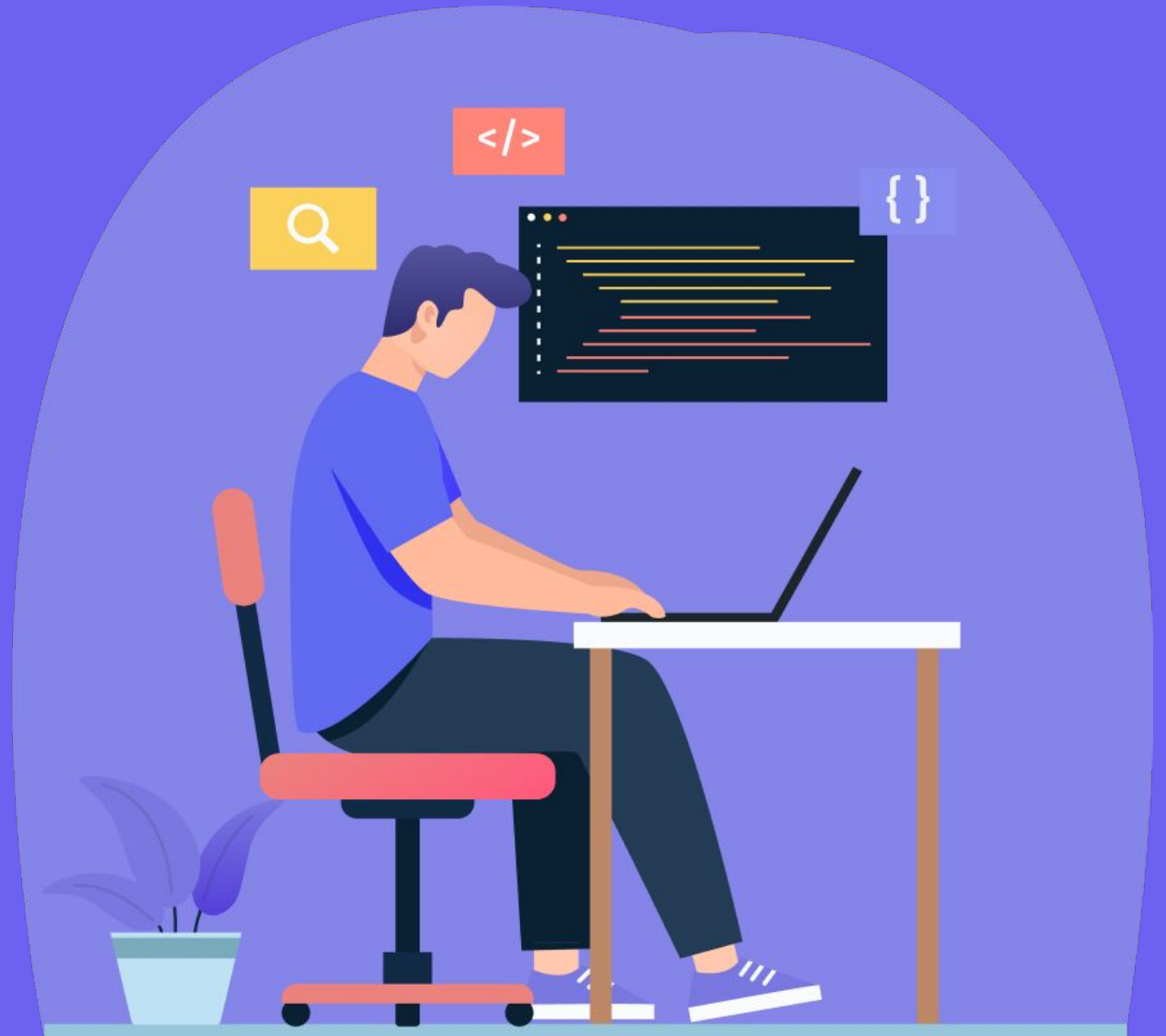


# Advanced Constructs: Advanced Function Concepts

**Relevel**  
by Unacademy



## Topics Covered

- Pass by Value
- Pass by Reference
- Pure / Impure function and Side Effects
- Closures
- Higher Order Functions
- Composability
- Arrow Functions
- Why are functions in JS 'first-class-citizen'
- IIFE
- Taking User input in JS



