

# Fundamentos de Programación

## Cuaderno de Trabajo 3b

### Ejercicios propuestos

1. Ya vimos que la validación de entradas es algo tan presente que no podemos escapar de ello, así que si te parece vamos a escribir una función que solicite al usuario introducir un entero y que no pare de pedírselo hasta que la información introducida sea válida.

La idea es usar la construcción `while` combinada con la función de validación programada en cuadernos anteriores, consiguiendo una función cuya cabecera sería la siguiente:

```
def leer_entero_validado():  
    """ None --> int  
    OBJ: Solicita un entero al usuario, lo valida y lo retorna sólo  
    cuando se ha asegurado de que es realmente un entero """
```

2. Una mejora útil, al anterior ejercicio, es programar una función `entero_pedido` (`min`, `max`, `msj`) que pedirá un entero pero limitará su rango, es decir, el entero deberá estar comprendido entre un máximo y un mínimo. Además, la función recibirá un mensaje que será el que empleará para interactuar con el usuario.

Como ayuda, te ponemos aquí cómo sería el probador:

```
#Probador  
min = 1  
max = 12  
print(entero_pedido(min, max, f'mes entre {min} y {max}: '))
```

3. Escribe un programa que, después de preguntar ¿Cuántos números se van a introducir?, pida esos números (enteros o reales) y devuelva su media aritmética, el mayor y el menor. El programa debe controlar que la cantidad de números es mayor de 2 y en caso contrario ha de mostrar un mensaje de error. Como siempre, valida las entradas.

```
import bib_input  
  
cuantos = bib_input.leer_entero('¿Cuántos números se van a introducir? ')  
if cuantos <= 2:  
    print('Tienes que poner más de dos números')  
else:  
    numero = bib_input.leer_entero('Introduce un número: ')  
    suma = numero  
    mayor = numero  
    menor = numero
```

```

for i in range(1, cuantos):
    numero = bib_input.leer_entero('Introduce un número: ')
    suma += numero
    if numero > mayor:
        mayor = numero
    if numero < menor:
        menor = numero
media = suma / cuantos
print(f'El mayor es {mayor}')
print(f'El menor es {menor}')
print(f'La media es {media}')

```

4. Escribe un programa que lea una serie de números enteros hasta que se introduzca el número -9999, y cuente el total de números introducidos, el total de valores positivos y el total de valores negativos (no consideres el cero ni positivo ni negativo). Reutiliza la función que hayas diseñado en el Ejercicio 1 para validar tus entradas.

```

import bib_input

total = 0
positivos = 0
negativos = 0
numero = bib_input.leer_entero('Introduce un número: ')
while numero != -9999:
    total += 1
    if numero > 0:
        positivos += 1
    elif numero < 0:
        negativos += 1
    numero = bib_input.leer_entero('Introduce un número: ')

print(f'Has introducido {total} números.')
print(f'Positivos: {positivos}')
print(f'Negativos: {negativos}')

```

5. Implementa una variante del *Ejercicio Resuelto 3* que dibuje en pantalla un rectángulo de dimensiones dadas pero con dos símbolos, las filas pares con '@' y las impares con '#'. Esta será la versión "3a". Después de hacerlo, crea una segunda versión "3b" para que esta vez los símbolos se alternen en las columnas, es decir, las columnas pares se dibujarán con '@' y las impares con '#'.

```

def es_par(num):
    """int --> bool
    OBJ: Devuelve si un número es par
    """
    return num % 2 == 0

def dibujar_rectangulo(a,b):
    """ int,int -> None
    OBJ: Dibuja en pantalla un rectángulo de dimensiones a por b
    utilizando la @ en las filas pares
    y la # en las impares
    """
    for i in range(a):
        for j in range(b):
            if es_par(i):
                print('@', end='')
            else:

```

```

        print('#', end='')
    print()

```

```
dibujar_rectangulo(5,5)
```

6. Escribe un programa que pida un número límite y calcule cuántos términos de la serie armónica son necesarios para que su suma supere dicho límite. Es decir, dado un límite se trata de determinar el menor número  $n$  tal que:

$$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} > \text{límite}$$

Prueba tu código, y ten en cuenta que para valores altos del límite el tiempo de cálculo se dispara. Así, para un límite = 5,  $n$  sería 83; para 10 ya asciende a 12.367 y para 15, ¡el número de términos es 1.835.421!! El programa ha de ser robusto y controlar que el número introducido es un entero positivo.

```

import bib_input

limite = bib_input.leer_real('Introduce el límite: ')
if limite > 0:
    contador = 0
    suma_armonica = 0
    while suma_armonica < limite:
        termino = 1 / (contador+1)
        suma_armonica += termino
        contador +=1
    print(f'El número de terminos es: {contador}')
else:
    print('El límite es menor que 0.')

