	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M. I. Marco Antonio Martínez Quintana
Asignatura:	Estructura de Datos y Algoritmos
Grupo:	17
No de Práctica(s):	12
Integrante(s):	Díaz Segura, Mauricio Iván
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	
Semestre:	2020-2
Fecha de entrega:	01 - V - 2020
Observaciones:	

CALIFICACIÓN: _____

OBJETIVO

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

INTRODUCCIÓN

Python es un lenguaje de programación de alto nivel; debido a su potencia muchas empresas e institutos lo emplean actualmente.

Uno de los rasgos importantes de Python son las librerías que se pueden manejar, con las cuales se puede hacer una gran variedad de actividades, como el estudio del lenguaje, animaciones, etc.

Aunque las librerías tienen un contenido inmenso, la actividad de un programa no recae completamente en ellas, sino en la forma en que el programador diseñe el código. Una de las formas para resolver problemas, como los métodos que vimos en la práctica pasada, es la recursividad, lo que consiste en dividir el problema en diversos problemas más pequeños que se puedan resolver con facilidad. En la recursividad se llama a la función, en el caso de Python, un número limitado de veces para resolver dicho problema.

DESARROLLO Y RESULTADOS

Para comenzar, en esta práctica se elaboró una solución para uno de los problemas más comunes en el aprendizaje de algún lenguaje de programación: obtener el factorial de un número. En esta ocasión, se ha empleado, en primera instancia, una solución iterativa, como se puede ver a continuación.

```
1 def factorial_no_recursivo(numero):
2   fact = 1
3   for in in range(numero, 1, -1):
4     fact *= 1
5   return fact
6
7 factorial_no_recursivo(5)
```

```
[Out] 120
```

Este código crea una función que contiene un ciclo `for`. La operación se repetirá las veces que haya indicado el usuario como parámetro.

Otro de los ejercicios más vistos en el aprendizaje de un lenguaje de programación, es la sucesión

de Fibonacci. Para aplicar la recursividad, se ha creado una función que devuelve la misma función, es decir, se llama a sí para repetir el proceso. Asimismo, se ha implementado una estructura de condición `if`; si el usuario ingresa un número igual o menor a 3, entonces la función regresará el valor predeterminado. Esto último se hace porque ya se conocen los valores de las primeras tres posiciones en la sucesión de Fibonacci, por tanto, es mejor concentrarse en el resto del programa.

```
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) #Llamada
   recursiva

  fibonacci_recursivo(13)
```

```
[Out] 233
```

Por último, se realizó una animación de una tortuga, la cual va dejando sus huellas. Lamentablemente, el manual de prácticas no contiene mucha información para programar este caso, pero deja un enlace en donde se puede estudiar el comportamiento de la librería que se usó, `turtle`.[4]

Antes que nada, se deben crear los elementos que se van a usar que son, en realidad, el puntero que se moverá y el fondo de pantalla.

```
#codigo obtenido de http://openbookproject.net/thinkcs/python/english3e/
   hello_little_turtles.html
import turtle
wn = turtle.Screen()      # Crea un fondo para las tortugas
alex = turtle.Turtle()    # Crea una tortuga, alex

alex.forward(50)          # alex se mueve 50 unidades
alex.left(90)             # alex gira 90 grados
alex.forward(30)

wn.mainloop()             # espera a que el usuario cierre la ventana
```

Como se puede observar, primero se crean el fondo de pantalla y la tortuga; después, esta última se mueve hacia delante, gira noventa grados y vuelta a avanzar. Con esto, describe dos líneas, que se muestran en la figura 1.

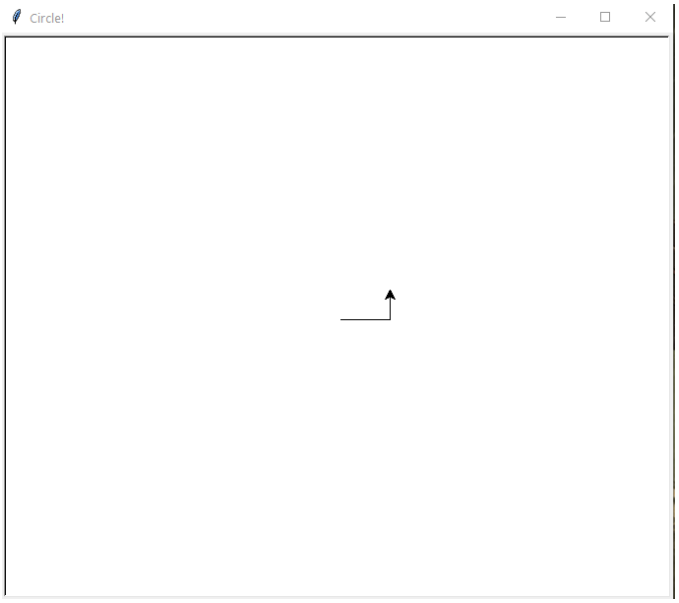


Figura 1. Dos líneas perpendiculares

La tortuga puede avanzar o girar tanto como se especifiqué o le indique el usuario, en caso de que se almacene la entrada en una variable.