## Pass by Value

- A function is called by directly passing the value of the variable and any change in the variable inside the function does not affect the outside value of variable.
- As a beginner the above statement is bit confusing, let me explain this with real time example.
- In this example passByValue function will add value in the variable with 10  $\Rightarrow$  (value + 10) , So adding 10 with the variable does not change the value in the num variable.
- Because we are making the copy and send the value as an argument, here primitive type play an role in this example.

```
JS passByValue.js > ...
1  function passByValue (value) {
2      // Adding the value with 10
3      return (10 + value);
4  }
5
6  const num = 99;
7  console.log('num before passByValue function call', num);
8
9  const pbv = passByValue(num);
10
11 console.log('num after passByValue function call', num);
12 console.log('result after passByValue function call', pbv);
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

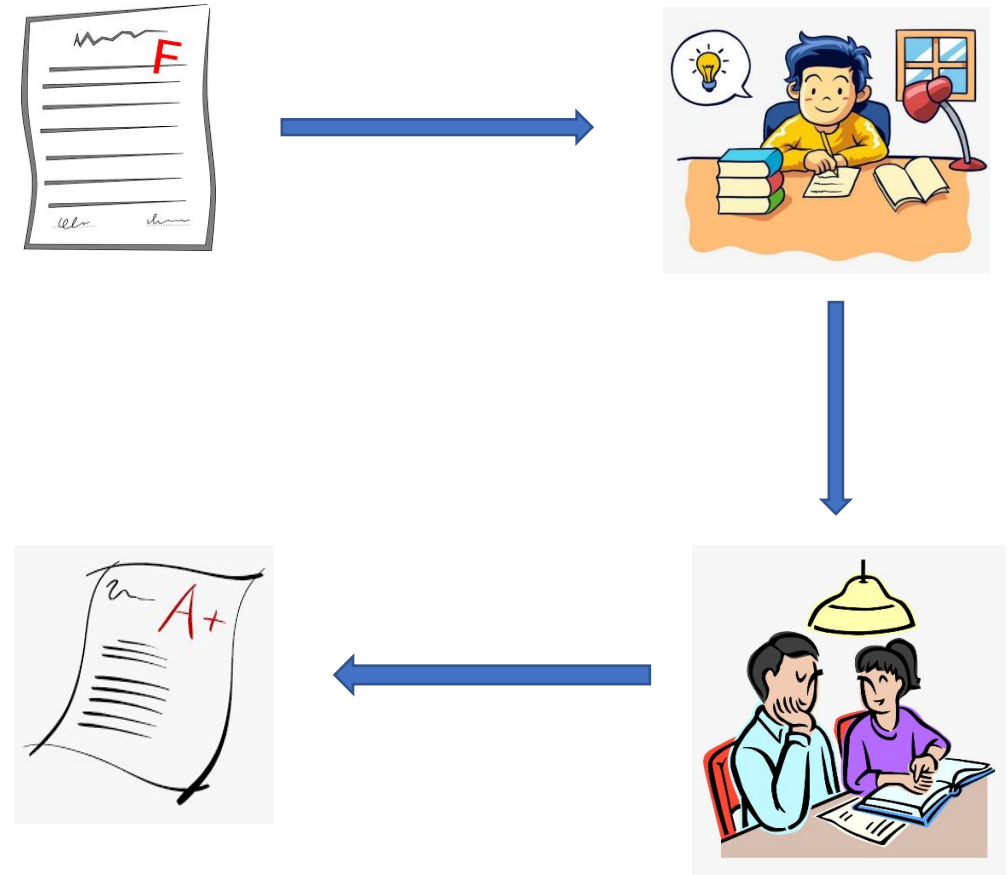
```
PS G:\Github\Relevel> node .\passByValue.js
num before passByValue function call 99
num after passByValue function call 99
result after passByValue function call 109
```

Let's consider you have scored low mark in the examination and your teacher asked you to get a signature in the copy of your mark sheet from your parent.

But getting signature for a low mark sheet is not easy, so you are changing the marks in the mark sheet copy and get a sign from your parent.

In this scenario, changing the marks in the copied mark sheet won't affect in the real mark sheet and at the end you are happy and the teacher also happy for completing the work.

If you caught then that is out of scope in this **example**.



- Below are the data types which support pass by value,

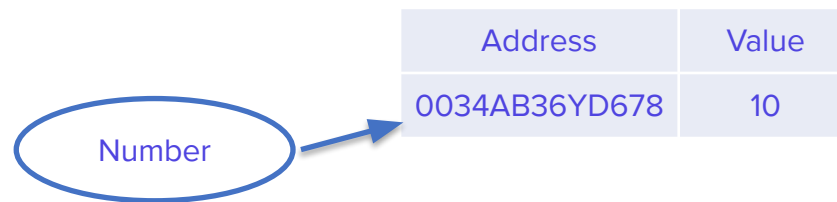
**Number**

**String**

**Boolean**

- Let me explain how pass by value works in terms of memory allocation in javascript.
- Consider a variable with name number and assigning value as 10

let number = 10



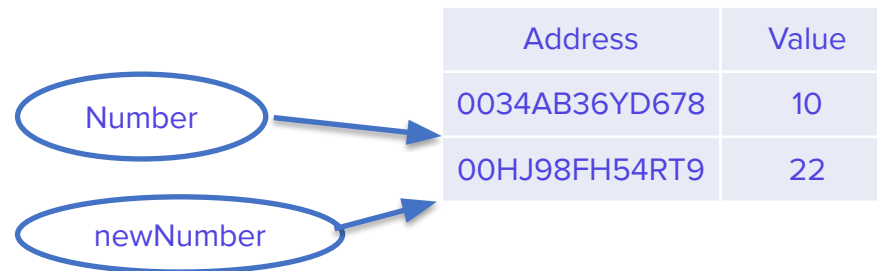
- In computers the variables are stored in memory with address and value pairs and the variable number is pointing to the address.
- Let us assign the variable number to variable newNumber

Let newNumber = number



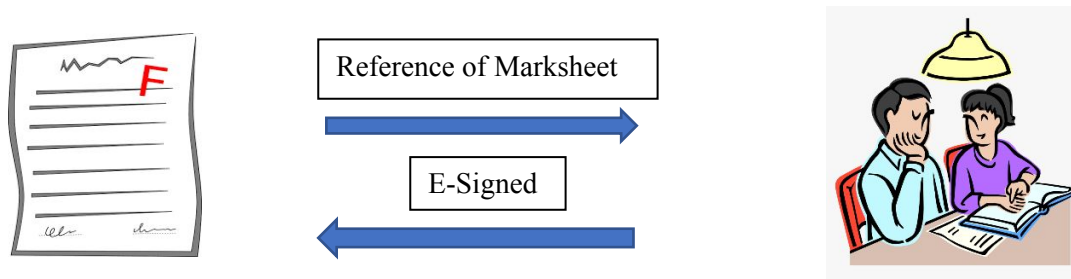
- Now both number and newNumber are pointing to same memory address.
- Let me change the add value of newNumber with 12. In this case both are pointing to same address and that is holding value 10, so changing the value in newNumber will change the value in number, but here the primitive data type comes into picture because they are immutable so the changes should not happen.

