UD 06.
PROGRAMACIÓN FUNCIONAL

Desarrollo Web en entorno cliente
CFGS DAW

Ejercicios

Álvaro Maceda Arranz
alvaro.maceda@ceedcv.es
2021/2022
Versión:211209.1154

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

ÍNDICE DE CONTENIDO

UD06. Programación Funcional

Esta semana los ejercicios son más cortos, pero puede que conceptualmente te resulten un poco más complejos.

> Recuerda crear los tests para los ejercicios

1. Ejercicio 1: Purificación

Modifica la función `mergeValues` para que sea una función pura.

```
const MINIMUM = 15;

function mergeValues(arrayOfIntegers) {
  let element;
  let sum = 0;

  while(element = arrayOfIntegers.pop()) {
    sum += element
  }

  sum = Math.max(sum, MINIMUM)
  arrayOfIntegers.push(sum);

  return arrayOfIntegers;
}

console.log(mergeValues([10,20,30,40])) // [100]
console.log(mergeValues([1,2,3,4])) // [15] (MINIMUM)
```

2. Ejercicio 2: Idés

Crea una función que permita crear funciones para generar IDs. Las funciones devueltas generarán una cadena de la longitud definida cuando se invoquen. La cadena se irá incrementando con cada invocación.

Ejemplos:

```
const len3Id = createIDGenerator(3);
len3Id() // 001
len3Id() // 002
len3Id() // 003

const len5Id = createIDGenerator(5);
len5Id() // 00001
```

> Date cuenta de que las funciones que crea `createIDGenerator` no son funciones puras, ya que devuelven diferentes valores con los mismos parámetros de entrada

3. Ejercicio 3: Multiplicación perezosa

Crea una función llamada `lazyMultiply` para multiplicar dos números. Si a la función se le pasan dos parámetros devolverá inmediatamente la solución. Si a la función se le pasa un único parámetro devolverá una función que, al pasarle un segundo parámetro, devolverá el resultado de la multiplicación.

Ejemplos:

```
lazyMultiply(7,4) // 28

const perTwo = lazyMultiply(2)
perTwo(3) // 6

lazyMultiply(5)(10) // 50
```

4. Ejercicio 4: Pipas

Crea una función llamada `doublePipe` que admita un número variable de funciones y devuelva una función nueva que aplique esas funciones en el mismo orden en que se han pasado, pero aplicando dos veces cada una de las funciones.

Por ejemplo:

```
function double(x) { return x*2 }
function add3(x) { return x+3 }

let multiplyPerFourAndAddSix = doublePipe(double, add3)
console.log(multiplyPerFourAndAddSix(10)) // 46 = (10*2*2+3+3)

let addSixAndMultiplyPerFour = doublePipe(add3, double)
console.log(addSixAndMultiplyPerFour(10)) // 64 = (10+3+3)*2*2
```

Las funciones deben admitir un sólo parámetro.

> Recuerda que puedes utilizar el operador spread para manejar un número variable de parámetros en una función

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- Álvaro Maceda Arranz