Ejercitación № 3: Recursión en Python

Cátedra Programación II Marzo 2024

Ejercicios de conteo recursivos en Python

a) Función `cuenta_regresiva`

Programar y probar en el intérprete de Python el siguiente código para la función recursiva `cuenta_regresiva` probar de invocarlas con diferentes argumentos. Agregar también una consulta al usuario que nos indique desde dónde quiere empezar a contar hacia atrás.

$$\text{cuenta\_regresiva (n)} = \begin{cases} \text{"Achis!"} & \text{si } n = 0 \\ \text{cuenta\_regresiva(n-1)} & \text{si } n > 0 \end{cases}$$

```python
def cuenta_regresiva(n):
    """ Diseño:
            Número: Natural
            Mensaje: String
        Signatura:
             Nat->String
        Propósito:
            Realiza una cuenta regresiva a partir de un número natural dado,↩
                imprimiendo en pantalla la cuenta regresiva
            como mensaje hasta llegar al valor cero.
    """
    if n == 0:
       return "Achis!!"
    else:
       return str(n)+ "\n" + cuenta_regresiva(n-1)

def main():
    respuesta = cuenta_regresiva(10)
    print (respuesta)

main()
```

Ejercicio 1. Una variante de esta función es la función `cuenta_progresiva` la cual cuenta en forma ascendente comenzando desde cero hasta el valor 10, y al llegar a éste último valor, emite el mensaje *Boom!*.

$$\text{cuenta\_progresiva (n)} = \begin{cases} \text{"Boom!"} & \text{si } n = 10 \\ \text{cuenta\_progresiva(n+1)} & \text{si } n \geqslant 0 \wedge n < 10 \end{cases}$$

Te pedimos en este caso que:

(a) Implementes y diseñes una función recursiva en Python que nos permita programar la función `cuenta_progresiva`. ¿Qué diferencia hay con la anterior? ¿Cómo debo invocarla en la función `main`?

(b) ¿Qué problemas podrían llegar a surgir en este tipo de código? ¿Puedo variar el tope de conteo?: Si, no, y/o porqué, ¿qué necesitarías? De ser posible, implementar dicha función pidiéndole al usuario que ingrese un valor que indique hasta donde se realizará el conteo. En caso contrario argumenta con palabras tu decisión.

b) Función factorial

Otro ejemplo clásico de recursión es la función *factorial* de un número natural n, definida como la multiplicación de los números naturales desde 1 hasta n, por ejemplo, el factorial de 7, señalado como 7! sería:

$$7! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 = 5040.$$

El cálculo del número factorial n ya lo hemos implementado en la *programación imperativa* usando la repetición simple en la unidad 1, recordemos el diseño de esta función implementada aquí en: `factorial_ite`

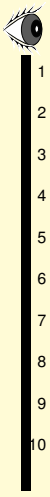
```python
def factorial_ite(num):
  """ Número: Natural.
      Nat->Nat
      Calcular en forma iterativa el factorial de un número natural dado.
      factorial_ite(5) = 120,  factorial_ite(0) = 1,  factorial_ite(1) = 1
  """
  fact = 1
  for i in range(2,num+1):
    fact = fact * i
  return fact

def main():
    valor = int (input ("Ingrese un valor natural para cualcular su ↩
       factorial: "))
    print("El factorial de ", valor , "es ", factorial_ite(valor))

main()
```

Esta solución se la conoce como *"la versión iterativa del factorial"*

La definición matemática de la operación *factorial* queda dada mediante la siguiente función recursiva:

$$\text{factorial (n)} = \begin{cases} 1 & \text{si } n = 0 \\ \text{factorial(n-1)*n} & \text{si } n > 0 \end{cases}$$

y su implementación recursiva en la programación imperativa sería la función: `factorial_rec`



```
def factorial_rec(n):
    """ Número: Natural.
        Nat->Nat
        Calcular en forma recursiva el factorial de un número natural dado.
        factorial_rec(5) = 120, factorial_rec(0) = 1, factorial_rec(1) = 1
    """
    if n == 0:
      return 1
    else:
      return n*factorial_rec(n-1)
```

En estos casos que hemos visto y analizados, las funciones recursivas se clasifican o categorizan como *funciones recursivas simples lineales*, donde aparece una sola llamada recursiva.

Ejercicio 2. Dada la definiciones de las funciones anteriores, te pedimos en este caso que completes las definiciones agregando:

(a) Completar la receta de las funciones agregando los casos de test usando el módulo **Pytest** para: `factorial_ite` y `factorial_rec`.

(b) ¿Qué diferencias y similitudes hay entre ambas recetas? Detallar y comparar.

(c) Pedir al usuario que ingrese un valor por teclado y calcular el factorial usando ambas funciones, mostrando ambos resultados en pantalla. Verificar con varios valores como por ejemplo: $5, 10, 500, 1024$.

(d) Realizar en papel el comparativo respecto del proceso de cálculo llevado adelante entre la invocación a la función `factorial_ite(5)` y la función recursiva `factorial_rec(5)`. Para la función iterativa te pedimos que elabores la tabla de iteraciones y para la función recursiva el cálculo paso a paso.

