Creating Factorials

**Algorithm Challenge**

Return the factorial of the provided integer.  
  
If the integer is represented with the letter n, a factorial is the product of all positive integers less than or equal to n.  
  
Factorials are often represented with the shorthand notation **n!**  
  
For example: **5! = 1 \* 2 \* 3 \* 4 \* 5 = 120**

function factorialize(num) {

return num;

}

factorialize(5);

***Provided test cases***

* ***factorialize(0)*** should return 1
* ***factorialize(5)*** should return 120
* ***factorialize(10)*** should return 3628800
* ***factorialize(20)*** should return 2432902008176640000

**What is factorializing a number all about?**

When you factorialize a number, you are multiplying that number by each consecutive number minus one.

If your number is 5, you would have:

5! = 5 \* 4 \* 3 \* 2 \* 1

The pattern would be:

0! = 1

1! = 1

2! = 2 \* 1

3! = 3 \* 2 \* 1

4! = 4 \* 3 \* 2 \* 1

5! = 5 \* 4 \* 3 \* 2 \* 1

**1. Factorialize a Number With Recursion**

function factorialize(num) {

// If the number is less than 0, reject it.

if (num < 0)

return -1;

// If the number is 0, its factorial is 1.

else if (num == 0)

return 1;

// Otherwise, call the recursive procedure again

else {

return (num \* factorialize(num - 1));

/\*

First Part of the recursion method

You need to remember that you won’t have just one call, you’ll have several nested calls

Each call: num === "?" num \* factorialize(num - 1)

1st call – factorialize(5) will return 5 \* factorialize(5 - 1) // factorialize(4)

2nd call – factorialize(4) will return 4 \* factorialize(4 - 1) // factorialize(3)

3rd call – factorialize(3) will return 3 \* factorialize(3 - 1) // factorialize(2)

4th call – factorialize(2) will return 2 \* factorialize(2 - 1) // factorialize(1)

5th call – factorialize(1) will return 1 \* factorialize(1 - 1) // factorialize(0)

Second part of the recursion method

The method hits the if condition, it returns 1 which num will multiply itself with

The function will exit with the total value

5th call will return (5 \* (5 - 1)) // num = 5 \* 4

4th call will return (20 \* (4 - 1)) // num = 20 \* 3

3rd call will return (60 \* (3 - 1)) // num = 60 \* 2

2nd call will return (120 \* (2 - 1)) // num = 120 \* 1

1st call will return (120) // num = 120

If we sum up all the calls in one line, we have

(5 \* (5 - 1) \* (4 - 1) \* (3 - 1) \* (2 - 1)) = 5 \* 4 \* 3 \* 2 \* 1 = 120

\*/

}

}

factorialize(5);

**Without comments:**

function factorialize(num) {

if (num < 0)

return -1;

else if (num == 0)

return 1;

else {

return (num \* factorialize(num - 1));

}

}

factorialize(5);

**2. Factorialize a Number with a WHILE loop**

function factorialize(num) {

// Step 1. Create a variable result to store num

var result = num;

// If num = 0 OR num = 1, the factorial will return 1

if (num === 0 || num === 1)

return 1;

// Step 2. Create the WHILE loop

while (num > 1) {

num--; // decrementation by 1 at each iteration

result = result \* num; // or result \*= num;

/\*

num num-- var result result \*= num

1st iteration: 5 4 5 20 = 5 \* 4

2nd iteration: 4 3 20 60 = 20 \* 3

3rd iteration: 3 2 60 120 = 60 \* 2

4th iteration: 2 1 120 120 = 120 \* 1

5th iteration: 1 0 120

End of the WHILE loop

\*/

}

// Step 3. Return the factorial of the provided integer

return result; // 120

}

factorialize(5);

**Without comments:**

function factorialize(num) {

var result = num;

if (num === 0 || num === 1)

return 1;

while (num > 1) {

num--;

result \*= num;

}

return result;

}

factorialize(5);

**3. Factorialize a Number with a FOR loop**

function factorialize(num) {

// If num = 0 OR num = 1, the factorial will return 1

if (num === 0 || num === 1)

return 1;

// We start the FOR loop with i = 4

// We decrement i after each iteration

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

// We store the value of num at each iteration

num = num \* i; // or num \*= i;

/\*

num var i = num - 1 num \*= i i-- i >= 1?

1st iteration: 5 4 = 5 - 1 20 = 5 \* 4 3 yes

2nd iteration: 20 3 = 4 - 1 60 = 20 \* 3 2 yes

3rd iteration: 60 2 = 3 - 1 120 = 60 \* 2 1 yes

4th iteration: 120 1 = 2 - 1 120 = 120 \* 1 0 no

5th iteration: 120 0 120

End of the FOR loop

\*/

}

return num; //120

}

factorialize(5);

**Without comments:**

function factorialize(num) {

if (num === 0 || num === 1)

return 1;

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

num \*= i;

}

return num;

}

factorialize(5);

I hope you found this helpful. This is part of my “How to Solve FCC Algorithms” series of articles on the Free Code Camp Algorithm Challenges, where I propose several solutions and explain step-by-step what happens under the hood.