Laboratorios de computación salas A y B

Profesor:	*Marco Antonio Martinez Quintana*
Asignatura:	*Estructuras de datos y algoritmos I*
Grupo:	*17*
No de Práctica(s):	*12*
Integrante(s):	*González Cuellar Arturo*
No. de equipo de cómputo empleado:	
No. de Lista o Brigada	*16*
Semestre:	*2020-2*
Fecha de entrega:	*30 - Abril - 2020*
Observaciones:	

Calificación: _____

Recursividad

Objetivo:

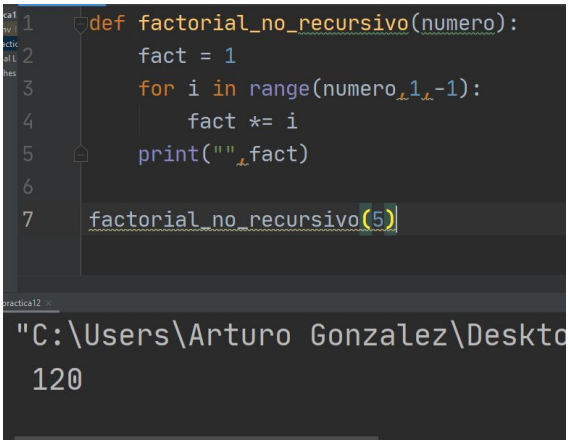
El objetivo de esta guía es aplicar el concepto de recursividad para la solución de problemas.

Introducción:

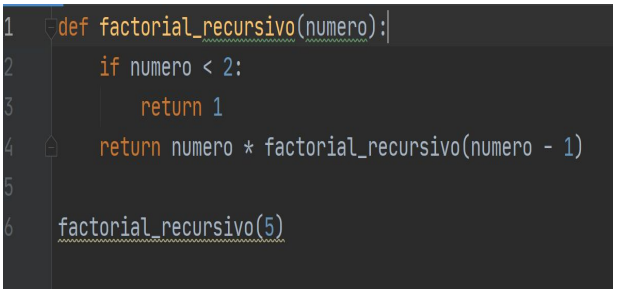
La recursividad nos sirve para dividir el problema en problemas más pequeños, de esta manera el problema se hace un poco más fácil, nos dicen que la recursividad se puede explicar como una función que se llama así misma y que para aplicarla se debe cumplir con 3 reglas, debe haber uno o más casos base, la expansión debe terminar en un caso base y la última es que la función se debe llamar a sí misma.

Desarrollo y resultados.

El primer ejemplo es uno de los más básicos, este es el cálculo del factorial de un número. Primero nos apoyamos en la formula para calcular el numero factorial. Para comenzar se calcula el factorial de un número de forma iterativa con un ciclo for



Para resolver este problema por medio de la recursividad se tienen que tomar subproblemas, desarrollando la fórmula, al final llegamos a que el factorial del número 5 es:



5×4×3×2×1×(0!)=120.

Por lo tanto aplicando las reglas de la recursividad se resuelve el problema de la siguiente manera.

El primer caso nos permite terminar la recursión, en este caso vemos como la función se llama así misma y toma el lugar del ciclo for del primer caso, de esta manera cada que se llama de nuevo a la función esta tiene la copia de los datos anteriores para modificarlos.

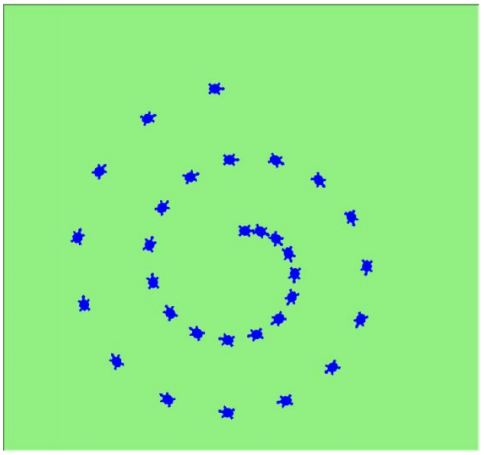
Huellas de tortuga

Para este ejemplo hacen uso de la biblioteca turtle, el objetivo de esto es que la tortuga deje un determinado número de huellas,y estas se van a ir esparciendo mientras avanza.

```
#Archivo: recorrido_no_recursivo.py

for i in range(30):
    tess.stamp()
    size = size + 3
    tess.forward(size)
    tess.right(24)
```

Para comenzar se da el rango en el ciclo para que se impriman 30 huellas, posteriormente se incrementa el paso de la tortuga, posteriormente se mueve la tortuga hacia adelante y se gira a la derecha con la última función.



Para hacer este ejemplo recursivo primero se tiene que encontrar el caso base para hacer que la función se llame a sí misma, de acuerdo con eso nuestro caso base es cuando se ha completado el número de huellas deseadas.

```
1
2   #Archivo: recorrido_recursivo.py
3
4   def recorrido_recursivo(tortuga,espacio,huellas):
5       if huellas > 0:
6           tortuga.stamp()
7           espacio = espacio + 3
8           tortuga.forward(espacio)
9           tortuga.right(24)
10          recorrido_recursivo(tortuga,espacio,huellas-1)
11
12
```

Fibonacci

Si recordamos, este ejemplo lo realizamos en la práctica pasada implementado lo siguiente:

```
def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1,f2=f2,f1+f2    #Asignación paralela
    return f2
```

```python
def fibonacci_recursivo(numero):
    if numero == 1:
        return 0
    if numero == 2 or numero == 3:
        return 1
    return fibonacci_recursivo(numero-1) + fibonacci_recursivo(numero-2)

fibonacci_recursivo(13)

```

Ahora, realizando este código de manera recursiva queda de la siguiente manera

Al igual que en la versión pasada, las operaciones se están repitiendo de la siguiente manera:

f(5) =

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

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

```python
memoria = {1:0, 2:1, 3:1}

def fibonacci_memo(numero):
    if numero in memoria:
        return memoria[numero]
    memoria[numero] = fibonacci_memo(numero-1) + fibonacci_memo(numero-2)
    return memoria[numero]

fibonacci_memo(13)

```

Sin embargo aún es posible mejorar la eficiencia del algoritmo quedando de la siguiente manera:
La ventaja de esto es que la función tiene acceso a la variable de memoria, de esta manera efectúan menos operaciones, entonces la memoria cambia después de la ejecución.

Desventajas de la recursividad

Muchas veces es complejo encontrar una lógica en los programas para aplicar el método de recursión, además de que la función tiene un límite en la cual puede ser llamada, nos dicen que esto se puede modificar pero no es muy recomendable, por lo tanto son para programas cortos o muy definidos.

Conclusión

Gracias al desarrollo de esta práctica se conoce el método de recursión para hacer de alguna manera los códigos más simples, al realizarla poco a poco iremos reconociendo y aprendiendo más acerca de la lógica para poder aplicar esto, si bien puede facilitarnos el trabajo en algunas ocasiones puede ser muy complejo encontrar esa lógica, sin embargo hay algunos problemas como algunos de los ejercicios propuestos, en los cuales es muy fácil identificar la lógica de lo que están haciendo para dividir el algoritmo en subproblemas y de esta manera resolverlo de una manera más sencilla.

Referencias: Manual de prácticas.