```
newNumber = newNumber + 12
```



## Pass by Reference

- A function is called by passing the reference of the variable and any change in the variable inside the function will make a change in the value of the outside variable.
- I hope this is also bit confusing, let me come up with an similar example of pass by value.
- Let's consider you have attended an online examination and scored low mark and your teacher asked you to get a e-signature in the original mark sheet from your parent which was uploaded in the school website.
- But getting signature for a low mark sheet is not easy, but you have convinced your parent and got a e-sign from your parent. In this scenario, your parents have signed the mark sheet in their home but that changes are reflected in the original marksheet.
- Marksheet is in the school Database but parents are opening the same marksheet in their home and e-signing by referring the marksheet.





## Programming Example

- In this example we have declared an array with string 'pass' and declared one function called pass By Reference, it should ask two argument one is array and second one is string.
- Here we are passing the array having one index and adding the string 'by Reference' in the array inside the function.
- But changing the variable inside the function make a change in the outside function because we are passing the reference of an array inside the function.
- Here the Non primitive data type comes in the picture because the value inside an array is mutable. Once it is initialized, we can change the value, adding the value, deleting the value inside an array.

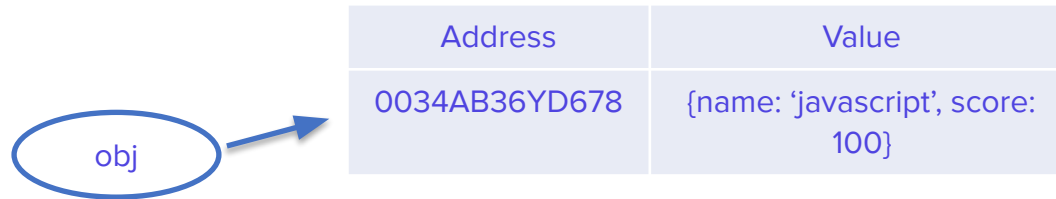
```
JS passByReference.js > ...
1 // Function Declaration
2 function passByReference (arr, value) {
3     arr.push(value);
4     console.log('***** Console Array into Pass by Reference *****');
5     console.log(arr);
6 }
7
8 // Declaring array and add "pass" string in the array
9 const arr = ['pass']
10
11 // logging the array before pass by Reference
12 console.log('***** Console Array Before Pass by Reference *****');
13 console.log(arr)
14
15 // Calling a function - passByReference with two argument 1 -> Array 2-> string
16 passByReference(arr, 'by Reference');
17
18 // logging the array after pass by Reference
19 console.log('***** Console Array After Pass by Reference *****');
20 console.log(arr);
21
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\passByReference.js
***** Console Array Before Pass by Reference *****
[ 'pass' ]
***** Console Array into Pass by Reference *****
[ 'pass', 'by Reference' ]
***** Console Array After Pass by Reference *****
[ 'pass', 'by Reference' ]
```

- Below are the data types which support pass by reference,
  - Object
  - Array
- Let me explain how pass by reference works in terms of memory allocation in javascript.
- Consider a object variable with name obj and having property name as javascript and score as 100