(e) ¿Qué puede observar en los cálculos entre estas funciones? Pensar en los problemas que trae la recursión en el uso de la memoria y describirlos en papel.

c) Función Fibonacci

La definición recursiva puede involucrar en el cuerpo varios llamados a si misma, en este caso la recursión se llama *recursión múltiple*. La función de Fibonacci es uno de éstos casos:

$$\text{fibonacci (n)} = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{fibonacci(n-1)+ fibonacci(n-2)} & \text{si } n \geqslant 2 \end{cases}$$

```python
def fibonacci(n):
    """ Diseño:
            Número: Natural.
        Signatura:
             Nat->Nat
        Propósito:
             Calcular en forma recursiva el número de fibonacci de un número natural dado.
         Ejemplos:
             fibonacci(0) = 0
             fibonacci(1) = 1
             fibonacci(2) = 1
             fibonacci(8) = 21
    """
    if n == 0 or n == 1:
      return n
    else:
      return fibonacci(n-1) + fibonacci(n-2)
```

Ejercicio 3. Dadas las siguientes ecuaciones recursivas diseñar funciones en Python que permitan evaluarlas.

a)
$$\begin{aligned} lucca(0) &= 2 \\ lucca(1) &= 1 \\ lucca(n) &= lucca(n-1) + lucca(n-2) \quad \text{para } n \geqslant 2 \end{aligned}$$

b)
$$\begin{aligned} pell(0) &= 0 \\ pell(1) &= 1 \\ pell(n) &= 2 \times pell(n-1) + pell(n-2) \quad \text{para } n \geqslant 2 \end{aligned}$$

c)
$$\begin{aligned} jacob(0) &= 0 \\ jacob(1) &= 1 \\ jacob(n) &= jacob(n-1) + 2 \times jacob(n-2) \quad \text{para } n \geqslant 2 \end{aligned}$$

d) Operaciones aritméticas sobre Naturales

Diseñar una función en Pyhton que calcula de forma recursiva la suma de los primeros n números naturales.

Versión 1: **Recursión simple lineal NO final**

```
1  def suma_n(n):
2      """ Diseño:
3              Número: Natural.
4          Signatura:
5               Nat->Nat
6          Propósito:
7               Calcular en forma recursiva la suma de los primeros n números ↩
                   dados.
8           Ejemplos:
9               suma_n(0) = 0
10              suma_n(1) = 1
11              suma_n(5) = 15
12     """
13     if n == 0:
14       return n
15     else:
16       return (n + suma_n(n-1))
```

Versión 2: **Recursión simple lineal final**

```
1  def suma_cont_n(n,result=0):
2      """ Diseño:
3              Número: Natural.
4          Signatura:
5               Nat Nat ->Nat
6          Propósito:
7               Calcular en forma recursiva la suma de los primeros n números ↩
                   dados.
8           Ejemplos:
9               suma_cont_n(0,0) = 0
10              suma_cont_n(1,0) = 1
11              suma_cont_n(5,0) = 15
12     """
13     if n == 0:
14       return result
15     else:
16       return suma_cont_n(n-1, result+n)
```

EJERCICIO 4. Redefinir la función factorial definida de forma recursiva no final, pero ahora utilizando una recursión final.

EJERCICIO 5. Resuelva los siguientes ejercicios comparativamente, definiendo la versión recursiva e iterativa, compare así cada una de las formas de resolución, evaluándolas en distintos valores, primero en valores menores para luego alcanzar valores naturales más grandes, como por ejemplo: 5,15,25,75,100,500, y 1000.

(a) Diseñar una función que calcule e imprima el resultado de la suma de los primeros 50 números naturales usando una función recursiva.

(b) Diseñar una función que calcule e imprima el resultado de la suma de los primeros n números naturales usando una función recursiva.

(c) Diseñar una función que calcule e imprima el resultado de la suma de los números naturales mayores que n y menores que m usando una función recursiva.

(d) Diseñar una función que calcule de forma recursiva los primeros n múltiplos de 3.

(e) Diseñar una función que calcule de forma recursiva los primeros n múltiplos de k.

EJERCICIO 6. Diseñe funciones en Python que le permitan calcular las siguientes operaciones dadas por sus ecuaciones recursivas. Indique en la declaración de propósito de cada función, las operaciones aritméticas a las que corresponden. Las operaciones se suponen que se calculan sobre los números naturales.

a)

$$div\_nat(n, m) = \begin{cases} 0 & \text{si } m > n \\ div\_nat(n - m, m) + 1 & \text{si } m \leqslant n \end{cases}$$

b)

$$mult\_nat(n, m) = \begin{cases} 0 & \text{si } m = 0 \\ mult\_nat(n, m - 1) + n & \text{si } m \geqslant 1 \end{cases}$$

c)

$$pot\_nat(n, m) = \begin{cases} 1 & \text{si } m = 0 \\ pot\_nat(n, m - 1) \times n & \text{si } m \geqslant 1 \end{cases}$$

d)

$$rest\_nat(n, m) = \begin{cases} 0 & \text{si } m > n \\ n & \text{si } m = 0 \\ rest\_nat(n - 1, m - 1) & \text{si } m \leqslant n \end{cases}$$

Conclusión

Tener en cuenta que la mayoría de los lenguajes de programación soportan la recursión. La recursión nos ofrece la posibilidad de repetición, de procesar un cálculo. La facilidad para resolver un problema mediante un planteo recursivo tiene el coste de la eficiencia de la implementación. Una solución iterativa puede llegar a ser compleja pero eficiente para resolver un problema. Habrá que equilibrar los beneficios de cada técnica de resolución a la hora de utilizarlas.

Entrega

El **ejercicio NRO 2** de esta práctica deberá ser entregado en papel, de puño y letra, a los/as docentes de la cátedra el **MARTES 16/04/2024**. Se evaluará:

- Documentación: Receta y Comentarios
- Solución algorítmica: Estrategia de Resolución
- Comprensión de la recursión y la iteración

Referencias

[1] Think Python: How to Think Like a Computer Scientist, Allen B. Downey, 2nd Edition, Version 2.2.18.

[2] Algoritmos y Programación I, Aprendiendo a programar usando Phyton como herramienta, Rosita Wachenchauzer et.al., 2016, (sin publicar).