	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Ing. Marco Antonio Martínez Quintana

Asignatura: Estructura de datos y algoritmos I

Grupo: 17

No de Práctica(s): 12

Integrante(s): Vega Gutierrez Ricardo Daniel

No. de Equipo de cómputo empleado: 9

No. de Lista o 40

Semestre: 2020-2

Fecha de entrega: 5 de mayo del 2020

Observaciones:

CALIFICACIÓN: _____

Objetivo

Conocer y aplicar el concepto de recursividad con algunos ejercicios en Python para resolver y llegar a la solución de problemas

Introducción

El propósito de la recursividad es dividir un problema en problemas más pequeños, para los cuales la solución sea trivial, en pocas palabras, la recursividad se define como una función que se manda a llamar 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.

En el caso de Python, hay un límite en el número de veces que se puede llamar recursivamente una función, si se excede ese límite se genera el error: maximum recursion depth exceeded in comparison. Este límite puede ser modificado, pero no es recomendable.

Desventajas de la recursividad

- Es complejo generar el código.
- El número de veces que una función recursiva se puede utilizar es limitado.

Desarrollo

Para entender la recursividad, el factorial es un buen ejemplo, ya que el factorial de un número se puede calcular de múltiples maneras, en esta ocasión se hará de una forma recursiva y otra no.

Caso iterativo

Se observa que el factorial se calcula con una función, la cual usa un ciclo for para ir multiplicando cada iteración por el factorial que en su inicio vale 1 y al final retorna el factorial del número que ingresamos.

```
def facNoR(num):
    fact=1
    for hola in range(num, 1, -1):
        fact *= hola
    return fact
print("\n")
print(facNoR(5))
```

```
120
```

Caso recursivo

Para obtener el factorial se crea una función, en ella hay un caso base, el cual que el numero ingresado sea menor a 2, si el número es mayor a 2, se retorna hace uso

de la recursividad, porque retorna el número ingresado multiplicado por la función que tiene como parámetro el mismo número disminuido en uno.

```
def facRec(numero):
    if numero < 2:  #Es el caso base,
        return 1
    return numero * facRec(numero - 1)

print(facRec(5))
```

```
120
```

Huellas de tortuga

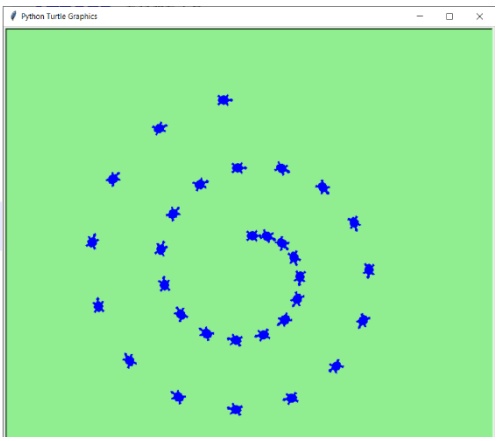
Python cuenta con un módulo «Huellas de tortuga», se necesita importarlo, tiene múltiples funciones, se dice que es huella de tortuga, porque al realizar una figura, por ejemplo: un cuadrado, te muestra en que orden dibuja cada una de las aristas de este. El cuadrado se ejecutará en una ventana emergente distinta a la terminal

El siguiente programa ejemplifica «Huellas de tortuga», se importa la biblioteca **turtle** y se configura las características de la ventana emergente, así como el tamaño y color de la tortuga, en un ciclo for se programa como se va a mover y con qué características lo hará la tortuga.

```
import turtle
wn = turtle.Screen()
wn.bgcolor("lightgreen")
tess = turtle.Turtle()
tess.shape("turtle")
tess.color("blue")

tess.penup()
size = 20
for d in range(30):
    tess.stamp()
    size = size +3
    tess.forward(size)
    tess.right(24)

wn.mainloop()
```