```
let obj = {name: 'javascript', score: 100}
```

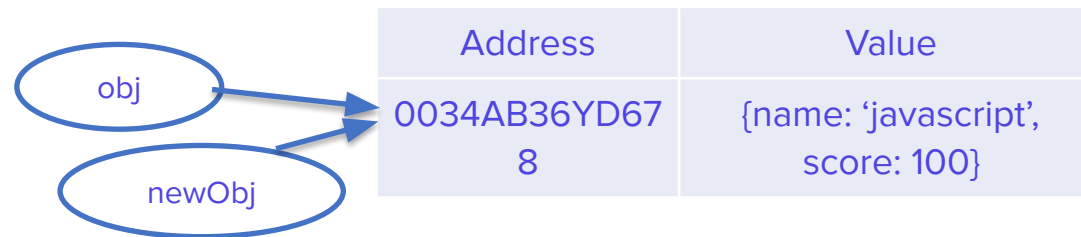


In computers the variables are stored in memory with address and value pairs and the variable `obj` is pointing to the address.

Let us assign the variable `obj` to variable `newObj`

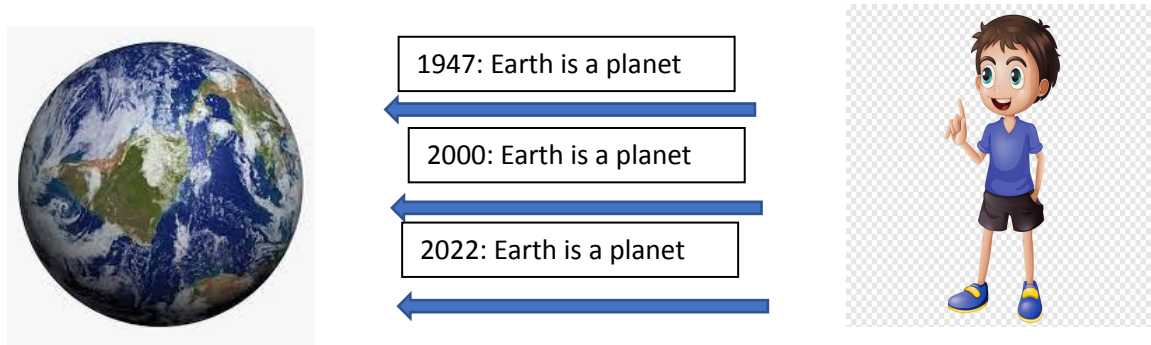
```
Let newObj = obj
```

- Now both obj and newObj are pointing to same memory address.
- Let me change the newObj property score as 90. In this case both are pointing to same address and that is holding value 10, so changing the property in newObj will change the property in obj since these are non primitive data type



## Pure Function

- Pure function does not change the state of variables out of its scope and it will always return same output if we pass the same input multiple number of times.
- The above definition is little bit hard to understand, so let me explain this with example.
- This real time example will show whenever you are studying about our planet earth every time the statement 'Earth is a planet' is same.



## Programming Example

Lets quickly jump into the coding example, here the variable num is declared outside of the function scope ie. num is a global variable and the function is not mutating the variable.

so every time we call the function getting the expected output and it has no side effects.

