0. Práctica Recursión

TUIA - Programación 2

Recursión

Ejercicio 0

Escriba una función recursiva `factorial` que tome un numero natural `n` y calcule su factorial `n!`.

Ejercicio 1

1. Escriba una función recursiva `repite_hola` que reciba como parámetro un número entero `n` y escriba por pantalla `n` veces el mensaje `"Hola"`. Invóquela con distintos valores de `n`.
2. Escriba otra función `repite_hola` que reciba como parámetro un número entero `n` y devuelva la cadena formada por `n` concatenaciones de `"Hola"`. Invóquela con distintos valores de `n`.

Ejercicio 2

1. Escriba una función `repite_saludo` que reciba como parámetro un número entero `n` y una cadena `saludo` y escriba por pantalla `n` veces el valor de `saludo`. Invóquela con distintos valores de `n` y de `saludo`.
2. Escriba otra función `repite_saludo` que reciba como parámetro un número entero `n` y una cadena `saludo` y devuelva el valor de `n` concatenaciones de `saludo`. Invóquela con distintos valores de `n` y de `saludo`.

Ejercicio 3

1. Piense cuál sería el resultado de la expresión `misterio(5)` y luego compruebe su hipótesis ejecutándola.
2. Explique con sus palabras qué hace `misterio(a)` para cualquier `a`, sea positivo, negativo o 0.

```
def misterio(a: int) -> int:
    if a == 0:
        return a

    return 1 + misterio(a - 1)
```

Ejercicio 4

Considere la siguiente función recursiva:

```python
def f(n: int, d: int) -> None:
    if n > 1:
        if n % d == 0:
            print(d)
            f(n // d, d)
        else:
            f(n, d + 1)
```

1. ¿Qué muestra por pantalla la llamada `f(30, 2)`? Intente deducirlo sin ejecutar el código.
2. A nivel general, ¿qué muestra por pantalla la llamada `f(x, 2)` para un `x` cualquiera?
3. Implemente una función iterativa equivalente.

Ejercicio 5

Considere la siguiente función recursiva:

```python
def mystery(a: int, b: int) -> int:
    if (b == 0):
        return a
    return mystery(2 * a, b - 1)

result = mystery(7, 3)
print(result)
```

1. ¿Qué muestra por pantalla este código? Intente deducirlo sin ejecutar el código.
2. ¿Cuántas veces se llama recursivamente `mystery` en este código?
3. A nivel general, ¿qué muestra por pantalla la llamada `f(x, 3)` para un `x` cualquiera? ¿Qué muestra por pantalla la llamada `f(x, y)` para `x`, `y` cualesquiera?

Ejercicio 6

Escriba una función recursiva que tome un numero natural `n` e imprima todos los números desde `n` hasta 1.

Ejercicio 7

Convierta la siguiente función recursiva a una iterativa.

```python
def recursiva(t: int, k: int) -> int:
    if t >= 100:
        return k
    else:
        return recursiva(t + k, k + 1)
```

Ejercicio 8

Escriba una función recursiva que calcule el *n-ésimo* número triangular (el número `1 + 2 + 3 + ... + n`).

Ejercicio 9

Escriba una función recursiva que reciba un número positivo `n` y devuelva la cantidad de dígitos que tiene.

Ejercicio 10

Escriba una función recursiva que reciba 2 enteros `n` y `b` y devuelva `True` si `n` es potencia de `b`.

Ejemplos:

- `es_potencia(8, 2) -> True`
- `es_potencia(64, 4) -> True`
- `es_potencia(70, 10) -> False`

Recursión con listas

Ejercicio 11

Escriba una función recursiva que encuentre el mayor elemento de una lista.

Ejercicio 12

Convierta la siguiente función iterativa a una recursiva.

```
def iterativa(l: list[int]) -> int:
    c = 1
    for i in l:
        c = c * i
    return c
```

Ejercicio 13

Escriba una función recursiva para replicar los elementos de una lista una cantidad `n` de veces. Por ejemplo:

Ejemplo:

- `replicar([1, 3, 3, 7], 2) -> ([1, 1, 3, 3, 3, 3, 7, 7])`

Ejercicio 14

Escriba una función que tome una lista y devuelva esa misma lista en orden inverso. Realice dos versiones:

- `reversaR` que resuelva utilizando recursión.
- `reversaI` que resuelva utilizando iteración.

Nota: No utilice la función *built-in* `reversed` en su solución, ni el método `reversed`.

Recursión con strings

Ejercicio 15

Escribir una función recursiva que reciba como parámetros dos cadenas `a` y `b`, y encuentre la posición de la primer ocurrencia de `b` como subcadena de `a`.

Ejemplo:

- `posicion_de("Un tete a tete con Tete", "te") -> 3`
- `posicion_de("Un tete a tete con Tete", "Te") -> 19`

Ayuda: Puede ser de utilidad el método de `str.startswith` que nos dice si una cadena empieza con una subcadena dada. Puede leer más sobre este método en la documentación oficial.

Recursión con wrappers

Al diseñar funciones recursivas muchas veces puede ser útil implementar una función **wrapper**, por ejemplo para cambiar los valores que hay que tomar, validar parámetros, inicializar datos o manejar excepciones.

Se utiliza una función recursiva, llamada función auxiliar, como base y luego una función no recursiva, llamada wrapper, que utiliza la función auxiliar para resolver lo pedido.

Por ejemplo, si necesitamos escribir una función recursiva para calcular el promedio, puede no ser evidente cómo hacerlo:

```
def promediar(lista: list[float]) -> float:
    if len(lista) == 0:
        return ???
    promediar(lista[1:]) ???
```

Para calcular un promedio necesitamos computar tanto una suma como contar la cantidad de elementos. Podemos hacerlo con una función auxiliar:

```
def _promediar(lista: list[float]) -> int, float:
    if len(lista) == 0:
        return 0, 0
    cant, suma = promediar(lista[1:])
    return cant + 1, suma + lista[0]

def promediar(lista: list[float]) -> float:
    cant, suma = _promediar(lista)
    return suma / cant
```

Ejercicio 16

Escriba una función recursiva que dada una cadena determine si en la misma hay más letras `'A'` o letras `'E'`. Utilice una función auxiliar.

Ejercicio 17

Escriba una función `potencia(b,n)` que calcule la potencia para cualquier `n` entero, incluso si `n` es negativa. Utilice una función auxiliar.

Ejercicio 18

Ya sabemos que la implementación recursiva del cálculo del número de Fibonacci `F(0) = 0, F(1) = 1, F(n) = F(n - 1) + F(n - 2)` es ineficiente porque muchas de las ramas calculan reiteradamente los mismos valores.

Escriba una función `fibonacci(n)` que calcule el *n-ésimo* número de Fibonacci de forma recursiva, pero que utilice un diccionario para almacenar los valores ya computados y no computarlos más de una vez.

Nota: Será necesario implementar una función wrapper para cumplir con la signatura de la función pedida.