| | |
|---|---|
| | **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):* José Luis Arroyo Chavarría

*No. de Equipo de cómputo empleado:* 1

*No. de Lista o Brigada:* 5

*Semestre:* 2

*Fecha de entrega:* 07/04/2020

*Observaciones:*

CALIFICACIÓN: _____

**Objetivo:**

Aplicar la recursividad encontrar la solución de problemas.

**Introducción:**

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 y la función se debe llamar a sí misma

Desventajas de la recursividad: A veces es complejo generar la lógica para aplicar recursión y hay una limitación en el número de veces que una función puede ser llamada, tanto en memoria como en tiempo de ejecución.

- **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 contiuación se muestra la sección de código que hace el recorrido de la tortuga.

- **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.

**Desarrollo y resultados:**

- **Código**

1. **Factorial**

```
def factorial_no_recursivo(numero):
    fact = 1
    #Se genera una lista que ve de n a 1, el -1 indica que cada iteracion se resta 1 al indice
    for i in range(numero, 1, -1):
        fact *= i  #Esto es equivalente a fact = fact * i
    return fact
factorial_no_recursivo(5)
```

2. **Recursión**

```
def factorial_recursivo(numero):
    if numero < 2:   #El caso base es cuando numero < 2 y la funcion regresa 1
        return 1
    return numero * factorial_recursivo(numero - 1) #La funcion se llama a si misma
factorial_recursivo(5)
```

**3. Huellas de 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

#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)
```

**4. Comando (!comando)**

```
#Ejecutando el código no recursivo.
!python recorrido_no_recursivo.py
```

**5. Argparse**

```
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
```

**6. Fibonacci**

```
def fibonacci_recursivo(numero):
    if numero == 1:    #Caso base
        return 0
    if numero == 2 or numero == 3:
        return 1
    return fibonacci_recursivo(numero-1) + fibonacci_recursivo(numero-2) #Llamada
recursiva
fibonacci_recursivo(13)
```

**7. Memorización**

```
#Memoria inicial
memoria = {1:0, 2:1, 3:1}

def fibonacci_memo(numero):
    if numero in memoria:     #Si el numero ya se encuentra calculado, se regresa el
valor ya no se hacen mas calculos
        return memoria[numero]
    memoria[numero] = fibonacci_memo(numero-1) + fibonacci_memo(numero-2)
    return memoria[numero]
fibonacci_memo(13)
```

**Captura de pantalla**

```python
def factorial_no_recursivo(numero):
    fact = 1
    #Se genera una lista que ve de n a 1, el -1 indica que cada iteracion se resta 1 al indice
    for i in range(numero, 1, -1):
        fact *= i  #Esto es equivalente a fact = fact * i
    return fact
factorial_no_recursivo(5)
```

```
120
```

```python
def factorial_recursivo(numero):
    if numero < 2:   #El caso base es cuando numero < 2 y la funcion regresa 1
        return 1
    return numero * factorial_recursivo(numero - 1) #La funcion se llama a si misma
factorial_recursivo(5)
```

```
120
```

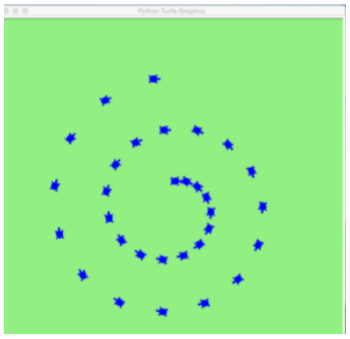
```python
def fibonacci_recursivo(numero):
    if numero == 1:    #Caso base
        return 0
    if numero == 2 or numero == 3:
        return 1
    return fibonacci_recursivo(numero-1) + fibonacci_recursivo(numero-2) #Llamada recursiva
fibonacci_recursivo(13)
```

```
144
```

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

def fibonacci_memo(numero):
    if numero in memoria:     #Si el numero ya se encuentra calculado, se regresa el valor ya no se hacen mas calculos
        return memoria[numero]
    memoria[numero] = fibonacci_memo(numero-1) + fibonacci_memo(numero-2)
    return memoria[numero]
fibonacci_memo(13)
```

```
144
```



**Conclusión:**

Al realizar los problemas pude ver que puedo resumir o hacer más cortas las instrucciones que se le piensa agregar en el código. En mi opinión siento que esto nos puede ayudar al realizar problemas o programas con varias funciones.

**Bibliografías y Cibergrafías:**


- 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
- https://jupyter.org/try