```
JS pureFunction.js > [?] result1
1  var num = 5;
2  // Pure function
3  const pureFunction = (num1, num2) => {
4    return num1 + num2;
5  };
6
7  //always returns same result given same inputs
8  const result1 = pureFunction(4, num);
9  console.log(result1);
10 //9
11 const result2 = pureFunction(num, 4);
12 console.log(result2);
13 //9
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\pureFunction.js
9
9
```

# Impure Function

- Impure function will change the state of variables out of its scope and it will always return different output if we pass the same input multiple number of times.
- The above definition is little bit hard to understand, so let me explain this with example.
- This above real time example will show whenever you are studying about pluto in school days we heard that pluto is the 9<sup>th</sup> planet in our solar system but now pluto is not a planet. It shows that every time it is changing and it has side effects.



1930: pluto is a 9th planet

2022: pluto is not a planet



## Programming Example

Lets quickly jump into the coding example, here the variable mutateNum is declared outside of the function scope

ie. mutateNum is a global variable and the function is mutating the variable every time when we are calling the function getting the different output and it has side effects.

```
JS impureFunction.js > ...
1  //Impure function
2  let mutateNum = 0;
3  const impureFunction = (num) => {
4    return (mutateNum += num);
5  };
6
7  //returns different result given same inputs
8  const result1 = impureFunction(5);
9  console.log(result1);
10 //5
11 const result2 = impureFunction(5);
12 console.log(result2);
13 //10
14 console.log('mutateNum', mutateNum);
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\impureFunction.js
5
10
mutateNum 10
```

## Comparison

Pure Function	Impure Function
It has no side-effects	It may have side-effects
It will return same output if same arguments are passed how many times it executes	it will return different output if same argument passed on multiple times
It will always returns something	It may take effect without returning anything
It is useful in some use cases	It is useful in some use cases



# Closure

- Closure is one of the important concept in Javascript. It is widely discussed concept in javascript world and still confusing concept. Let's understand closure in very simple ways.
- Closure is a function having access to the parent scope, even after the parent function has closed.
- Lets quickly check this definition by splitting the definition
  - o Closure is a function having access to the parent scope => which means a function(Parent function) is returning a function(child function)
  - o even after the parent function has closed => the variable which are present in the parent functions are accessible from the child function after called the parent function.

## Programming Example

- In this example function counter is returning the function which is called as anonymous function and that function is referring the variable which is available in the parent function.
- In the 10<sup>th</sup> line we are calling the function counter and storing the anonymous function in the variable, then calling the anonymous function will make the variable in the parent function available for child function.
- This concept is used to achieve the private function in javascript.

```
JS closure.js > counter
1  function counter () {
2      let count = 0;
3
4      return function (value) {
5          count += value;
6          console.log(count);
7      }
8  }
9
10 const counterCall = counter();
11 counterCall(1);
12 counterCall(2);
13 counterCall(3);
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\closure.js
1
3
6
```

- Let me explain you one more example with setTimeout function,
- In 1<sup>st</sup> for loop the output should be 4 for every execution, why because javascript won't wait for the setTimeout to be executed so the loop will execute and move the setTimeout function into execution and loop stopped at the end 'i' value is 4 and in the var keyword every 'i' in the log is pointing to the same memory location so the value is logging as 4
- In 2<sup>nd</sup> for loop the let keyword will allow to make a copy of value and pass into the setTimeout function, so every execution in for loop has a copy of 'i' so it is logging the expected output 0, 1, 2, 3

```
for (var i = 0; i < 4; i++) {  
  setTimeout(function() {  
    console.log(i);  
  }, i * 1000);  
}  
  
// Answer: 4 4 4 4  
  
for (let i = 0; i < 4; i++) {  
  setTimeout(function() {  
    console.log(i);  
  }, i * 1000);  
}  
  
// 0 1 2 3
```

# Higher Order Function

- Higher Order Function (HOF) are functions that takes other function as argument or return function as a result. I hope you are familiar with Arrays and their functions.
- Let me take you into deeper in this HOF concept. Arrays function such as map, filter, sort, reduce, forEach and so on are Higher Order function because these functions accept function as an argument and return desired output.

## Programming Example

In this example, using the inbuilt array manipulation functions,

- Using filter function which accept function as an argument with one argument in the function and do the filtration process
- Using sort function to arrange the array in the descending order and that sort function will accept function as an argument with two argument and do the sorting.

```
JS higherOrderFunction.js > [🔍] filteredArr
1  const arr = [1, 2, 3, 4, 5];
2
3  // Filter the array
4  const filteredArr = arr.filter(function(item){
5    |   return item > 3
6  })
7  console.log(filteredArr); // [4, 5]
8
9  // Sort the array in descending order
10 arr.sort(function(a, b) {
11   |   return b - a;
12 })
13 console.log(arr)
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\higherOrderFunction.js
[ 4, 5 ]
[ 5, 4, 3, 2, 1 ]
```

# Composability

- Function composition is a mechanism of combining multiple simple functions to build a more complicated one. The result of each function is passed to the next one.
- In mathematics, we often write something like:  $f(g(x))$ . So this is the result of  $g(x)$  that is passed to  $f$ . In programming we can achieve the composition by writing something similar.
- Let's take a quick example. Suppose I need to make some arithmetic by doing the following operation:  $2 + 3 * 5$ . As you may know, the multiplication has the priority over the addition. So you start by calculating  $3 * 5$  and then when add 2 to the result.

