELY INVOKED FUNCTION EXPRESSION [IIFE]

- Function executed immediately after they are defined

```
(function() {
        // code to be executed
})();
```

CALLBACK FUNCTION

- A function is passed as a parameter to another function

```
function fetchData(callback) {
        // code to fetch data
        callback(data);
}
```

PROMISE

- A promise is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

```
let promise = new Promise(function(resolve, reject) {
        // code to execute
        if (success) {
        resolve(result);
        } else {
        reject(error);
        }
});
```

CREATING A PROMISE

- Creating a promise in JavaScript involves instantiating a `Promise` object using the `new Promise` constructor and providing it with a function that takes `resolve` and `reject` as parameters to handle asynchronous operations.

```
let promise = new Promise((resolve, reject) => {
        // code to execute
        if (success) {
        resolve(result);
        } else {
        reject(error);
        }
});
```

PROMISE METHODS

```
promise.then(function(result) {
        // handle result
        }).catch(function(error) {
        // handle error
        }).finally(function() {
        // execute code regardless of result or error
});
```

ASYNC FUNCTION

- A function declared with the `async` keyword that allows the use of `await` within it to handle asynchronous operations more easily and readably.

```
async function name() {
        // code to be executed
}
```

AWAIT EXPRESSION

```
let result = await promise;
```

ERROR HANDLING IN ASYNC/AWAIT

```
try {
    let result = await promise;
} catch (error) {
    // handle error
}
```