Functions

A BLOCK OF CODE ==> PERFORMS A TASK

There are multiple ways of function declaration.

```
function square(number) {
  // Function declaration
  return number * number;
}

// FUNCTION CALL (calling / executing a function)
const ans = square(6);
console.log(ans); // 36
```

The advantage of using a function declaration is having an access to 'this' keyword.

A function expression

Another way of declaring a function.

```
// const varName = function(params){
//statements
// }

const sayHi = function (name) {
  console.log(`Hello, ${name}`);
};

sayHi("Shubham");
```

An arrow Function

A modern way of writing functions in JavaScript.

```
// const varName = (params) => {
// statements
// };

const sayHi = (name) => {
  console.log(`Hello, ${name}`);
};

sayHi("Shubham");
```

```
const square = (number) => {
  return number * number;
};

// A short hand to declare arrow function if there is just one statement in the function:
const square1 = (number) => number * number;
```

Parameters and Arguments

Parameters are used when defining a function

Arguments are passed when making a function call.

```
const sayHi = (name, age = 0) => {
  // By default we can make the
  // value of age to be 0
  // to avoid undefined as a
  // result when age is not being passed.
  // name and age are parameters here
  console.log(`Hi ${name}, you are ${age} years old!`); // Hi Shubham, you are 0 years old.
};

sayHi("Shubham"); // 'Shubham' is an argument that is being passed
```

Functions - Custom

- Functions are **created/ defined** then they are **called**.
- Defining a function:

  ```
  // Function definition

  function calculateBill() {
      // this is the function body
      console.log('running calculateBill');
  }
  ```

- Calling a function:

  ```
  // Function call or run

  calculateBill(); // running calculateBill (returns undefined)
  ```

- Variables created inside a function are not available outside the function. e.g. `total` above.

  It is a **temporary variable.** After running of the function is complete, the variable is cleaned up or garbage-collected.

- **Returning value from function:**

  ```
  function calculateBill() {
      const total = 100 * 1.13;
      return total; // total is returned
  }

  calculateBill(); // returns 112.999999999
  ```

- Capturing returned value from a function into a variable:

  `const myTotal = calculateBill();` (myTotal will have value 112.999999999)

Scope

There are three types of Scope :

1. Global Scope
2. Function Scope
3. Block Scope

Global Scope

the global scope is the scope that contains, and is visible in, all other scopes.

```
const firstName = "Shubham";
const GlobalScope = () => {
  console.log(firstName); // Shubham
};

GlobalScope();
```

Local Scope

Local variables have Function Scope. They can only be accessed from within the function.

```
const someFunction = () => {
  let firstName = "John";
  console.log(firstName); // John

  const someFunction2 = () => {
    console.log(firstName); // John
  };

  someFunction2();
};

someFunction();
```

Block Scope

This scope restricts the variable that is declared inside a specific block, from access by the outside of the block.

```
if (true) {
  var firstName = "Jane";
}

console.log(firstName);
// Declaring a variable with var
// inside a blockscope can also let
// you access the variable in global scope.
```