```
JS composition.js > ...  
1  const add = (a, b) => a + b;  
2  const mult = (a, b) => a * b;  
3  add(2, mult(3, 5))
```

## When We Use Composability

Let me explain with real time example,

- Think of an industrial plant that produce bottles of cool drinks; first there is the operation (or function)  $f1f1$  that puts the cool drinks inside the bottle, followed by the operation  $f2f2$  that close the bottle with the cap.
- In the above example we need to follow certain series of action, in the similar way we need to implement certain functionality which needs to be followed one after another

# Currying

- Currying is when you break down a function that takes multiple arguments into a series of function that each take only one argument.
- In the below example curryAdd function is returning a series of function and at the last function it is returning the value.

```
JS currying.js > ...
1 // Ordinary function for addition using 3 arguments
2 const add = (a, b, c) => {
3   return (a + b + c)
4 }
5
6 add(1, 2, 3); // 6
7
8 // Currying function for addition
9 const curryAdd = (sum) => {
10   return (a) => {
11     return (b) => {
12       return (c) => {
13         return sum(a, b, c);
14       }
15     }
16   }
17 }
18
19 const addition = curryAdd(add);
20 console.log(addition(1)(2)(3)); // 6
```



- Let me explain more simpler, If you want to buy a chocolate cake in shop.
- What is the process go out of the home -> take a bus -> find the shop -> check whether the chocolate cake is available or not -> if available then buy.
- In the above scenario, let us assume this as a task and we can split each and every in single function calling every function if everything is good we will be getting the cake but any one of the task is failed we won't get cake and get proper reason and it is easy to find.
- Instead of that if we put all task in a single function and passing multiple argument that is hard to manage, that's why currying comes into picture

# Anonymous Function

Anonymous functions in javascript are those functions that are created without any identifier or name to refer to it.

Normally we use the function keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the function keyword without the function name.

## Example

```
var myFunc = function() {  
    alert('Anonymous Function');  
}  
myFunc();
```

## Use of Anonymous Function

1. Anonymous functions can be used as arguments to other functions.

```
setTimeout(function() {  
  alert('relevel');  
}, 1000);
```

2. Anonymous functions are very useful in creating IIFE(Immediately Invoked Function Expression).

```
(function(text) {  
  alert(text);  
})('relevel');
```

3. Anonymous functions are commonly used for creating function expressions.

```
let myFunc = function(text){  
  alert('Inside Function expression');  
};
```

## Benefits of Named Function

Named function is very useful in javascript. Some of the benefits of using named functions are listed below:

1. Named functions are very helpful in debugging, in knowing which function caused an error, as you will get the function name in the error log.
2. Named functions are more readable, thus helping your code more understandable by other developers.
3. Named Functions are easier to reuse and thus helps you in writing clear code.

# Arrow Function

- Traditional function expressions are `function [name]([param1[, param2[, ..., paramN]]]) {`  
`statements`  
`}`

- The difference between named and unnamed functions are, If function name is omitted, it will be the variable name (implicit name). If function name is present, it will be the function name (explicit name).
- Unnamed functions are called as anonymous function.
- Arrow function is a different form of writing function compare to traditional function and it was introduced in the year 2015 ES6 (ECMAScript6) edition. They are less verbose than traditional function expression.
- Let's have a quick example and comparison of Arrow function with traditional function.
- Arrow functions are new way to write an anonymous function and are similar to Lambda function in other programming languages.
- Syntax: `(argument) => { ... Logic }`

## Difference Between Arrow and Regular function

	Regular Function	Arrow Function
Constructor	<pre>function Car(color) {   this.color = color; }  const redCar = new Car('red'); redCar instanceof Car; // =&gt; true</pre> <p>We can create an instance for Car</p>	<pre>const Car = (color) =&gt; {   this.color = color; };  const redCar = new Car('red'); // TypeError: Car is not a constructor</pre> <p>We cannot create an instance</p>
Argument Object	<pre>let user = {   show(){     console.log(arguments); // 1, 2, 3   } };  user.show(1, 2, 3);</pre> <p>Argument object are available</p>	<pre>let user = {   show_ar : () =&gt; {     console.log(...arguments); // error   } };  user.show_ar(1, 2, 3);</pre> <p>Argument object are not available</p>

## Difference Between Arrow and Regular function

	Regular Function	Arrow Function
this Keyword	<pre>let user = {   name: "Relevel",   regularfn(){     console.log("hello " + this.name); // 'this' binding     here   } }; user.regularfn(); this binding here</pre>	<pre>let user = {   name: "Relevel",   arrowfn:()=&gt;{     console.log("hello " + this.name); // no 'this' binding here   } }; user.arrowfn(); no this binding here</pre>
Implicit return	<pre>normalfn () {   12;   return; } normalfn(); // undefined</pre>	<pre>const arrowfn = () =&gt; 44 arrowfn(); // 44</pre>

## Programming Example

In this program, function **addTwoNumberTraditional** is the function expression we have seen in the class which is the traditional way of declaring the function.

The function **addTwoNumberArrow** is called as Arrow function because in the expression we are using `=>` and this function is assigning to a variable using `const` keyword, so hoisting will consider this as a variable.

