Universidad del Valle

Práctica 5

Fecha límite de entrega: 31 de marzo de 2021

Desarrolle los siguientes ejercicios los cuales se deben tener en cuenta funciones, condicionales y recursividad utilizando el paradigma funcional y lenguaje de programación JavaScript. Usted debe enviar el código fuente y pasar los test a través de la plataforma INGInious M-IDEA (http://ingin.ddns.net/courselist). Puede apoyarse de la herramienta repl.it (https://repl.it/~)

Documento de repaso

https://docs.google.com/document/d/1kWkNKLZ9dak7lEcAwWuF0Eq4uah1AJ4UhzOO9Oz0G_U/edit?usp=sharing

IMPORTANTE

Una función es recursiva cuando el cuerpo de la función utiliza a la propia función. Dentro de una función recursiva suelen distinguirse dos partes:

- Los casos base: Son aquellos que para su solución no requieren utilizar la función que se está definiendo.
- Los casos recursivos: Son aquellos que sí que requieren utilizar la función que se está definiendo.

Las definiciones recursivas funcionan siempre y cuando las llamadas recursivas se realicen de forma que en algún momento se lleguen a los casos base.

Ejemplo

Para calcular el factorial de un número se distinguen dos casos: si el número es cero o si es mayor. El primer caso es un caso base, pues sabemos que la solución es 1, mientras que para el resto de los casos utilizaremos una llamada recursiva. La distinción de casos puede realizarse por cualquiera de los 4 métodos que conocemos. Veámoslo por ejemplo con el uso de patrones:

```
fact 0 = 1

fact n = n * fact (n-1)
```

1. El procedimiento se llama a sí mismo
2. El problema se resuelve, tratando el mismo problema pero de tamaño menor
3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará

Universidad del Valle

1. Factorial de un Número

Escribir un programa para calcular el factorial de un número

código/solución

```
function factorial(n) {
    if (n <= 0) {
        return 1;
    } else {
        return n * factorial(n - 1)
    }
}
```

Comprueba las salidas

Entradas

```
factorial(0)
factorial(1)
factorial(10)
factorial(12)
factorial(18)
```

Salidas

```
?
?
3628800
?
6402373705728000
```

2. Sumatoria

En matemáticas es común usar una notación para indicar múltiples operaciones, por ejemplo la sumatoria que representa la suma de varios elementos de manera recursiva. Implemente una función *sumatoria(i)* que haga la sumatoria de números naturales de m hasta n de:

$$sumatoria(i) = \sum_{i=m}^{n} a_i = a_m + a_{m+1} + a_{m+2} + \ldots + a_n$$

código/solución 1

```
function sumatoria(m, n) {
    if (m == n) {
        return n;
    } else {
        return m + sumatoria(m + 1, n);
    }
}
```

código/solución 2

```
function sumatoria(m, n) {
    if (n == m) {
        return m;
    } else {
        return n + sumatoria(m, n - 1);
    }
}
```

Comprueba las salidas

Entradas

```
sumatoria(1,4)
sumatoria(2,4)
sumatoria(8,9)
sumatoria(0,100)
```

Salidas

```
10
9
17
?
```

3. Sumatoria X formula

En matemáticas es común usar una notación para indicar múltiples operaciones, por ejemplo la sumatoria que representa la suma de varios elementos de manera recursiva. Implemente una función *sumatoriaXCuadrado(i)* que haga la sumatoria de m hasta n de:

$$sumatoriaXCuadrado(i) = \sum_{i=m}^{n} (i^2 + 7)^2$$

código/solución 1

```
function sumatoriaXCuadrado(m, n) {
    if (n < m) {
        return 0
    } else {
        return (Math.pow(Math.pow(n, 2) + 7, 2)) + sumatoriaXCuadrado(m, n - 1)
    }
}
```

Comprueba las salidas

Entradas

```
sumatoriaXCuadrado(1,3)
sumatoriaXCuadrado(2,4)
sumatoriaXCuadrado(8,9)
sumatoriaXCuadrado(0,100)
```

Salidas

```
441
906
12785
2055075179
```

4. Fibonacci

Escribir un programa para calcular la sucesión de Fibonacci dado un número. **NOTA:** NO se permite el uso de estructuras iterativas (for, while,do..While).

https://www.youtube.com/watch?v=dxyYP3BSdcQ

código/solución 1

```
function fibonacci(n) {
    if (n == 0) {
        return (0)
    } else if (n <= 1) {
        return (1)
    } else {
        return (fibonacci(n - 1) + fibonacci(n - 2))
    }
}
```

Comprueba las salidas

Entradas

```
fibonacci(0)
fibonacci(1)
fibonacci(9)
fibonacci(15)
```

Salidas

```
0
1
34
?
```

5. Multiplicación de 2 números enteros positivos

Para todos nosotros es muy bien sabido que una multiplicación es una operación binaria (i.e., entre dos números) en un conjunto numérico. Ahora bien, la multiplicación a*b consiste en calcular el resultado (producto) de sumar un mismo número “a” (multiplicando) tantas veces como indica otro número “b” (multiplicador); Por ejemplo, 4×3 es igual a sumar tres veces el valor 4 por sí mismo (4+4+4).

Implemente la función **multiplicacionRecursiva(multiplicando,multiplicador)** que represente la operación de multiplicar de forma recursiva dos valores enteros positivos mayores a cero.

Entrada Ejemplo 1

```
multiplicaciónRecursiva(3,4)
multiplicaciónRecursiva(102,17)
multiplicaciónRecursiva(32,114)
```

Salida Ejemplo 1

```
12
?
?
```

6. Función recursiva esParRecursivo

En el curso se ha visto una forma sencilla de verificar si un número es par o impar utilizando el operador %. No obstante, es posible definir de manera recursiva si un número entero positivo es par o impar. Implemente la función **esParRecursivo(numero)** que represente la operación de verificar si un número es par o impar de forma recursiva (piense que deberá llegar a un caso base para decidir si el número dado es par o impar).

