**Function Currying in JS**

With the rapid rise of functional programming in JavaScript, a lot of attention has been given to the curried function. So in this post, let's discuss this.  
  
**Q.** What is Function Currying ❓  
  
Currying is a transformation that can be applied to functions to allow them to take one less argument than previously.  
  
Okay okay, let's simplify this  
   
Currying is when a function, rather than taking all arguments at once, takes the first one and returns a new function, which then takes the second one and returns a new function, and so on until all arguments are completed.  
  
**Q.** Uses and Need of Function Currying ❓  
  
➡ It helps us to avoid passing the same variable again and again.  
   
➡ It separates our function into several smaller functions, each of which can handle a single duty. This ensures that our function is error-free and free of side effects.  
   
➡ It is used in functional programming to create a higher-order function.  
  
➡ It is extremely useful in event handling.  
  
  
**Q.** How can we do Function Currying ❓  
  
There are multiple ways by which we can do function currying, in this post we will discuss one of that methods.  
  
⭐ Function Currying using Closures  
  
function add (a) {  
return function (b) {  
console.log ( a + b );  
};  
}  
add(2)(5);  
  
Confused❓, Let's discuss  
  
Here, the result of add(a) is the wrapper function(b).  
   
When it's called add(2), the argument is saved in the Lexical Environment, and a new wrapper is returned function(b).  
   
Then this wrapper is called with ‘5’ as an argument, this wrapper will already have access to ‘a’ due to closure, which is currently equal to ‘2’, so we will get ‘7’ printed on the console.  
  
  
Currying is a difficult concept to grasp. But, with time and practice, we can certainly get the hang of it.