G1962 - Programación

Grado en Ingeniería Civil
Problemas 2

Javier González Villa
(19 de diciembre de 2025)

Descomposición, Abstracción y Funciones

Ejercicio 1:

Diseñar una función que solicite como argumento tres valores numéricos y retorne como resultado la media aritmética. Posteriormente, realizar algunas pruebas de llamada a la función para comprobar su correcto funcionamiento.

Ejercicio 2:

Completar la implementación de la siguiente función para que realice las operaciones descritas en la documentación.

```
def combina_strings(s1,s2):
    """ Entrada: dos cadenas s1 y s2 ---> SALIDA: una cadena que entremezcla
                de manera alterna los caracteres de las cadenas
    Requisito previo: las dos cadenas tienen la misma longitud len(s1)==len(2).
    Devuelve una cadena conteniendo los caracteres alternativos de s1 y s2
    comenzando con s1[0], entonces s2[0], s1[1], s2[1],...
    """
```

Ejercicio 3:

Implementa una función que reciba como argumento un número entero n y retorne el cálculo del factorial $n!$ del mismo.

Ejercicio 4:

Escribir el resultado que muestra por pantalla la ejecución del siguiente código y explicar su resultado y orden de ejecución de funciones de manera detallada.

```
def f1(x,y):
    print('f1:',x,y)
    return x+y

def f2(x,y):
    print('f2:',x,y)
    return x*y

print(f1(f2(6,5),f1(2,4)))
```

Ejercicio 5:

Un palíndromo es una palabra que se lee igual de izquierda a derecha, que de derecha a izquierda. Por ejemplo: anna, salas, radar, etc. Diseña una función iterativa que tengan como argumento una cadena y devuelva el booleano *True* si es un palíndromo o *False* en caso contrario.

Ejercicio 6:

El *Cifrado César* o también conocido como cifrado por desplazamiento fue uno de los primeros sistemas de cifrado de la historia. Este sistema consistía en desplazar las letras del abecedario un número dado de veces para obtener un mensaje incomprensible. De manera matemática se puede definir las funciones para encriptar y desencriptar de la siguiente forma:

$$E_n(x) = (x + n)\ mod\ 27 \tag{1}$$

$$D_n(x) = (x - n)\ mod\ 27 \tag{2}$$

En este ejercicio se pide realizar dos métodos uno para cifrar y otro para descifrar que permitan dado como argumentos una cadena de caracteres y una clave numérica entera, obtener el mensaje cifrado o descifrado respectivamente.

Recursión

Ejercicio 1:

Ejecutar el siguiente código que llama a la función recursiva y discutir los resultados retornados por pantalla entendiendo el orden de llamadas, el cambio en los valores introducidos como argumentos en la función y los diversos criterios de parada que tienen las funciones.

```
def funcion_recursiva(n, profundidad):
    if n == 1:
        print("Fin de la función recursiva por límite inferior.")
    elif n == 10:
        print("Fin de la función recursiva por límite superior.")
    elif profundidad == 5:
        print("Fin de la función recursiva por profundidad de llamadas.")
    else:
        print("Llamada a función recursiva para n=",n-1)
        funcion_recursiva(n-1, profundidad+1)
        print("Llamada a función recursiva para n=",n+1)
        funcion_recursiva(n+1, profundidad+1)

funcion_recursiva(5,0)
```

Ejercicio 2:

Implementar una función recursiva que te permita jugar contra tu ordenador al juego *Piedra, Papel o Tijera* de manera que hasta que no ganes al programa, este siga intentando jugar contra

ti de manera automática. Para esto se deberá utilizar únicamente la recursividad de las funciones y en ningún caso bucles de ningún tipo. La selección del usuario se introducirá por teclado, pero la seleción de el ordenador será aleatoria. Para ello puede hacerse uso de la librería *Random* importándola mediante *import random*. Dicha librería tiene una función llamada *random.choice(lista de valores)* que dada una lista de valores retorna una selección al azar.

Ejercicio 3:

Basándonos en la función que determina si una cadena de caracteres es o no un palíndromo que se implementó en el ejercicio previo, en este caso se pide diseñar una función recursiva que tengan como argumento una cadena y devuelva el booleano *True* si es un palíndromo o *False* en caso contrario.

Ejercicio 4:

Implementar una función recursiva que permita resolver el problema de las *Torres de Hanói*. El objetivo del problema es trasladar una pila de discos de diferente tamaño del poste de origen a otro de los dos postes siguiendo ciertas reglas, como que no se puede colocar un disco más grande encima de un disco más pequeño o que los discos solo se pueden mover de uno en uno. Para la solución de este juego de computación es necesario entender como poder subdividir nuestro problema en problemas más pequeños para poder hacer uso de la función de manera recursiva, teniendo en cuenta que el caso de parada mas sencillo que es cuando la $n = 1$ simplemente es mover el disco de la torre origen a la torre destino.

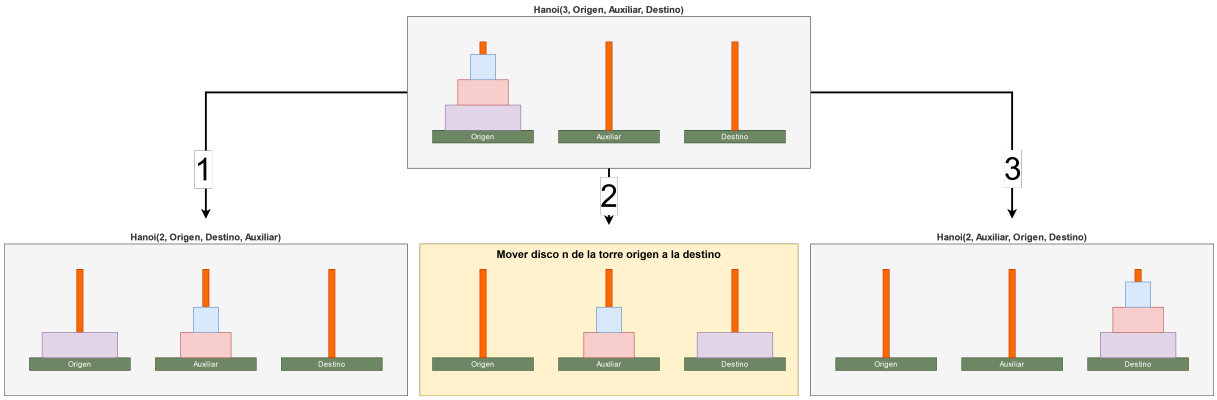

Figura 1: Esquema recursivo de solución de Torres de Hanói para $n = 3$.

La función implementa deberá tomar como argumentos el número de discos, y cuales son las torres origen, destino y auxiliar y deberá pintar por pantalla los pasos necesarios que deberá realizar el usuario para resolver el problema con el menor número de movimientos posibles $2^n - 1$ que está demostrado que sigue una exponencial en función del número de discos.