```
JS arrowFunction.js > ...
1  // Traditional function
2  function addTwoNumberTraditional (a, b) {
3      return (a + b);
4  }
5
6  // Arrow function
7  const addTwoNumberArrow = (a, b) => {
8      return (a + b);
9  }
10
11 console.log(addTwoNumberTraditional(1, 2));
12
13 console.log(addTwoNumberArrow(1, 2))
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\arrowFunction.js
3
3
```



## Function – A first class citizen

Function in javascript are first class citizen which means you can store function in a variable, pass function as an argument, return function as a result.

```
JS firstClassCitizen.js > ...
1  // Storing function in a variable
2  const add = (a, b) => {
3    |    return (a + b);
4  |  }
5
6  const addition = add;
7  add(1, 2);
8  addition(1, 2);
9
10 // passing function as an argument
11 const pass = (func) => {
12 |    return func(1, 2);
13 |  }
14
15 pass(add);
16
17 // Return function as a result
18 const funcReturn = (a, b) => {
19 |    return () => {
20 |        |    console.log(a + b + 5);
21 |    }
22 |  }
23
24 funcReturn(1, 3)();
```

## IIFE (Immediately Invoked Function Expression)

- IIFE is a function that runs as soon as it is declared. Example will help you to understand what is IIFE.
- This is similar to declaring the function and invoking the function but only difference here is it will invoke as soon as it is declared.

```
JS IIFE.js
1  ✓ (() => {
2    |   console.log('IIFE, I am Invoked')
3    | } )();
4
5  // Output : IIFE, I am Invoked
```

# Advantages

## Secure Variables Scope

As you know var keyword scope is global so to secure the reference we can use IIF

```
(function () {
```

```
  var greeting = 'Good morning! How are you today?';
```

```
  console.log(greeting); // Good morning! How are you today?
```

```
})();console.log(greeting); // error: Uncaught ReferenceError: greeting is not defined
```

As you can see in the example above, what happens in the IIFE scope, stays in the IIFE scope. You can't use the variable defined inside IIFE from the outside.

## Avoid Naming Conflict

Using many JavaScript libraries can cause conflicts because some of them might export an object with the same name.

Let's say you're using jQuery. We all know it export \$ as its main object. So, if there's any library in your dependencies using \$ as its exported object as well, a conflict will occur.

Fortunately, you can use IIFEs to solve this problem by applying the aliasing technique:

```
(function ($) {
```

```
  // You're safe to use jQuery here
```

```
})(jQuery);
```

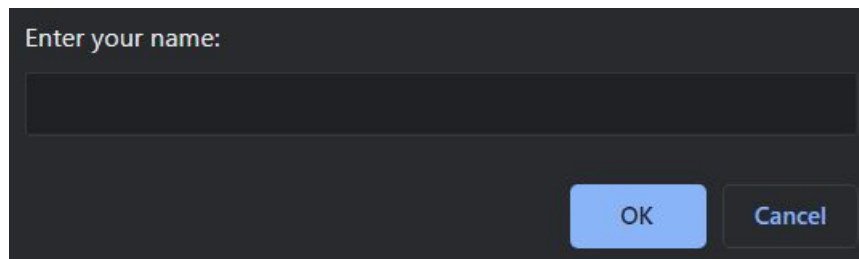
By wrapping your code inside an IIFE that takes jQuery as an argument, we will make sure that the \$ symbol now refers to jQuery, not other libraries.

## Taking user input in JS

- We can take user input in various way using javascript, but as a beginner we should be aware of the prompt which will get user input and do the logic as per the next process.
- JavaScript has a few window object methods that you can use to interact with your users. The prompt() method lets you open a client-side window and take input from a user.

