|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Marco Antonio Martinez Quintana |
| *Asignatura:* | Estructura de Datos y Algoritmos I |
| *Grupo:* | 17 |
| *No de Práctica(s):* | 12 |
| *Integrante(s):* | Enzo Valdés Zavala |
| *No. de Equipo de cómputo empleado:* |  |
| *No. de Lista o Brigada:* |  |
| *Semestre:* | 2020-1 |
| *Fecha de entrega:* | 27 de mayo del 2020 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

**Objetivos:**

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

**Introducción:**

**Recursividad:**El propósito de la recursividad es dividir un problema en problemas mas pequeños, de tal forma que la solución del problema se vuelva trivial.  
Basicamente, la recursión se explica como una función que se llama asi misma.  
  
3 reglas de aplicación:  
-Minimo un caso base (puede haber mas)  
-La expansión debe terminar en un caso base  
-La función debe llamarse a si misma.  
  
**Factorial:**Uno de los ejemplos mas básicos es el calculo del factorial cuya formula se muestra a continuación:

En el siguiente ejemplo, calcularemos el factorial de una forma iterativa usando un ciclo for:  
A screenshot of a computer screen

Description automatically generated

Como se menciono anteriormente, para resolver un problema por medio de recursividad hay que generar problemas mas pequeños.   
Analizando la forma en que se calcula el factorial en la función pasada se tiene que:

5!=5×4×3×2×1

Si se remueve el 5 se tiene:

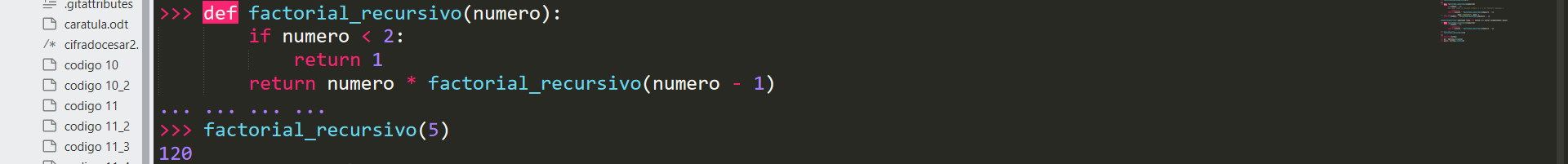
4!=4×3×2×1=4(4−1)!=4×(3!)

Se puede afirmar que:

4×3!=4×[3(3−1)!]=4×3×(2!)

Si se aplica esto a toda la secuencia, al final tenemos la siguiente expansión:

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

Aplicando las reglas explicadas en un principio sobre recursividad, se puede resolver el problema del factorial por medio de recursión de la siguiente manera:  
  
  
De la ejecución de factoria\_recursivo se puede observar lo siguiente:  
  
- El caso base permite terminar la recursión.  
- Conforme se va decrementando la variable numero, se aproxima el caso base. El caso base ya no necesita recursión debido a que se convirtió en la versión mas simple del problema.  
- La función se llama a si misma y toma el lugar del ciclo for usado en la función factorial\_no\_recursivo().  
- Cada vez que se llama de nuevo a la función, esta tiene la copia de las variables locales y el valor de los parámetros.

**Huellas de 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.   
El objetivo es hacer que la tortuga deje un determinado numero de huellas, cada una de las huellas se va ir espaciando incrementalmente mientras esta avanza. A continuación se muestra la sección de código que hace el recorrido de la tortuga.  
  
**#Archivo: recorrido\_no\_recursivo.py**

**for i in range(30): #Esta determinado que se impriman 30 huellas de la tortuga**

**tess.stamp() # Huella de la tortuga**

**size = size + 3 # Se incrementa el paso de la tortuga cada iteración**

**tess.forward(size) # Se mueve la tortuga hacia adelante**

**tess.right(24) # y se gira a la derecha**  
¿Cómo hacer el recorrido de la tortuga de manera recursiva? Primero se tiene que encontrar el caso base y después hacer una función que se llame a si misma.  
En esta función, el case base es cuando se ha completado el numero de huellas requerido. A continuación, se muestra el código de la función para el recorrido de la tortuga.  
  
  
  
**#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)**

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 en la notebook.  
**#Ejecutando el código no recursivo.**

**!python recorrido\_no\_recursivo.py**Para la implementación recursiva se hace uso de la biblioteca argparse, esta biblioteca permite mandar datos de entrada al programa por medio de banderas, tal y como se hace con los comandos del sistema operativo.   
  
**ap = argparse.ArgumentParser()**

**#El dato de entrada se ingresa con la bandera --huellas**

**ap.add\_argument("--huellas", required=True, help="número de huellas")**

**#Lo que se obtiene es un diccionario (llave:valor) , en este caso llamado args**

**args = vars(ap.parse\_args())**

**# Los valores del diccionario son cadenas por lo que se tiene que**

**# transformar a un entero con la función int()**

**huellas = int(args["huellas"])**El código se ejecuta 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 fibonacci\_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:  
A screen shot of a computer

Description automatically generated  
  
Al igual que en la versión iterativa, se están repetiendo operaciones.  
Para calcular el elemento 5 se tiene:  
  
**f(5) =**

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

**(n-2) = f(3)+f(2)+f(1)**Retomando lo visto en la practica 11, es posible mejorar la eficiencia del algoritmo si se utiliza la memorización.  
**A screen shot of a computer

Description automatically generated**

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 efectua 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 con la nueva implementación se tiene:

**memoria = {1:0, 2:1, 3:1}**

**f(5) =**

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

**(n-2) = memoria(3)  
  
Desarrollo:**

En esta practica, solo se nos pidió replicar los ejercicios puestos en la guía.

**Conclusión:**  
Pude completar la practica sin ningún problema, replicando todos los problemas y resolviéndolos en sublimetext Python.

**Bibliografía:**

Tutorial oficial de Python: https://docs.python.org/3/tutorial/

Galería de notebooks: https://wakari.io/gallery

Matplotlib: http://matplotlib.org/  
  
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.

https://docs.python.org/3/library/itertools.html#

https://docs.python.org/3/library/itertools.html#itertools.product

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

<https://docs.python.org/3.5/library/pickle.html>  
  
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.