**Recursion**

**What is recursion?**A process (a function in our case) that calls itself.

**Why do we need to know about recursion?**

It's EVERYWHERE!

* JSON.parse / JSON.stringify
* document.getElementById and DOM traversal algorithms
* Object traversal
* Very common with more complex algorithms
* It's sometimes a cleaner alternative to iteration

**How recursive function works?**Invoke the same function with a different input until you reach your base case!  
**Base Case** means, The condition when the recursion ends.

**The Call Stack**It’s a stack data structure that means is that, things are added to the top and remove first from the top.  
Any time a function is invoked it is placed (**pushed)** on the top of the call stack.  
When JavaScript sees the return keyword or when the function ends, the compiler will remove (**pop**).

**Note:**

* You’re used to functions being pushed on the call stand and popped off when they are done.
* When we write recursive functions, we keep pushing new functions onto the call stack!
* We can visualize the recursive function with the Chrome Dev Tool.

**Two essential parts of a recursive function!**

1. Base Case
2. Different Input

**Example-1: *//Our first recursive function***

function countDown(num){  
 if(num <= 0) {  
 console.log("All done!");  
 return; ***//Final Base Case***  
 }

console.log(num);  
 num--;  
 countDown(num); ***//Function calling itself***

}   
countDown(5);

**Output:**5  
4  
3  
2  
1  
6   
All done!

**Example-2: *//Our Second recursive function***/\***Explanation**(Follow First Top to Bottom then Bottom to Top ):-

sumRange(5) //return value: 15 (Final Base Case)  
 return 5 + sumRange(4) //return 5 + 10: 15 to sumRange(5)  
 return 4 + sumRange(3) //return 4 + 6: 10 to sumRange(4)  
 return 3 + sumRange(2) //return 3 + 3: 6 to sumRange(3)  
 return 2 + sumRange(1) //return 2 + 1: 3 to sumRange(2)  
 return 1 //Got Return: 1 to sumRange(1) (Starting Base Case)

function sumRange(num){

if(num === 1) return 1; ***// Firs/Starting Base Case***   
 return num + sumRange(num-1); ***//Final Base Case***

}

sumRange(5);

**Output:**15

**Example-3:** ***//Factorial of a Number***

**//Writing Iteratively**

function factorial(num){

if(num===1){  
 return 1;  
 }

return num \* factorial(num-1);

}

factorial(5);

**Output:**120

**Example-3:** ***//Factorial of a Number***

**//Writing Iteratively**

function factorial(num){  
 let total = 1;

for(let i=num; i>1; i--){  
 total \*=i;  
 }

return total;

}

factorial(5);

**Output:**120

**Example - 4:** ***//Factorial of a Number***

**//Writing Recursively**

function factorial(num){

if(num===1){  
 return 1;  
 }

return num \* factorial(num-1);

}

factorial(5);

**Output:**120

**Common Recursion Pitfalls**

**Where things go Wrong:**

* No base Case or Wrong base case
* Forgetting to return or returning the Wrong thing!
* Result: Call Stack Overflow or Call Stack Size excedded

**Ex-1:**function factorial(num){

if(num===1){  
 return 1;  
 }

return num \* factorial(num); ***//returning the wrong thing***

}

factorial(5);

**Output:**Uncaught RangeError: Maxium Call Stack Size excedded

**Ex-2:**function factorial(num){

if(num===1){  
 console.log(1); ***//No base Case***  
 }

return num \* factorial(num);

}

factorial(5);

**Output:**Uncaught RangeError: Maxium Call Stack Size excedded

**Helper Method Recursion:**

Helper Method is a design pattern that’s commonly used with recursion called helper method.

Point to be noted in Helper Method:

* Make a result variable
* Define a function insider outer function.
* Inner function would be recursive whereas Outer function wouldn’t be recursive.

