**Universidad Nacional Autónoma de México**

Facultad de Ingeniería

**Laboratorios de computación**

**salas A y B**

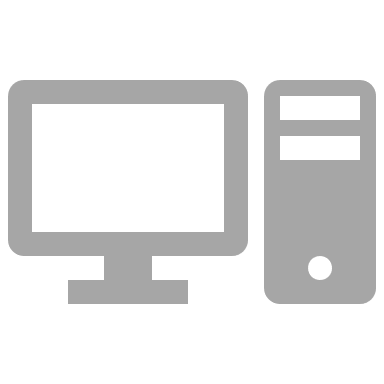
**PROFESOR:** M.I. Marco Antonio Martínez Quintana

**ASIGNATURA:** Estructura de Datos y Algoritmos I

**GRUPO:** 17

**NO DE PRÁCTICA:** 12

**NOMBRE:** Reyes Mendoza Miriam Guadalupe

**SEMESTRE:** 2020-2

**FECHA DE ENTREGA:** 28/04/2020

**OBSERVACIONES:**

**CALIFICACIÓN:**

**RECURSIVIDAD**

**OBJETIVO**

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

**INTRODUCCIÓN**

**RECURSIVIDAD**

Las funciones tienen la capacidad de llamarse a sí mismas, lo que se conoce como *recursividad* o *recursión*. La *programación recursiva*,hace uso de estas funciones, nos permitirá resolver problemas que de la forma habitual hasta ahora (programación iterativa o mediante bucles) sería muy complicado. Lo consigue aplicando un método en el que la solución a un problema se basa en resolver casos más pequeños el mismo problema.

Una *función recursiva* Debe cumplir una importante condición para ser utilizada en un programa: tiene que terminar. Lo logrará sí con cada llamada recursiva la solución del problema se reduce y nos acercamos hacia un caso límite, qué ocurre cuando el problema puede resolverse sin más recursiones. De no alcanzar nunca ese caso límite podríamos entrar en un bucle infinito.

**FACTORIAL**

Es común en matemáticas la definición de determinadas funciones dividiéndolas en dos partes: *Caso Límite* y *Caso Genérico*. En el ejemplo concreto del factorial tenemos:

**0! = 1** y **n! = n\*(n-1)!**

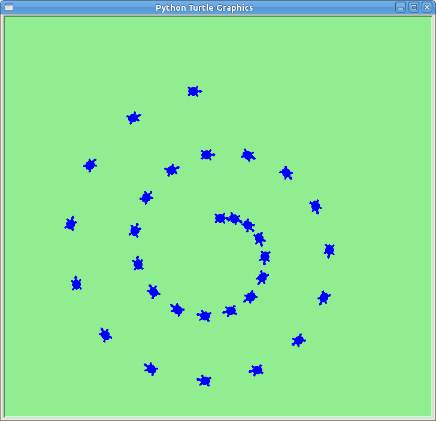
El factorial De 0 es 1; el de un número n mayor que 0 ser el mismo factorial de (n-1). En el caso genérico de la definición de factorial es donde tenemos la recursividad, ya que para calcular n! debemos calcular (n-1)!.

De la ejecución de ***factorial\_recursivo()*** se puede observar lo siguiente:

* El caso base permite terminar la recursión.
* Conforme se va decrementando la variable numero, se aproxima al caso base. El caso base ya no necesita recursión debido a que se convirtió en la versión más simple del problema.
* La función se llama a sí misma y toma el lugar del ciclo for usado en la función ***factorial\_no\_recursivo().***
* Cada que se llama de nuevo a la función, ésta tiene la copia de las variables locales y el valor de los parámetros.

Una captura de pantalla de una red social

Descripción generada automáticamente

**HUELLAS DE TORTUGA**

Las funciones para manejar la tortuga no son parte del lenguaje Python, sino que son provistas por el módulo ***turtle***. El objetivo es hacer que la tortuga deje un determinado número de huellas, cada una de las huellas se va a ir espaciando incrementalmente mientras ésta avanza. A continuación, se muestra la sección de código que hace el recorrido de la tortuga.

Para hacer ***recursivo***, primero se tiene que encontrar el caso base y después hacer una función que se va llame a sí misma. En esta función, el caso base es cuando se ha completado el número de huellas requerido. A continuación, se muestra el código de la función para el recorrido de la tortuga.

**FIBONACCI**

Es posible mejorar la eficiencia del algoritmo si se utiliza ***memorización***. La memoria cambia después de la ejecución. En comparación con la versión iterativa de la guía 11, la función ***fibonacci\_memo()***tiene acceso a la variable memoria, por lo que efectúa menos operaciones.

Una captura de pantalla de una red social

Descripción generada automáticamente

A diferencia de la versión anterior, como los resultados se están guardando en la variable memoria, el número de operaciones que se realizan es menor y no requiere de arreglos.

**DESVENTAJAS**

La principal ***ventaja*** de la recursividad frente a los algoritmos iterativos es que da lugar a ***algoritmos simples y compactos***.

La principal desventaja es que la recursividad resulta más lenta y consume más recursos al ejecutarse.

La razón por la que las funciones recursivas consumen más recursos que las iterativas es por la forma de resolverse las llamadas recursivas. Al ejecutarse una llamada función recursiva se almacena en la pila:

* Los argumentos de la función o subrutina.
* Las variables locales del subprograma.
* La dirección de retorno, es decir el punto del programa que debe ejecutarse una vez acabe la llamada actual.
* A veces es complejo generar la lógica para aplicar recursión.
* Hay una limitación en el número de veces que una función puede ser llamada, tanto en memoria como en tiempo de ejecución

Todo algoritmo recursivo puede convertirse en su equivalente iterativo y viceversa.

La recursividad será útil cuando la naturaleza de los datos así lo impongan o cuando convenga dividir un problema en casos más sencillos. Un ejemplo son los recorridos de los árboles, lo ideal para su tratamiento es utilizar algoritmos recursivos, aunque se puedan utilizar iterativos, mediante el uso de pilas.

**CONCLUSIONES**

Con nuestra práctica he podido observar diferentes técnicas y métodos para hacer eficiente un programa como lo has estado haciendo con todas las prácticas anteriores. En este caso la recursividad nos permite una resolución de problemas de manera más natural, sencilla y elegante. Una facilidad de comprobar y convencerse que la solución funciona correctamente, flexibilidad para adecuarse con mucha facilidad a los problemas, además de que la evaluación de la complejidad de las funciones recursivas es más fácil que la de las iterativas.

Nosotros como programadores siempre tendremos que estar buscando la mejor solución para resolver un problema. Una solución que nos traiga más beneficios que desventajas y que sea eficiente en cuestión de tiempo, memoria e incluso de código. A lo largo del curso hemos aprendido todo eso, y tendremos que ponerlo en práctica e irlo mejorando, porque a final de cuentas estas cosas nos diferenciarían de un programador principiante, ya que nosotros sabremos utilizar y poner en práctica más funciones que nos permitirán hacer un programa mucho más completo e incluso más complejo.

**BIBLIOGRAFÍA**

* Cuevas, A. (2019). *Programar con Python 3* (1ª edición). Madrid, España: RA-MA.
* Cobo, A. (s.f.). *Programar desde un punto de vista científico*. Madrid, España: Visión Libros.
* Arboledas, D. (2017). *Criptografía sin secretos con Python* (1ª edición). Madrid, España: RA-MA.