# **Pregunta 04**

Considere la siguiente definición para una familia de funciones:

Notemos que corresponde a la definición para los números de Fibonacci:

Como un segundo ejemplo, corresponde a:

Tomando como referencia las constantes X, Y y Z planteadas en los párrafos de introducción del examen, definamos:

Se desea que realice implementaciones, en el lenguaje imperativo de su elección:

(a) Una subrutina recursiva que calcule para los valores de α y β obtenidos con las

fórmulas mencionadas anteriormente. Esta implementación debe ser una traducción

directa de la fórmula resultante a código.

(b) Una subrutina recursiva de *cola* que calcule

(c) La conversión de la subrutina anterior a una versión iterativa, mostrando claramente

cuáles componentes de la implementación recursiva corresponden a cuáles otras de la

implementación iterativa.

Debe usar el mismo el lenguaje para estos tres ejercicios y asegurarse que su lenguaje tenga las estructuras de control de flujo necesarias para realizarlos (su lenguaje escogido debe, por tanto, ser imperativo).

Realice también un análisis comparativo entre las tres implementaciones realizadas, mostrando tiempos de ejecución para diversos valores de entrada y ofreciendo conclusiones sobre la eficiencia. Es recomendable que se apoye en herramientas de visualización de datos (como los *plots* de Matlab, R, Octave, Excel, etc.)

# **Respuesta 04**

Sea el carné del estudiante Alan Argotte, quien presenta el siguiente examen, 19-10664. Luego, según las instrucciones dadas en el planteamiento inicial, tenemos que las variables X, Y y Z son los últimos tres números del carné, que son 6, 6 y 4. Por lo tanto, definimos a las constantes como







Se nos pide evaluar las expresiones

En función de los valores de X, Y y Z. Luego,

5

3

**Salida**

X = 6

Y = 6

Z = 4

Alpha = ((X + Y) % 5) + 3

Beta = ((Y + Z) % 5) + 3

print(Alpha)

print(Beta)

**Lua**

Por lo tanto, determinamos que

y

Y, en consecuencia, nos corresponde la función,

Simplificando, quedaría

Debido al tamaño del código, no podemos adjuntarlo en este documento. Sin embargo, puede verificar el repositorio de esta entrega en el siguiente link para leer el código fuente:

[**https://github.com/Arcargotte/Examen-2-Lenguajes-de-Programacion**](https://github.com/Arcargotte/Examen-2-Lenguajes-de-Programacion)

A continuación procedemos a hacer el análisis comparativo correspondiente entre las tres implementaciones realizadas para hacer conclusiones respecto a la eficiencia de los programas.

A graph with numbers and lines

AI-generated content may be incorrect.

En el siguiente gráfico puede ver tres funciones de **n** sobre tiempo. Estas tres funciones corresponden, como lo indica la leyenda, a las implementaciones de la recursión, la recursión de cola y la versión iterativa de la función .

Podemos ver como a partir de , la recursión normal comienza a tardar más tiempo en retornar un resultado. Esto coincide con lo que se ha visto en la teoría: se suelen repetir operaciones ya realizadas durante cada recursión y, para evaluar esas operaciones, se deben abrir otras recursiones. Al final, estas operaciones repetidas y las recursiones implicadas terminan haciendo el código sumamente ineficiente tal que, para , se comenzara a ser notable el tiempo de ejecución del código, tal como se ve en la gráfica. La historia cambia cuando nos fijemos en los tiempos de la recursión de cola y la versión iterativa. A grandes rasgos, parece que los tiempos son iguales (y podemos considerarlas como tal, porque la diferencia entre ambos tiempos es marginal). Esto es gracias a que la recursión de cola y la iteración sí aprovechan resultados de recursiones o iteraciones anteriores para determinar los siguientes, de manera que ese tiempo que a la recursión normal le cuesta procesar un resultado es ahorrado en el tiempo de ejecución de la recursión de cola y la versión iterativa. Por lo tanto, podemos concluir que este gráfico demuestra una realidad ya sabida de antemano sobre la recursión, la recursión de cola y la iterativa: estas últimas dos son versiones sumamente eficientes de la recursión normal.

A continuación presentamos un boceto para ilustrar las operaciones redundantes que la recursión debe hacer para procesar los resultados de las expresiones de retorno.

Para ,

+

…

+

+

+

+

+

+

+

+

…

Observe que estamos ignorando a propósito las llamadas a realizar por las otras aplicaciones de y que solo nos estamos enfocando en la primera llamada, que es . Además, de que cada llamada recursiva tiende a, inevitablemente, a llamar constantemente a con un que fue evaluado en una llamada anterior. A esto nos referimos con la redundancia de operaciones hechas por la versión recursiva de y que inevitablemente causa su ineficiencia comparada con la recursión de cola y la versión iterativa.