**Pattern of Helper Method:**

function outer(input){  
 var outerScopedVariable = [];

function helper(helperInput){  
 //modify the outerScopedVariable  
 helper(helperInput--)  
 }

helper(input);

return outerScopedVariable;

}

**Example -1: *//Collect Odd Values of an Array***

**//Using Helper Method Recursion**function collectOddValues(arr){

let result = [];

function helper(helperInput){  
 if(helperInput.length===0){  
 return; ***//Base Case***  
 }

if(helperInput[0]%2!==0){  
 result.push(helperInput[0]);  
 }

helper(helperInput.slice(1)); ***//Recursion* //slice() method gives everything from the given number of element onwards.**

}

helper(arr);

return result;

}

collectOddValues([1,2,3,4,5,6,7,8]);

**Output:**[1, 3, 5, 7]

**Example -2: *//Collect Odd Values of an Array***

**//Using Pure Recursion**

/\* **Explanation:- (Follow Firstly Top to Bottom then Bottom to Top)**

Final Base Case : return newArr; (At Bottom)  
newArr = newArr.concat(collectOddValues(arr.slice(1)));  
Input: [1,2,3,4,5]

newArr.concat(collectOddValues(arr.slice(1))) //newArr = [1,3,5]

[1].concat(collectOddValues([2,3,4,5])) //[1].[3,5] ==> [1,3,5]  
 [].concat(collectOddValues([3,4,5])) //[].[3,5] ==> [3,5]  
 [3].concat(collectOddValues([4,5])) //[3].[5] ==> [3,5]  
 [].concat(collectOddValues([5])) //[].[5] ==> [5]  
 [5].(collectOddValues([]))   
//return newArr: [] 🡪(collectOddValues([])), [5].[] ==> [5] 🡪(collectOddValues([5]))

\*/

function collectOddValues(arr){

let newArr = []; ***//Since, we're going to call the function Recursively, newArr would be reset to empty everytime.***

if(arr.length===0){

return newArr;  ***//First / Start Base Case***

}

if(arr[0]%2!=0){

newArr.push(arr[0]);

}

newArr = newArr.concat(collectOddValues(arr.slice(1)));

return newArr; ***//Last/Final Base Case***

}

collectOddValues([1,2,3,4,5]);

**Output:**[1,3,5]

**Pure Recursion Tips:-**

* For Arrays, use methods like **slice**, the **spread operator**, and **concat** that make copies of arrays so you do not mutate them.
* Remember that strings are immutable so you will need to use methods like **slice, substr, or substring** to make copies of strings.
* To make copies of objects use **Object.assign, or the spread operator**.

**Recursion Problem Set**

**Coding Exercise-1:**

Write a function called power which accepts a base and an exponent.   
The function should return the power of the base to the exponent. This  
function should mimic the functionality of Math.pow()

**Test Cases:**

power(2,0) // 1  
power(2,2) // 4  
power(2,4) // 16

**//Iterative Way**function power(bs, ex){

let pw=1;

for(let i = ex;i>0;i--)  
 {  
 pw \*=bs;   
 }

return pw;

}

power(2, 4);

**Output:**  
16

**//Recusive Way**

**Explaination:**

bs\*power(bs, ex-1) pw=16

2\*power(2,3) //2\*8=16  
 2\*power(2,2) //2\*4=8  
 2\*power(2,1) //2\*2=4  
 2\*power(2,0) //2\*1=2  
 2\* return1

\*/

function power(bs, ex){

let pw;

if(ex===0) return 1; ***//Base Case***

pw = bs\*power(bs, ex-1);

return pw;

}

power(2,0);

**Output:**  
16

**Coding Exercise-2:**

Write a function factorial which accepts a number and returns the factorial of that number. A factorial is the product of an integer and all the integers below it; e.g., factorial four(4!) is equal to 24, because 4\*3\*2\*1 equals 24. factorial zero(0!) is always 1.)

