**Заміна циклів за допомогою рекурсії**

Рекурсія - це концепція, згідно з якою функція може бути вираженою в термінах сама по собі. Щоб краще зрозуміти це, почніть з роздумів про наступне завдання: перемножте перші n елементів масиву, щоб створити добуток цих елементів. Ви могли б це зробити, використовуючи цикл for:

function multiply(arr, n) {

let product = 1;

for (let i = 0; i < n; i++) {

product \*= arr[i];

}

return product;

}

Проте, зверніть увагу, що multiply(arr, n) == multiply(arr, n - 1) \* arr[n - 1]. Це означає, що ви можете перезаписати multiply саме по собі і ніколи не використовувати цикл.

function multiply(arr, n) {

if (n <= 0) {

return 1;

} else {

return multiply(arr, n - 1) \* arr[n - 1];

}

}

Рекурсивна версія multiply переривається таким чином. У *base case*, де n <= 0, повертає до 1. Для більших значень n, він викликає сам себе, але з n - 1. Цей виклик функції обчислюється так само, викликаючи multiply знову до n <= 0. На даному етапі, всі функції можуть повертати результат і початковий multiply повертає відповідь.

**Примітка:** Рекурсивні функції повинні мати базовий регістр, коли вони повертають результат, не викликаючи функцію знову (в цьому прикладі, коли n <= 0), інакше вони ніколи не зможуть закінчити виконання.

Напишіть рекурсивну функцію, sum(arr, n), що повертає суму перших n елементів масиву arr.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

**Tests**

* В очікуванні:sum([1], 0) повинен дорівнювати 0.
* В очікуванні:sum([2, 3, 4], 1) повинен дорівнювати 2.
* В очікуванні:sum([2, 3, 4, 5], 3) повинен дорівнювати 9.
* В очікуванні:Ваш код не має опиратися на жодні цикли (for, while або функції більш високої послідовності, такі як forEach, map, filter або reduce.).
* В очікуванні:Вам слід використати рекурсію, щоб вирішити цю проблему.
* function sum(arr, n) {
* if(n <= 0) {
* return 0;
* } else {
* return sum(arr, n - 1) + arr[n - 1];
* }
* }