Entrada Ejemplo 1

```
esParRecursivo(2)
esParRecursivo(98)
esParRecursivo(111)
esParRecursivo(43)
```

Salida Ejemplo 1

```
true
true
false
false
```

7. Suma recursiva de dos enteros positivos

Implemente una función llamada sumaRecursiva(A,B) que implemente recursivamente la suma de dos número enteros positivos A y B.

Entrada Ejemplo 1

```
sumaRecursiva(3,3)
sumaRecursiva(1294,374)
sumaRecursiva(1111,33)
```

Salida Ejemplo 1

```
6
1668
1144
```

8. Número primo

Implemente una función **primo(numero)** utilizando recursión que valide si un número es primo o no. Es posible que requiera definir una función auxiliar.

Entrada Ejemplo 1

```
primo(2)
primo(12000)
primo(853)
primo(997)
primo(11)
```

Salida Ejemplo 1

```
true
false
?
?
?
```

9. Suma de números impares

Implemente una función llamada **sumaImparesRecursivo(A,B)** que recibe como parámetros A y B. Se asume que A<=B y que pueden tomar cualquier valor entero positivo, la función deberá retornar la suma de los números impares en el rango [A,B] utilizando recursión:

Entrada Ejemplo 1

```
sumaImparesRecursivo(3,4)
sumaImparesRecursivo(3,8)
sumaImparesRecursivo(12,12)
sumaImparesRecursivo(10,11)
```

Salida Ejemplo 1

```
3
15
0
11
```

Universidad del Valle

10. Números de Lucas

Lucas ha creado la función L, esta función evaluada en 0 siempre da como resultado 2, evaluada en 1 siempre da 1. Lucas le ha indicado que usted puede hallar cualquier valor de n teniendo en cuenta que L(n) = L(n-1) + L(n-2), obteniendo una fórmula recursiva de esta forma.

L(0) = 2

L(1) = 1

L(n) = L(n-1) + L(n-2), si n > 1.

Entrada Ejemplo 1	*Salidas*
5 32	11 ?

11. Máximo Común Divisor

El Máximo Común Divisor (MCD) de dos números positivos se puede hallar utilizando la fórmula recursiva de Euclides:

$$\begin{cases} mcd(a,0) = a \\ mcd(a,b) = mcd(b, mod(a,b)) \end{cases}$$

Donde mod es una función que realiza el cálculo del resto/residuo de la división entre a y b. Esta función se define de manera recursiva como:

$$mod(x,y) = \begin{cases} mod(x+y,y);\ Si\ x < 0 \\ x;\ Si\ 0 <= x < y \\ mod(x-y,y);\ Si\ x > 0 \end{cases}$$

Implemente las funciones mod y mcd, y responda el siguiente enunciado. Giovanny y Jefferson son compañeros del curso de Fundamentos de Programación y han decidido llevar algo comer para compartir durante su exposición. Giovanny ha llevado (A) caramelos para repartir, mientras que Jefferson ha llevado (B) galletas. Si desean repartir los dulces entre sus compañeros de modo que todos tengan la mayor cantidad posible de caramelos y galletas. ¿A cuántos de sus compañeros les podrán dar dulces?. Determinar a cuántos les deben dar la misma cantidad teniendo en cuenta el número de caramelos (A) y de galletas (B) que cada uno llevó a la clase.

Entradas	Salidas
10, 8 25, 45 110, 180	? 5 ?

12. Valor obtenido con la fórmula

Implemente una función que reciba un valor de A, uno de B y un x. Calcule el valor obtenido con la fórmula recursiva:

$$\begin{cases} g(1) = A \\ g(x) = g(x-1) + B \end{cases}$$

Halle las salidas para las siguientes entradas A, B, x.

Entradas	Salidas
1,2,8 1,12,9 8,5,90	15 ? ?

13. Tetración

La tetración es una operación que es reconocida por ser la "cuarta hermana" de otras operaciones basadas en la suma: 1. La sumatoria, 2. La multiplicación, 3. La exponenciación y 4. La tetración (Tetra = cuarta e inter-ación). Aunque no es común su notación se define como nA, donde n es exponente y A es la base, pero note la ubicación de cada uno. No es igual a An, aunque es necesario conocer la operación de potencia para hallar nA. La tetración de A se define como:

La tetración también es conocida como la operación de la torre de exponentes.

$${}^{x}A = \begin{cases} 1 & si\ x = 0 \\ A^{[(x-1)A]} & si\ x > 0 \end{cases}$$

utilice Math.pow(X,Y) en lugar de ** para la potencia.

Entradas	Salidas
4,2 30,1 3,3	65536 ? ?

Universidad del Valle

14. Cuenta caracteres (extra)

Desarrolle un programa que lea un texto y diga cuántos caracteres tiene (los espacios también se cuentan). Pueden usar la función slice(1) que retorna la cadena de texto sin el primer carácter.

Entradas

```
Hola mundo
la programación hace un mejor mundo
```

Salidas

```
10
?
```

15. Palindromo (extra)

Desarrollar una función en JavaScript que determine si una palabra es palíndromo o no (devuelve true o false).

Ayuda: Se puede usar la función charAt(i), que devuelve el carácter de la cadena de texto en la posición i. Además, las funciones length y substring para devolver la longitud de la cadena de texto y una subcadena de la cadena original.

Entradas

```
ala
alambrala
reconocer
```

Salidas

```
true
false
true
```