SOFT CLUB

# Tasks about “Function” in Java Script

# Recurtion

# Recursion: Factorials

Published by Matt in JavaScript

complete numbers recursion

Write a function that calculates the **factorial** of a number **recursively**.

## Examples

```
factorial(5) → 120

factorial(3) → 6

factorial(1) → 1

factorial(0) → 1
```

# Double Factorial

Published by Matt in JavaScript

complete numbers recursion

Create a function that takes a number `num` and returns its **double factorial**.

## Examples

```
doubleFactorial(0) → 1

doubleFactorial(2) → 2

doubleFactorial(9) → 945
// 9*7*5*3*1 = 945

doubleFactorial(14) → 645120
```

## Notes

- Assume all input values are greater than or equal to `-1`.
- Try to solve it with recursion.
- Double factorial is not the same as factorial * 2.

# Recursion: Sum

Published by Matt in JavaScript

complete | language_fundamentals | math | numbers | recursion

Write a function that finds the sum of the first `n` natural numbers. **Make your function recursive.**

## Examples

```
sum(5) → 15
// 1 + 2 + 3 + 4 + 5 = 15

sum(1) → 1

sum(12) → 78
```

# Recursion: Length of a String

Published by Matt in JavaScript

complete | language_fundamentals | recursion | strings

Write a function that returns the length of a string. **Make your function recursive.**

## Examples

```
length("apple") → 5

length("make") → 4

length("a") → 1

length("") → 0
```

# Recursion: Fibonacci Numbers

Published by Matt in JavaScript

complete numbers recursion

Fibonacci numbers are created in the following way:

```
F(0) = 0
F(1) = 1
...
F(n) = F(n-2) + F(n-1)
```

Write a function that calculates the `nth` Fibonacci number.

## Examples

```
fib(0) → 0

fib(1) → 1

fib(2) → 1

fib(8) → 21
```

# Closure

# A Redundant Function

Published by Helen Yu in JavaScript

complete | closures | functional_programming | language_fundamentals

Write a function `redundant` that takes in a string `str` and returns a function that returns `str`.

## Examples

```
const f1 = redundant("apple")
f1() → "apple"

const f2 = redundant("pear")
f2() → "pear"

const f3 = redundant("")
f3() → ""
```

# All About Anonymous Functions: Adding

Published by bangyen in JavaScript

complete | closures | higher_order_functions | language_fundamentals

Write a function that returns an **anonymous function**, which adds `n` to its input

## Examples

```
adds1 = addsNum(1)

adds1(3) → 4
adds1(5.7) → 6.7

adds10 = addsNum(10)

adds10(44) → 54
adds10(20) → 30
```

# Returning an "Add" Function

Published by Jon Ingram in JavaScript

complete | closures | higher_order_functions | language_fundamentals | numbers

Given a number, `n`, return a function which adds `n` to the number passed to it.

## Examples

```
add(10)(20) → 30

add(0)(20) → 20

add(-30)(80) → 50
```

# All About Anonymous Functions: Adding Suffixes

Published by bangyen in JavaScript

complete | closures | higher_order_functions | language_fundamentals

Write a function that returns an **anonymous function**, which transforms its input by adding a particular `suffix` at the end.

## Examples

```
add_ly = add_suffix("ly")

add_ly("hopeless") → "hopelessly"
add_ly("total") → "totally"

add_less = add_suffix("less")

add_less("fear") → "fearless"
add_less("ruth") → "ruthless"
```

# Function Times 3

Published by BijogFc24 in JavaScript

complete | closures | functional_programming | language_fundamentals | scope

Create a function that takes three collections of arguments and returns the sum of the product of numbers.

## Examples

```
product(1,2)(1,1)(2,3) → 8
// 1 * 1 * 2 + 2 * 1 * 3

product(10,2)(5,0)(2,3) → 100
// 10 * 5 * 2 + 2 * 0 * 3

product(1,2)(2,3)(3,4) → 30
// 1 * 2 * 3 + 2 * 3 * 4

product(1,2)(0,3)(3,0) → 0
// 1 * 0 * 3 + 2 * 3 * 0
```