Recursividad

Programación
2024

UNRaf UNIVERSIDAD NACIONAL DE RAFAELA

Cálculo del factorial

- El factorial de un entero no negativo *n* (escrito n! y pronunciado *n* factorial) se calcula como el producto:

$$n! = n * (n - 1) * (n - 2) * \ldots * 1$$

- Por ejemplo:

$$4! = 4 * 3 * 2 * 1$$

$$5! = 5 * 4 * 3 * 2 * 1 = 5 * 4!$$

- Por la tanto la definición puede escribirse como:

$$n! = n * (n - 1)!$$

$4! = 4 * 3!$ | $3! = 3 * 2!$ | $2! = 2 * 1!$ | $1! = 1$

Funciones dentro de funciones

- Al definir una función, es posible incorporar sentencias que llamen a otras funciones.
- La función original no continúa hasta no obtener los resultados de la función a la que llama.

Funciones dentro de funciones

- Ejemplo: cálculo del desvío estándar de una lista de valores.

```
array = [1, 3, 5, 7]
desv = desvio(array)
print(desv)
```

Programa principal

Funciones dentro de funciones

- Ejemplo: cálculo del desvío estándar de una lista de valores.

```
array = [1, 3, 5, 7]
desv = desvio(array)



print(desv)
```

```
prom = media(array)
# calculos …
return sd
```

Programa principal

Función “desvio”

Funciones dentro de funciones

- Ejemplo: cálculo del desvío estándar de una lista de valores.

```
array = [1, 3, 5, 7]
desv = desvio(array)

print(desv)
```

Programa principal

```
prom = media(array)

# calculos …
return sd
```

Función “desvio”

```
for i in array:
  # mas calculos…
return promedio
```

Función “media”

Recursividad

- Una función recursiva es aquella que se llama a sí misma.

```
r = func_recurv(var)
```

```
r2 = func_recurv(var2)
return r2
```

```
r3 = func_recurv(var3)
return r3
```

...

Programa principal

Función “func_recurv”

Componentes de la recursividad

- Todo algoritmo recursivo tiene que tener, al menos, 2 componentes:
  - Componente recursivo.
  - Componente base.
- El **componente** o **caso base** es un caso que tiene una solución directa (no recursiva).

$$f(n)\begin{cases} 1 & n \leq 1 \\ n*f(n-1) & n > 1 \end{cases}$$

Implementando función “factorial”

```
def factorial_recursivo (n):
    if n <= 1:          # Caso base
        return 1
    else:               # Caso recursivo
        resultado = n * factorial_recursivo(n-1)
        return resultado

fact_rec = factorial_recursivo(10)
print(“Recursivo: “, fact_rec)
```

```
Recursivo:  3628800
```

Recursión vs Iteración

- Por definición, todo problema que pueda resolverse mediante recursión se puede resolver por iteración.
- Las soluciones recursivas suelen ser más lentas y demandar más memoria, pero suelen ser más fáciles de comprender, implementar y depurar.
- Si una solución de un problema se puede expresar iterativa o recursivamente con igual facilidad, **es preferible la solución iterativa**.

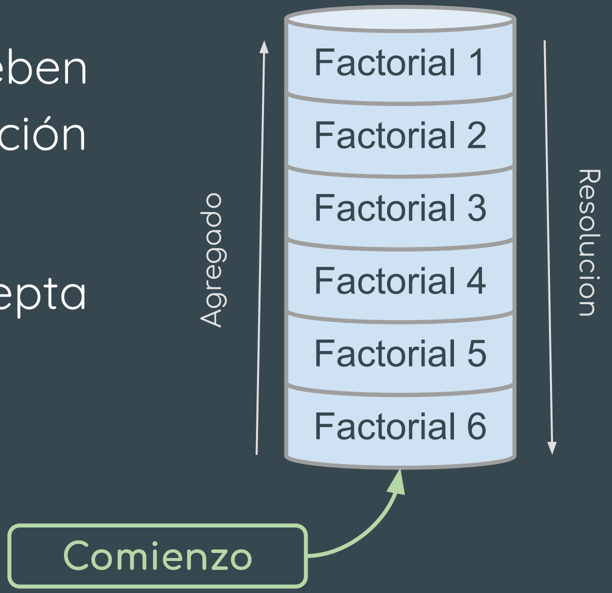
Implementando función “factorial”

```
def factorial_recursivo (n):
    if n <= 1:          # Caso base
        return 1
    else:               # Caso recursivo
        resultado = n * factorial_recursivo(n-1)
        return resultado

def factorial_iterativo (n):
     resultado = 1
     for i in range (2,n+1):
     resultado *= i
     return resultado

fact_rec = factorial_recursivo(10)
fact_iter = factorial_iterativo(10)
print(“Recursivo: “, fact_rec, “Iterativo”, fact_iter)
```

```
Recursivo:  3628800 Iterativo 3628800
```