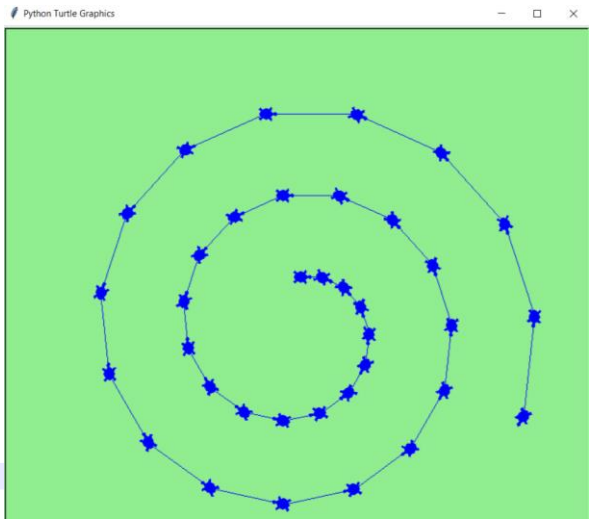


Huellas de tortuga (recursivo)

El siguiente programa comienza igual que el anterior, mismas características de la venta emergente y de la tortuga. Su diferencia esta en el código para mover la tortuga, en este caso se creó una función, la cual recibe de parámetros el nombre de la variable a la cual se le asigno el módulo de la tortuga, también recibe las huellas y el espacio que desea el usuario. La función trabaja con un caso base (huellas mayores a cero) a partir de ello genera el movimiento de la tortuga.

```
import turtle
wn = turtle.Screen()
wn.bgcolor("lightgreen")
tess = turtle.Turtle()
tess.shape("turtle")
tess.color("blue")

def tor(tortuga, espacio, huellas):
    if huellas > 0:
        tortuga.stamp()
        espacio = espacio + 3
        tortuga.forward(espacio)
        tortuga.right(24)
        tor(tortuga, espacio, huellas-1)
tor(tess, 25, 35)
wn.mainloop()
```



Fibonacci

Recordando la forma del código para generar la serie de Fibonacii, se sabe que fue programado de forma iterativa, lo cual nos hace pensar: es posible generar un código de forma recursiva, la respuesta es afirmativa, el ejemplo recursivo se muestra a continuación.

El siguiente código trabaja con un caso base, a partir del mismo muestra un elemento de la serie de fibonacci, según el indice ingresado a la función.

```
def fibonacciRecursivo(numero):
    if numero == 1: #caso base
        return 0
    if numero == 2 or numero == 3:
        return 1
    return fibonacciRecursivo(numero-1) + fibonacciRecursivo(numero-2)

print(fibonacciRecursivo(5))
```

```
3
```

Se observa que el código anterior, se puede hacer más eficiente con algunos cambios y con el método de memorización, por lo cual se crea una lista con los tres primeros elementos de la serie, en esa lista se iran guardando los nuevos elementos que se generen.

```
#Memoria inicial
memoria = {1:0, 2:1, 3:1}
print(memoria)
def fiboMemorizando(numero):
    if numero in memoria:   #si el numero ya se encuentra en memoria, solo r
        return memoria[numero]
    memoria[numero] = fiboMemorizando(numero-1) + fiboMemorizando(numero-2)
    return memoria[numero]

print(fiboMemorizando(9))
print(memoria)
```

```
{1: 0, 2: 1, 3: 1}
21
{1: 0, 2: 1, 3: 1, 4: 2, 5: 3, 6: 5, 7: 8, 8: 13, 9: 21}
```

Conclusiones

Python sigue sorprendiéndome con cada uno de sus módulos, en este caso fue «Turtle», es un módulo potente, porque te permite personalizar cada una de sus funciones, turtle te permite realizar múltiples tareas una de ellas es dibujar en la pantalla emergente, la recursividad es un método eficiente para resolver problemas, el único detalle es la complejidad para obtener su código.

Bibliografía

Design and analysis of algorithms; Prabhakar Gupta y Manish Varshney; PHI Learning, 2012, segunda edición.

Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein; The MIT Press; 2009, tercera edicion.

Problem Solving with Algorithms and Data Structures using Python; Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates; 2011, segunda edition.

http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html