Ahora, en un archivo diferente, se creó otra tortuga, pero los pasos que dará serán realizados por medio de una estructura iterativa. En el siguiente código, se crea la tortuga y el fondo de pantala, por medio de sus atributos, se puede cambiar el color de ambas. Después, se define su tamaño en la variable `size` y, por medio de `for`, la tortuga dejará manchitas separadas, las cuales, por cada iteración, aumentará su espacio en tres unidades. En la última línea, la función `mainloop` deja la ventana de la animación abierta hasta el usuario interactúe con ella o la cierre.

```
#nombre del archivo: recorrido_no_recursivo.py
wn = turtle.Screen()
wn.bgcolor("antiquewhite")
tortu = turtle.Turtle()
tortu.shape("turtle")
tortu.color("blueviolet")

tortu.penup()
size = 20

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

Como este código se encuentra en un archivo aparte del cuerpo del programa, para llamarlo, sólo se escribe el comando requerido para la consola del sistema, antecedido por un signo de exclamación (!). El resultado se encuentra en la figura 2.

```
!python recorrido_no_recursivo.py
```

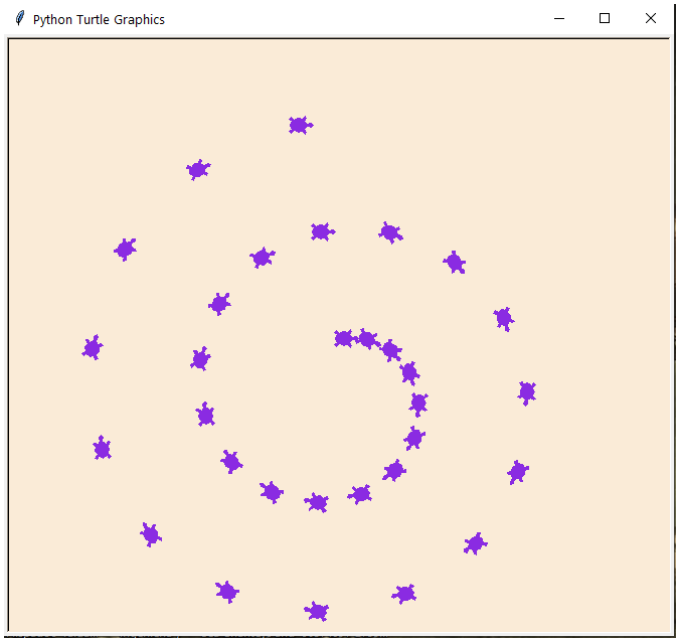


Figura 2. Pasos de tortuga por método iterativo

Finalmente, para mejorar el programa, se creó otro archivo que hará lo mismo que al anterior, con la diferencia de que empleará el método recursivo, es decir, la función se llamará a sí misma un número determinado de veces.

```
#Nombre del archivo: recorrido_recursivo.py
def recorrido_recursivo(tortuga, espacio, huellas):
  if huellas > 0:
    tortuga.stamp()
    espacio = espacio + 3
    tortuga.forward(espacio)
    tortuga.right(24)
    recorrido_recursivo(tortuga, espacio, huellas-1)
```

Esto es posible por una función que requiere tres parámetros: `tortuga`, `espacio` y `huellas`. Si el número de huellas mayor a cero, la tortuga dará un paso, el espacio aumentará tres unidades, rotará 24 grados y, al final, llamará de nuevo a la misma función. Esto lo repetirá hasta que el número de huellas sea igual a cero.

No obstante, para hacer este programa más divertido, se pueden modificar los parámetros desde

el código principal, en el momento en que se llama a la función. Para realizar esto, en el código recursivo, se deben añadir los argumentos que entrarán, como si fueran apuntadores en C.