Límite de recursividad

- Por cada llamada dentro de una función recursiva, se agrega un valor a una **pila de recursión**.
- Esta pila mantiene el orden en que se deben resolver las distintas llamadas a la función recursiva.
- Por defecto, en Python, esta cola acepta acepta 1000 valores (luego da error).



Precauciones!!

- Siempre debe haber, al menos, **un caso base**; sino la llamada recursiva se hace **infinita**.

- Si la llamada anterior requiere un resultado de una siguiente, siempre debe haber un ***return***.

- Cuando se trabaja con tipos de datos no mutables (listas), no hace falta retornarlos: todos los cambios se mantienen!

Ejemplo 1

- Implemente la operación “multiplicación” como una suma recursiva.

Ejemplo 1: pseudocódigo

- Implemente la operación “multiplicación” como una suma recursiva.

Pseudocodigo:

```
Recibir dos números: multiplicando y multiplicador
si multiplicador == 1:
    devolver multiplicando
sino:
   calcular multiplicando + f(multiplicando, multiplicador-1)
```

Ejemplo 1: código

```
def mult (multiplicando, multiplicador):
    print("multiplicando: ", multiplicando) # Solo para visualizar la recursión
    print("multiplicador: ", multiplicador)
    if multiplicador == 1:
        return multiplicando
    else:
        return multiplicando + mult(multiplicando, multiplicador-1)


resultado = mult(4,6)
print(resultado)
```

Ejemplo 1

```
(base) matias@rfgenom002:~$ python test.py
multiplicando:  4
multiplicador:  6

multiplicando:  4
multiplicador:  5

multiplicando:  4
multiplicador:  4

multiplicando:  4
multiplicador:  3

multiplicando:  4
multiplicador:  2

multiplicando:  4
multiplicador:  1

24
```

Ejemplo 2

- Encuentre todas las combinaciones de cara y cruz que se pueden obtener luego de lanzar n veces una moneda

Ejemplo 2: planteo del problema

- La solución para un lanzamiento es:

  C X

- La solución para dos lanzamientos es:

  CC XC CX XX

- La solución para 3 lanzamientos es:

  CCC XCC CXC XXC CCX XCX CXX XXX

Ejemplo 2: planteo del problema

- La solución para un lanzamiento es:

  C X

- La solución para dos lanzamientos es:

  CC XC CX XX

- La solución para 3 lanzamientos es:

  CCC XCC CXC XXC CCX XCX CXX XXX

- A la solución para 3 lanzamientos también la podemos pensar como cara más las soluciones para 2 lanzamientos y cruz más las soluciones para 2 lanzamientos.
- Asi mismo, podemos pensar a la solución para dos lanzamientos como cara y cruz más las soluciones para un lanzamiento.

Ejemplo 2: planteo del problema

- Generalizando:
  - la solución para cualquier número de lanzamientos n, mayor a 1, es “C” más la solución para los n-1 lanzamientos restantes y “X” más la solución para los n-1 lanzamientos restantes.
  - Cuando n es 1, a la solución final es la solución anterior más “C” y “X”

Ejemplo 2: pseudocódigo

- Si n > 1:
  - Sumar “C” a la solución actual y volver a calcular para n-1 lanzamientos.
  - Sumar “X” a la solución actual y volver a calcular para n-1 lanzamientos.
- Si n == 1:
  - Sumar “C” a la solución actual y mostrar.
  - Sumar “X” a la solución actual y mostrar.

Ejemplo 2: código

```
def lanzamientos (n, combinacion_actual=""):
    if n > 1:
        resultado_sumando_cara = "C" + combinacion_actual
        lanzamiento(n-1, resultado_sumando_cara)
        resultado_sumando_cruz = "X" + combinacion_actual
        lanzamiento(n-1, resultado_sumando_cruz)
    else:
        print(“C” + combinacion_actual)
        print(“X” + combinacion_actual)
```