**Functions**

Functions are a fundamental concept in JavaScript, allowing you to encapsulate and reuse code.

1 – Functions declerations :

* Function declarations are one of the primary ways to define functions in JavaScript. They are simple and straightforward, and they have some unique characteristics that distinguish them from other methods of defining functions
* A function declaration uses the function keyword followed by the function name, a list of parameters in parentheses, and a block of code enclosed in curly braces {}.
* One of the key features of function declarations is that they are hoisted. This means that the function definition is moved to the top of its enclosing scope during the compilation phase. As a result, you can call the function before it is defined in the code.
* Function declarations can return values using the return keyword. If no return statement is specified, the function returns undefined by default.
* Functions defined with a declaration have function scope. This means that variables declared within the function are not accessible outside of it.
* Function declarations are typically used when you need to define a reusable piece of code that can be called from different parts of your program. They are well-suited for defining functions that will be used multiple times throughout the code.

2 – Function expression :

* Function expressions are a powerful way to define functions in JavaScript. Unlike function declarations, function expressions involve creating a function and assigning it to a variable. This approach has unique characteristics and uses that make it suitable for various programming scenarios.
* A function expression assigns a function to a variable. It can be either named or anonymous (without a name).
* Function expressions are not hoisted in the same way that function declarations are. This means you must define the function expression before you can call it. If you try to call a function expression before its definition, you'll encounter a ReferenceError.
* Function expressions are often used in conjunction with closures. When a function expression is used inside another function, it can access variables from the outer function, creating a closure.
* A special case of function expressions is the Immediately Invoked Function Expression (IIFE), which is a function that runs immediately after it's defined. This pattern is often used to create a new scope and avoid polluting the global namespace.
* Arrow functions are a more concise syntax for writing function expressions introduced in ES6. They are particularly useful for short functions and have lexical scoping for this.

3 – Arrow functions :

Arrow functions, introduced in ECMAScript 6 (ES6), provide a concise syntax for writing functions and offer several key features that distinguish them from traditional function expressions.

Arrow functions have a more concise syntax compared to traditional function expression:

* Single Parameter and Single Expression
* Multiple Parameters and Block Body
* If there is only one parameter, parentheses around the parameter can be omitted.
* When an arrow function body contains a single expression, you can omit the curly braces {} and the return keyword. The expression is implicitly returned.

One of the key features of arrow functions is their lexical scoping of the this keyword. In arrow functions, this is not bound to the function itself but to the surrounding lexical context in which the function was defined. This is particularly useful for handling this in asynchronous code or callbacks.

4- Annonymous functions :

* Anonymous functions use the traditional function keyword and require explicit return statements for multi-line functions.
* Anonymous functions have their own this context, which can lead to issues when used inside methods or callbacks.
* Anonymous functions can be used as constructors with the new keyword.
* nonymous functions can be used as methods in objects and correctly handle this.

.