Guía de Ejercicios 2: Funciones

Objetivos:

- Comprender el concepto de espacio de memoria de una función.
- Incorporar la costumbre de especificar una función antes de programarla, describiendo la cantidad y el tipo de parámetros, el tipo y valor de retorno y los requerimientos sobre los argumentos.
- Ejercitar la construcción de conjuntos de ejemplos de uso, que ayuden a ilustrar el comportamiento esperado de la función, y que luego servirán como casos de test.
- Comenzar a trabajar en la detección de distintos tipos de errores en el código.

Ejercicio 1. Considerar los siguientes programas. Para cada uno, ejecutarlo primero a mano, haciendo un seguimiento detallado de las variables definidas en cada espacio de memoria, y determinar qué se imprime por pantalla. Después ejecutarlo en la computadora y comparar los resultados.

(a)

```
def f(x:int):
    x = x + 1
x:int = 10
f(x)
print(x)
f(x * x)
print(x)
```

(b)

```
def f(x:int) -> int:
    x = x + 1
    return x
x:int = 10
x = f(x)
print(x)
print(x * x)
```

(c)

```
def f(x:int) -> int:
    x = x + 1
    return x
def g(y:int) -> int:
    x:int = f(y)
    return f(x)
x:int = 10
x = g(x)
print(x)
x = g(x * x)
print(x)
```

(d)

```
def f(x:int, y:int) -> int:
    x = 2 * x + y
    return x
x:int = 3
y:int = 7
y = f(y, x)
x = f(y, x)
print(x, y)
print(x, x*x)
```

Ejercicio 2. En el Ejercicio 8 de la Guía 1 escribimos un programa que convierte una temperatura expresada en grados Fahrenheit a grados Celsius. Ahora queremos encapsular ese programa en una función llamada `f2c`, que reciba una temperatura en °F y la devuelva en °C.

(a) Escribir la especificación de la función `f2c`, describiendo los parámetros, las condiciones requeridas sobre los argumentos y el valor de retorno.

(b) Proveer al menos 5 ejemplos que ilustren el comportamiento esperado.

(c) Escribir el código de la función en Python.

(d) Verificar que esta función se ejecute correctamente para los 5 ejemplos elegidos.

Ejercicio 3. Para cada una de las siguientes especificaciones, armar un conjunto de ejemplos (argumentos y valores de retorno) que ilustren el comportamiento de la función y que más adelante servirán como casos de test.

(Hacer este ejercicio en papel. Todavía no se pide pensar los algoritmos ni escribir programas en Python.)

(a)

```
def cant_a(s:str) -> int:
    ''' Requiere: Nada
        Devuelve: La cantidad de ocurrencias de 'a' en s.
    '''
```

(b)

```
def mayus_n(s:str) -> str:
    ''' Requiere: Nada
        Devuelve: Una copia de s pero con todas las ocurrencias
                  de la letra 'n' en mayúscula.
    '''
```

(c)

```
def raiz_entera(n:int) -> int:
    ''' Requiere: n>=0
        Devuelve: La parte entera de la raíz cuadrada de n.
    '''
```

Ejercicio 4. Para cada uno de los siguientes problemas, **especificar** una función que lo resuelva, describiendo los parámetros, las condiciones requeridas sobre los argumentos y el valor de retorno. Armar además un conjunto de ejemplos (argumentos y valores de retorno) que ilustren el comportamiento de la función y que más adelante servirán como casos de test.

(Hacer este ejercicio en papel. Todavía no se pide pensar los algoritmos ni escribir programas en Python.)

(a) Dado un entero $n \geq 0$, calcular $n!$ (factorial de n).

(b) Dados dos enteros $n, k \geq 0$ (con $k \leq n$), calcular el combinatorio $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

(c) Dado un entero $n \geq 0$, devolver un string con los valores de $\binom{n}{i}$ para todo $0 \leq i \leq n$, separados por comas. Por ejemplo, para $n = 4$, debe devolver `"1,4,6,4,1"`.

(d) Dado un entero $n \geq 0$, devolver un string con una línea de n asteriscos.

(e) Dado un entero $n \geq 0$, devolver un string que al imprimirse forme un cuadrado de asteriscos de lado n.

(f) Dado un string, devolver su inversa. Por ejemplo, para `"qwerty"` debe devolver `"ytrewq"`.

(g) Dados dos strings, devolver la cantidad de veces que el primero está contenido en el segundo.

Ejercicio 5. El código del archivo `errores.py` tiene errores de tres tipos distintos: de sintaxis, de tiempo de ejecución (*runtime*) y semánticos. Inspeccionar el código, ejecutarlo y debuggearlo, a fin de encontrar al menos 7 errores, clasificándolos por tipo.