|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

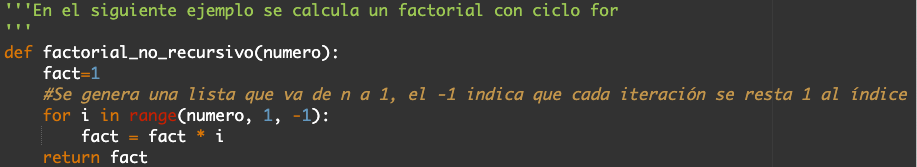
|  |  |
| --- | --- |
| *Profesor:* | M. I. Marco Antonio Martínez Quintana |
| *Asignatura:* | Estructuras de Datos y Algoritmos de programación I |
| *Grupo:* | 17 |
| *No de Práctica(s):* | 12 |
| *Integrante(s):* | Marcos Vega Alvarez |
| *No. de Equipo de cómputo empleado:* | 25 |
| *No. de Lista o Brigada:* | 39 |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 3 de Mayo de 2020 |
| *Observaciones:* |  |
|  |  |

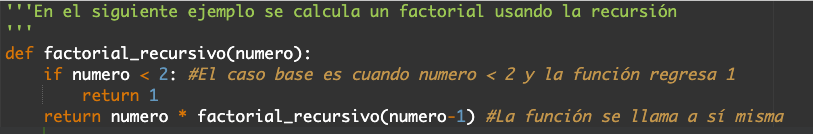
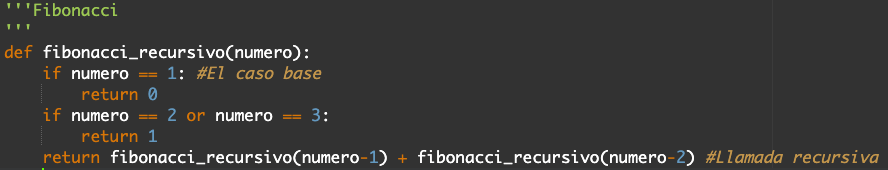
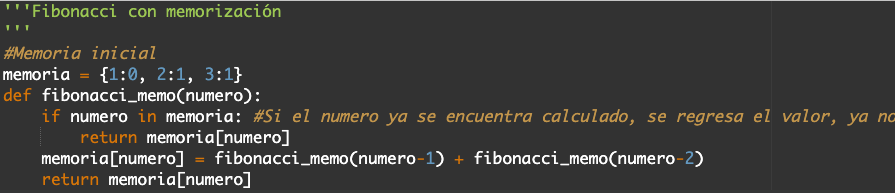
CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

**Objetivo:** El objetivo de esta guía es aplicar el concepto de recursividad para la solución de problemas.

**Introducción:**

Una función recursiva es una función que se llama a si misma. Esto es, dentro del cuerpo de la función se incluyen llamadas a la propia función. Esta estrategia es una alternativa al uso de bucles. Una solución recursiva es, normalmente, menos eficiente que una solución basada en bucles. Esto se debe a las operaciones auxiliares que llevan consigo las llamadas a las funciones. Cuando un programa llama a una función que llama a otra, la cual llama a otra y así sucesivamente, las variables y valores de los parámetros de cada llamada a cada función se guardan en la pila o stack, junto con la dirección de la siguiente línea de código a ejecutar una vez finalizada la ejecución de la función invocada. Esta pila va creciendo a medida que se llama a más funciones y decrece cuando cada función termina. Si una función se llama a si misma recursivamente un número muy grande de veces existe el riesgo de que se agote la memoria de la pila, causando la terminación brusca del programa. A pesar de todos estos inconvenientes, en muchas circunstancias el uso de la recursividad permite a los programadores especificar soluciones naturales y sencillas que sin emplear esta técnica serían mucho más complejas de resolver. Esto convierte a la recursión en una potente herramienta de la programación. Sin embargo, por sus inconvenientes, debe emplearse con cautela.

**Desarrollo:**



**Conclusión:**

• **Ventajas**:

– Soluciona problemas recurrentes

– Son programas cortos.

-Son mas cercanos a la descripción matemática.

-Generalmente mas fáciles de analizar

- Se adaptan mejor a las estructuras de datos recursivas.

-Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

• **Desventajas:**

– Creación de muchas variables

– Puede necesitar mucha memoria

**Las funciones recursivas se componen de:**

* Caso base: una solución simple para un caso particular (puede haber más de un caso base).
* Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:

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

**Factible de utilizar recursividad**

• Para simplificar el código.

• Cuando la estructura de datos es recursiva ejemplo: árboles.

**No factible de utilizar recursividad**

• Cuando los métodos usen arreglos largos.

• Cuando el método cambia de manera impredecible de campos.

• Cuando las iteraciones sean la mejor opción.

**Otros conceptos**

• Cuando un procedimiento incluye una llamada a sí mismo se conoce como recursión directa.

• Cuando un procedimiento llama a otro procedimiento y éste causa que el procedimiento original sea invocado, se conoce como recursión indirecta.

**Referencias:**

<https://www.uv.mx/personal/ocastillo/files/2011/04/Recursividad.pdf>

<https://www.inf.utfsm.cl/~smoreno/documentos/6_Tema.pdf>

<http://biolab.uspceu.com/aotero/recursos/docencia/TEMA%207.pdf>