```

7. Escribe un programa que lea un entero positivo  $n$  y genere una tabla con las  $n$  primeras potencias de 2, 3 y 5. Así:

1	2	4	8	16	32	64 ...
1	3	9	27	81	243	729 ...
1	5	25	125	625	3125	15625 ...

```

import bib_input

n = bib_input.leer_entero('Introduce un entero positivo: ')
if n > 0:
    for potencia in [2,3,5]:
        for exponente in range(n):
            print(f'{potencia**exponente:10}', end='')
        print()

```

8. Repasa los formatos de salida, de Python, y codifica un programa que genere la siguiente salida:

Tabla de multiplicar  
=====

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

9. Escribe un programa que acepte un año escrito en cifras arábicas y lo visualice escrito en números romanos, dentro del rango 1 a 2000. (I=1, V=5, X=10, L=50, C=100, D=500, M=1000).
10. Cada país acuña billetes y monedas de unos determinados valores. Por ejemplo, en los países de la Unión Europea los valores son: 500.0, 200.0, 100.0, 50.0, 20.0, 10.0, 5.0, 2.0, 1.0, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01 euros. Diseña un subprograma para una caja de un supermercado, el cual reciba un importe en euros y la cantidad pagada por el cliente en una compra y muestre al cajero cuántas piezas (billetes y monedas) y de qué tipo debe dar como cambio. Dicha salida no deberá en ningún caso especificar piezas para las que haya que dar cero unidades.

```
import bib_input

CAMBIO = [500.0, 200.0, 100.0, 50.0, 20.0, 10.0, 5.0, 2.0, 1.0, 0.5,
0.2, 0.1, 0.05, 0.02, 0.01]

dinero = bib_input.leer_real('Introduce la cantidad en €: ')
for moneda in CAMBIO:
    num_monedas = int(dinero // moneda)
    dinero = round(dinero % moneda, 2)
    if num_monedas > 0:
        if moneda < 5:
            print(f'Monedas de {moneda} euros: {num_monedas}')
        else:
            print(f'Billetes de {moneda} euros: {num_monedas}')
```

11. Escribe un programa, ahora con estructuras repetitivas, que muestre un menú en pantalla que permita calcular el seno, coseno, tangente, cotangente, secante y cosecante de un ángulo. El menú se mostrará hasta que el usuario decida salir, y mientras permitirá hacer todas las operaciones que desee el usuario. El menú quedará así:

```
1. Seno
2. Coseno
...
0. Salir
Elija una opción:
```

Sugerencia: El menú podría estar centrado, utiliza las bibliotecas ya realizadas.

12. La varianza poblacional de una población  $p$ , que dispone de  $n$  elementos se define como:

$$\sum_{i=0}^n \frac{(p_i - p_{media})^2}{n-1}$$

Construye un subprograma que devuelva la varianza de una población que recibe como parámetro.

13. Mejorar el *Ejercicio Resuelto 4*, modularizándolo y poniendo un límite al número de veces que el usuario puede intentar acertar el símbolo oculto.
14. Estudia la convergencia de la siguiente serie introduciendo un número creciente de términos y observando su tendencia. Implementa una función para la serie y un código que vaya analizando su convergencia. Comenta tus resultados.

$$\sum_{n \geq 1} \frac{(n!)^2}{2^{n^2}}$$

15. Escriba un programa modularizado en Python que genere 5 frases a partir de palabras en la siguiente lista:

```
['perro', 'niño', 'nube', 'padre', 'es', 'esta', 'come', 'mira', 'ama', 'el', 'la', 'al', 'en']
```

Las frases, que deben tener entre 3 y 10 palabras, deben generarse eligiendo un número aleatorio de palabras cada vez, de modo que la primera frase puede tener 3 palabras, mientras que la segunda podría tener 6 palabras. Una vez generada una frase, el programa la mostrará por pantalla. Y así hasta 5 frases en total.

```
import random

PALABRAS =
['perro', 'niño', 'nube', 'padre', 'es', 'esta', 'come', 'mira', 'ama', 'el', 'la', 'al', 'en']

for i in range(5):
    frase = ''
    num_palabras = random.randint(3, 10)
    for j in range(num_palabras):
        palabra = PALABRAS[random.randint(0, len(PALABRAS)-1)]
        frase += palabra + ' '
    print(frase)
```

16. Mejore el programa anterior para que genere, no 5 frases, sino tantas como sea necesario hasta que alguna de ellas tenga sentido. Para ello, después de generar una frase e imprimirla por pantalla se preguntará al usuario si considera que la frase generada tiene sentido y, si su respuesta es afirmativa, mostrará un mensaje final indicando el número de frases generadas hasta que se obtuvo una con sentido.