```
#Continuacion del archivo: recorrido_recursivo.py
#Crear las referencias para poder pasar la informaci n desde el cuerpo del
    programa
ap = argparse.ArgumentParser(conflict_handler='resolve')
ap.add_argument('--huellas', required=True, help="n mero de huellas")
ap.add_argument('--espacio', required=True, help="espacio de recorrido")
ap.add_argument('--scolor', help="color de fondo del programa")
ap.add_argument('--tcolor', help="color de la tortuga")

#Crear las variables que crear n a la tortuga y asignar n valor a las huellas
    y a los espacios
args = vars(ap.parse_args())
scolor = args["scolor"]
tcolor = args["tcolor"]
huellas = int(args["huellas"])
espacio = int(args["espacio"])

#crear la tortuga
wn = turtle.Screen()
if scolor is None:
    #Color predeterminado de la pantalla si no se introdujo alguno en el cuerpo
    del codigo
    wn.bgcolor("lightgreen")
else:
    wn.bgcolor(scolor)
tort = turtle.Turtle()
tort.shape("turtle")
if tcolor is None:
    #Color predeterminado de la tortuga si no se introdujo alguno en el cuerpo
    del codigo
    tort.color("blue")
else:
    tort.color(tcolor)
tort.penup()
```

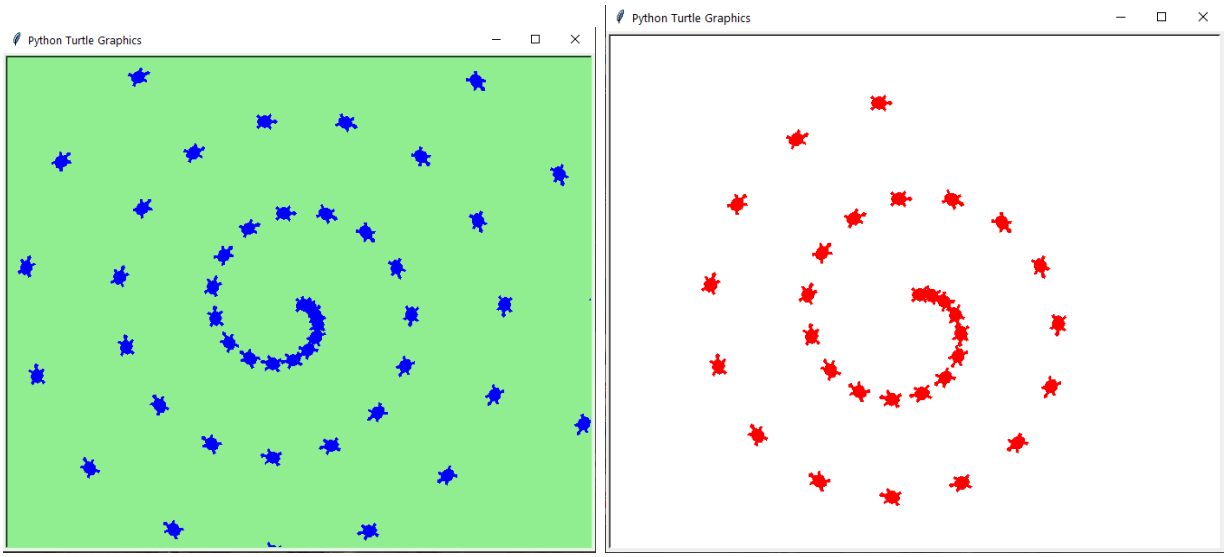
De manera general, en las primeras líneas, se definen qué parámetros se necesitan, en este caso, se encuentran `huellas`, `espacio`, `scolor` y `tcolor`, de los cuales, los primeros dos son obligatorios. Todo se guarda en un diccionario. Luego, se almacena el valor conferido desde el código principal en variables locales por medio de las llaves correspondientes. El siguiente paso es crear la tortuga,

como ya se ha hecho antes, sin embargo, esta vez hay estructuras de condición con el fin de que, si no se introdujo un parámetro para los colores del programa, se ponen algunos predeterminados que son verde claro para el fondo de pantalla y azul para la tortuga. Por último, se llama a la función con los parámetros establecidos.

```
!python recorrido_recursivo.py  --huellas 50 --espacio 0
```

Se pueden hacer combinaciones diversas con los parámetros disponibles.

```
!python recorrido_recursivo.py --scolor white --tcolor red --huellas 30 --
   espacio 10
```



(a) **(b)**

Figura 3. (a) Animación con colores predeterminados (sin parámetros introducidos por el usuario (b) Animación con colores a elección

CONCLUSIONES

La recursividad es un método de resolución de problemas bastante efectivo ya que permite crear una solución inmediata, sencilla y, al mismo tiempo, elaborar un código simple, cuyos beneficios podremos observar una vez que querramos leerlo.

REFERENCIAS

1. *Tutorial de Python.* Fecha de consulta: 28 de abril de 2020. Recuperado de http://docs.python.org.ar/tutorial/3/real-index.html

2. Distribuidor de Anaconda. Fecha de consulta: 28 de abril de 2020. Recuperado de https://www.anaconda.com/why-anaconda/

3. *argparse - Parser for command-line options, arguments and sub-comands.* Fecha de consulta: 28 de abril de 2020. Recuperado de https://docs.python.org/3/library/argparse.html

4. *Hello, little turtles!* Fecha de consulta: 28 de abril de 2020. Recuperado de http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html