```
const name = window.prompt("Enter your name: ");  
alert("Your name is " + name);
```

Window object will prompt for user input and asking the user to enter name

A screenshot of a JavaScript prompt dialog box. It has a dark background with the text "Enter your name:" in white. Below the text is a white input field. At the bottom right, there are two buttons: "OK" and "Cancel".

Your name is Saravanan N

OK

As per the 2nd line code the will show the name which user had given as an alert.



## Get Input using Node

- To access input from user, you need to create an Interface instance that is connected to an input stream.
- You create the Interface using `readline.createInterface()` method, while passing the input and output options as an object argument.

```
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.question("What is your name? ", function (answer) {
  console.log(`Oh, so your name is ${answer}`);
  console.log("Closing the interface");
  rl.close();
});
```

## Try this Question

Implement the mutation using object in javascript.

- Given object `const obj = {`  
    `Javascript: 'hard',`  
    `Java: 'easy',`  
    `Python: 'medium'`  
    `}`

Change property of Javascript from 'hard' to 'easy'

Solution: <https://jsfiddle.net/saravananslb/fhr4Lq5o/1/>

Explanation: Object property can be access using '.' and directly assign the value.



## Try this Question

Write a program array containing string and sort the string array in descending order. (Input array: ['a', 'b', 'c', 'z', 'h'])

Solution : <https://jsfiddle.net/saravananslb/sv46jLdf/1/>

Explanation: sort function will accept comparator passing localeCompare and comparing the string



## Try this Question

Write a program to find the particular element in an array using find function. (Input array: ['a', 'b', 'c', 'z', 'h']) find 'c'

Solution: <https://jsfiddle.net/saravananslb/1mjnqawh/1/>

Explanation: find function will return the value if the array satisfies the given condition else undefined.





## Try this Question

What will be the output of the below code ?

```
// Ordinary function for multiplication using 3 arguments
const multiply = (a, b, c) => {
  return (a * b * c)
}

// Currying function for multiplication
const currymultiply = (multi) => {
  return (a) => {
    return (b) => {
      return (c) => {
        return multi(a, b, c);
      }
    }
  }
}

const multiplication = currymultiply(multiply);
console.log(multiplication(1)(2)(3));
```

**Solution:** output is 6

Explanation: currymultiply function is called with multiply function as argument and invoking the series of function with 1, 2 and 3, at last it is returning the argument of 1<sup>st</sup> function value as function and referring the series of value passed from the function as an argument of multi function



## Try this Question

What will be the output of the below code ?

```
const obj = {  
  Javascript: 'hard',  
  Java: 'easy',  
  Python: 'medium'  
}  
  
const newObj = obj;  
  
newObj.Javascript = 'easy';  
obj.Javascript = 'very easy';  
console.log(newObj.Javascript);
```

Solution: very easy

Explanation: Reference of obj is assigned to newObj, so any change in obj or newObj will make a change in both variable because both variables are referring a same memory location.



## Try this Question

Write a function which will assign object to another variable change of object property will not affect the object property of another variable.

### Solution Code:

```
const obj = {  
  Javascript: 'hard',  
  Java: 'easy',  
  Python: 'medium'  
}  
  
// Destructuring the object  
const newObj = {...obj};  
  
newObj.Javascript = 'easy';  
obj.Javascript = 'very easy';  
console.log(newObj);  
console.log(obj);
```

**Explanation:** De-structuring the object will make an copy of the object, so changing the property of one object won't affect the property of another object



## Try this Question

What is the output of the below code?

```
const arrayOfOddNumbers = [1, 3, 5];  
arrayOfOddNumbers[100] = 199;  
console.log(arrayOfOddNumbers.length);
```

### Solution : 101

Explanation: The reason for this solution is as follows: JavaScript places empty as a value for indices 3 - 99. Thus, when you set the value of the 100th index



## Assignment

1. Write a program to multiply the value in the given array and return a result (use array functions) Array = [1, 2, 3, 4, 5, 6, 7]
2. Write a JavaScript program to sort by id an array of JavaScript objects.

```
Object =[ {  
    Id: 45,  
    Name: 'ram'  
  }, {  
    Id: 4,  
    Name: 'raju'  
  }, {  
    Id: 90,  
    Name: 'kumar'  
  }  
]
```

3. Write a program to get an input ('How are you') and by default it should be good in the prompt text box.



**Thank you**