**Test Cases:**

factorial(1) // 1  
factorial(2) // 2  
factorial(4) // 24  
factorial(7) // 5040

**//Iterative Way**function factorial(num){

let factNum = 1;

for(let i=num; i>1; i--){

factNum \*= i;

}

return factNum;

}

factorial(7);

**Output:**  
5040

**//Recursive Way**

/\*

num \* factorial(num - 1) //return 24

4 \* factorial(3) //4\*6= 24

3 \* factorial(2) //3\*2 = 6

2 \* factorial(1) // 2\*1 = 2

1 \* factorial(0); // 1\*1 = 1

return 1;

\*/

function factorial(num){

if(num===0){

return 1;

}

return num \* factorial(num - 1);

}

factorial(7);

**Output:**  
5040

**Coding Exercise-3:**

Write a function called productOfArray which takes in an array of numbers and returns the product of them all.

**Test Cases:**

productOfArray([1,2,3]) // 6

productOfArray([1,2,3,10]) // 60

**//Iterative Way**function productOfArray(arr){

let arrPrdt = 1;

for(let i=0; i<arr.length; i++){

arrPrdt \*=arr[i];

}

return arrPrdt;

}

productOfArray([1,2,3]);

**Output:**  
6

**//Recursive Way**

/\*Explanation:-

arr[0] \* productOfArray(arr.slice(1)) //return 60

1 \* productOfArray([2,3,10]) //1\*60 = 60

2 \* productOfArray([3,10]) //2\*30 = 60

3 \* productOfArray([10]) //3\*10 = 30

10 \* productOfArray([]) //10\*1 = 10

return 1;

\*/

function productOfArray(arr){

if(arr.length===0){

return 1;

}

return arr[0] \* productOfArray(arr.slice(1));

}

productOfArray([1,2,3,10]);

**Output:**  
60

**Coding Exercise-4:**

Write a function called recursiveRange which accepts a number and adds up all the numbers from 0 to the number passed to the function.

**Test Cases:**

recursiveRange(6) // 21  
recursiveRange(10) // 55

**//Iterative Way**function recursiveRange(num){

let numSum = 0;

for(let i = num; i>=0; i--){

numSum += i;

}

return numSum;

}

recursiveRange(10);

**Output:**  
55

**//Recursive Way**

/\*Explanation:-

num + recursiveRange(num-1) //return 21

6 + recursiveRange(5) //6+15 = 21

5 + recursiveRange(4) //5+10 = 15

4 + recursiveRange(3) //4+6 = 10

3 + recursiveRange(2) //3+3 = 6

2 + recursiveRange(1) //2+1 = 3

1 + recursiveRange(0) //1+0 = 1

return 0;

\*/

function recursiveRange(num){

if(num===0){

return 0;

}

return num + recursiveRange(num-1);

}

recursiveRange(10);

**Output:**  
55

**BONOUS CHALLENGING RECURSION PROLEM SET**

**Coding Exercise-1:**

Write a function called reverse which accepts a string and returns a new string in reverse.

**Test Cases:**

reverse('awesome') // 'emosewa'  
reverse('rithmschool') // 'loohcsmhtir'

**//Iterative Way**function reverse(str){

let newStr = '';

for(let i=str.length-1; i>=0; i--){

newStr = newStr.concat(str[i]);

}

return newStr;

}

reverse('awesome');

**Output:**  
'emosewa'

**//Recursive Way**

**Explaination:**

str[str.length-1]+ reverse(str.substring(0, str.length-1))

e + reverse('awesom') //e+mosewa = 'emosewa'

m + reverse('aweso') //m+osewa = 'mosewa'

o + reverse('awes') //o+sewa = 'osewa'

s + reverse('awe') //s+ewa = 'sewa'

e + reverse('aw') //e+wa = 'ewa'

w + reverse('a') //w+a = 'wa'

return str[0] //return 'a'

\*/

function reverse(str){

if(str.length===1){

return str[0] ;

}

return str[str.length-1]+ reverse(str.substring(0, str.length-1));

}

reverse('rithmschool');

**Output:**  
'loohcsmhtir'