# Pure Functions

## Pure Functions in Javascript

A pure function is a function that always returns the same output given the same input and has no side effects (i.e., it doesn't modify any variables outside of its scope, it doesn't mutate its input arguments, and it doesn't have any I/O operations such as reading from or writing to a file or a database). Pure functions are predictable and easier to reason about since they don't have any hidden dependencies or side effects. Pure functions can be composed together to create more complex functions or pipelines of functions since their input and output types are well-defined and consistent.

```
function add(a, b){ // A pure function adding two integers passed in it.
    return a+b;
}

function divide(a, b){  // Pure function to divide two integers passed
in it.
    return a/b;
}

function multi(a, b){    // Pure function to multiply two integers
passed in it.
    return a*b;
}
console.log(        // Calling all the pure functions
  add(2,5),
  multi(3,2),
  divide(20,5)
);
```

All three functions in the above code are pure functions. Their return value depends on the input arguments, they don't mutate any non-local state, and

they have no side effects (we will discuss side effects further in this article). Examples of pure functions in JavaScript include Math.abs(), parseInt(), JSON.stringify(), and many others.

Pure functions can be used with functional programming techniques such as higher-order functions, currying, and partial application. JavaScript libraries and frameworks such as Redux, Ramda, and Lodash emphasize using pure functions and functional programming principles.

## Impure Functions

An impure function is a function that either modifies variables outside of its scope, mutates its input arguments, has I/O operations such as reading from or writing to a file or a database, or has other side effects that are not purely computational. Impure functions can have hidden dependencies and side effects, which can make them harder to reason about and debug.

```
const message = 'Hi ! ';
function myMessage(value) {
    return `${message} ${value}`
}
console.log(myMessage('Aayushi'));
```

In the above code, the result the function returns is dependent on the variable that is not declared inside the function. That's why this is an impure function.
Examples of impure functions in JavaScript include console.log(), Math.random(), and Date.now(), Array.sort(), Array.splice(), and many others.

Impure functions can be necessary for tasks such as reading and writing to a file or a database, generating random numbers, or interacting with the user interface. However, it's important to minimize impure functions and keep them separate from pure functions as much as possible, to maintain a clear separation of concerns and avoid unexpected interactions or bugs. Pure functions can be composed together to create complex logic and pipelines of functions, whereas impure functions can only be used in a more limited and isolated way.

JavaScript libraries and frameworks such as React and Angular provide mechanisms for managing the state of an application and minimizing the use of impure functions in the user interface.