**Objetivo:**

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

**Introducción:**

Las funciones recursivas son funciones que se llaman a sí mismas durante su propia ejecución. Ellas funcionan de forma similar a las iteraciones, pero debe encargarse de planificar el momento en que dejan de llamarse a sí mismas o tendrá una función recursiva infinita.

Estas funciones se utilizan para dividir una tarea en sub-tareas más simples de forma que sea más fácil abordar el problema y solucionarlo

**Desarrollo:**

*Recursividad*

El propósito de la recursividad es dividir un problema en problemas más pequeños, de tal manera que la solución del problema se vuelva trivial. Básicamente, la recursión se puede explicar cómo una función que se llama así misma.

Para aplica recursión se deben de cumplir tres reglas:

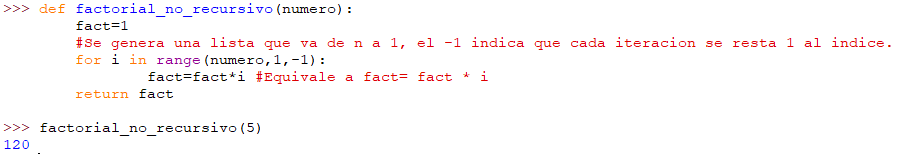
* Debe de haber uno o más casos base.
* La expansión debe terminar en un caso base
* La función se debe llamar a sí misma.

Factorial

Uno de los ejemplos más básicos es el cálculo de la factorial cuya fórmula se ve a continuación:

= 1 x 2 x 3…x (n-1) x n

En el siguiente ejemplo se calcula el factorial de un número de forma iterativa usando un ciclo for.



Para resolver este problema por medio de recursividad, hay que generar problemas más pequeños. Analizando se puede calcular mediante:

5!=5x4x3x2x1

Si se remueve el 5 se tiene

4!=4x3x2x1=4(4-1)!= 4x(3!)

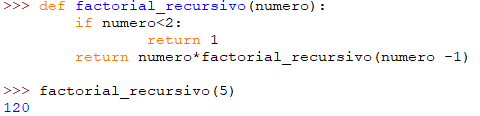
Y quedara de esta forma

4x3!=4x[3(3-1)!]=4x3x(2!)

Si se aplica a toda la secuencia, al final se tendrá

5!=5(4!)=5×4×(3!)=5×4×3×(2!)=5×4×3×2×(1!)=5×4×3×2×1×(0!)=120

Aplicando la regla explicada en un principio sobre recursividad, se puede resolver la factorial por medio de recursividad:



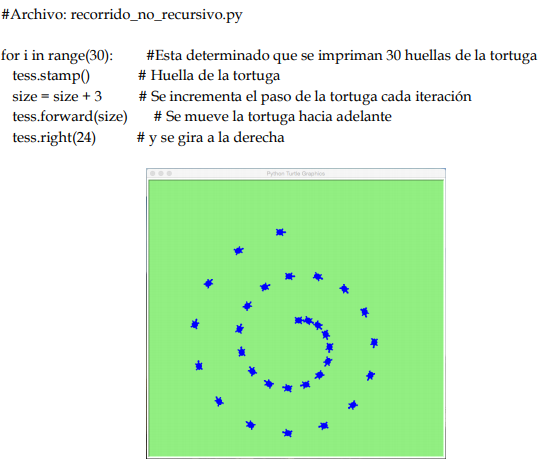
De la ejecución de factorial\_recursivo() se puede observar:

* El caso base permite terminar la recursión.
* Conforme se va decremento la variable número, 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.

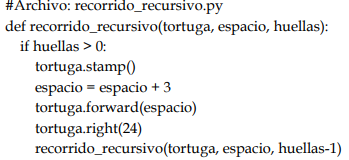
Huellas de tortuga

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 el siguiente ejemplo, se va a utilizar la biblioteca turtle. Como se observa en la siguiente imagen, hay una tortuga que se desplaza en espiral.



¿Cómo realiza el recorrido la tortuga? Primero se debe encontrar en el caso base y después hacer la función que se va llame a sí misma. En esta función, el caso base es cuando se ha completado el número de huellas requerido.

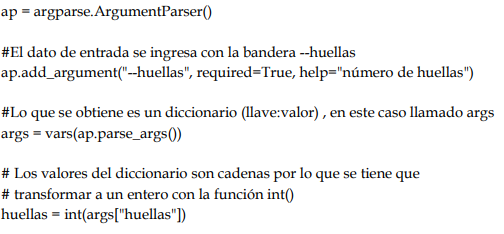


Nota: se pueden ejecutar comandos del sistema operativo, sólo se tiene que agregar el signo de admiración antes del comando (!comando). Si el comando no es del sistema operativo, se despliega un aviso.

Al momento de ejecutar las siguientes instrucciones, se abre una ventana donde se muestra el desplazamiento de la tortuga. Cuando se termina de ejecutar el código, es necesario cerrar la ventana para que finalice la ejecución



Nota: a la implementación recursiva (recorrido\_recursivo.py) se hace uso de la biblioteca agruparse, esta biblioteca permite mandar datos de entrada al programa por medio de banderas, tal y como se hace con los comandos del sistema operativo.



El código se ecjecura de la siguiente manera:

# Como se observa, hay un espacio después del nombre del archivo

# y un espacio después de la bandera

!python recorrido\_recursivo.py --huellas 25

La ventaja de utilizar esta forma de mandar datos de entrada al programa, es que hace la validación por nosotros, ya que si no se especifica la bandera o se especifica un valor, se genera un mensaje de error.

!python recorrido\_recursivo.py --huella

Fibonacci

Recordando, la implementación iterativa para calcular la sucesión de Fibonacci es:

def fibonnaci\_iterativo\_v2(numero):

f1=0

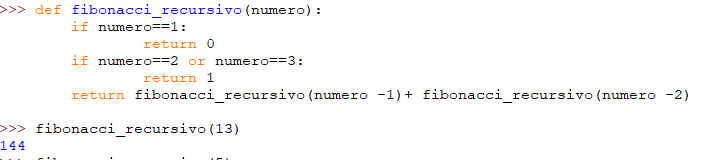
f2=1

for i in range (1,numero-1):

f1,f2=f2,f1+f2 #Asignación paralela

return f2

Esta función se puede transformar a su versión recursiva de la siguiente manera:

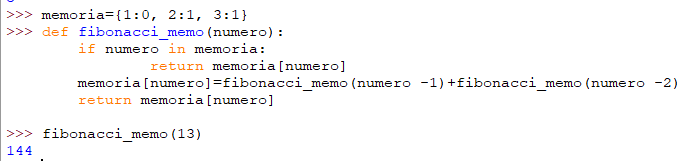


Como se vio en la práctica anterior en su versión iterativa, se repite las operaciones. Para calcular el elemento 5 se debe hacer:

f(5)=

(n-1)=f(4)+f(3)+f(2)+f(1)

(n-2)=f(3)+f(2)+f(1)

Retomando de la práctica anterior, se puede optimizar y volver eficaz el algoritmo si se utiliza la memorización. 

La memoria cambia después de la ejecución. A diferencia de la versión iterativa, en esta función Fibonacci\_memo() tiene acceso a la variable de memoria, por lo que realiza menos operaciones.

Memoria:

{1:0 2:1 3:1 4:2 5:3 6:5 7:8 8:13 9:21 10:34 11:55 12:89 13:144}

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. Para calcular el elemento 5 se tiene que hacer:

Memoria:{1:0,2:1,3:1}

F(5)=

(n-1)= f(4)+memoria(3)+memoria(2)+memoria(1)

(n-2)=memoria(3)

**Desventajas de la recursividad**

* 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.

**Conclusión.**

En esta práctica, se vio otro recurso en lo cual podemos ocupar y hacer uso haciendo el código más óptimo y eficiente, sin embargo hay que tener en cuenta las limitaciones que pueden tener al llamar la función y la organización que se le debe de dar al bloque de programa que se le va a asignar a la función, a pesar de eso, es útil ya que al llamarla realiza el procedimiento mencionado cambiando solamente algún aspecto, conservando los datos que anteriormente se dieron, también apoyándose de otras características que anteriormente se vieron.

La programación, en cualquier tipo de lenguaje, se busca siempre la mejora y la evolución al momento de codificar, dando un resultado más favorable y más conciso de lo que se quiere.

**Bibliografía.**

Laboratorio de Computación Salas A y B, INGENIERIA UNAM. (2017). Recursividad. abril 28,2020, de Recursividad Sitio web: <http://lcp02.fi-b.unam.mx/>

Wentworth,P., Elkner, J., B. Downey, A., & Meyers,C. (2008). Cómo pensar como un informático: aprender con Python 3. abril 28,2020, de Recursividad Sitio web: <http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html>

COVANTEC. (s/a). Funciones recursiva. abril 28,2020, de Recursividad Sitio web: <https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/funciones_recursivas.html>

UNIWEBSIDAD. (s/a). Llamadas recursivas. abril 28,2020, de Recursividad Sitio web: <https://uniwebsidad.com/libros/python/capitulo-4/llamadas-recursivas>