

# Evaluación Integradora

## Clave de corrección y rúbrica

### Parte A • Clave y justificación breve

| Ítem | Sem. | Tipo           | Clave                          | Justificación                                                                                                                                                                                                                                                                                         |
|------|------|----------------|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | S1   | Opc. simple    | <b>a</b>                       | Correcta a). La distinción esencial es memoria/bus único (Von Neumann) vs. memorias y buses separados (Harvard). b) invierte la propiedad de auto-modificación; c) invierte los ámbitos de uso; d) los x86 son Von Neumann con caché L1 separada (Harvard modificada).                                |
| 2    | S1   | Opc. simple    | <b>b</b>                       | Correcta b). La representación binaria finita de IEEE 754 impide expresar exactamente 0,1 y 0,2; su suma no coincide bit a bit con 0,3. Por eso no se comparan <code>float</code> con <code>==</code> .                                                                                               |
| 3    | S2   | Opc. simple    | <b>b</b>                       | Correcta b). CPython compila el fuente a bytecode y lo interpreta sobre una máquina virtual; el intérprete es un programa nativo. El bytecode es portable, no es código de máquina del procesador.                                                                                                    |
| 4    | S3   | Resp. múltiple | <b>a, b, c</b>                 | Correctas a), b) y c). El cortocircuito hace de guarda; la separación <code>for/while</code> es la convención del curso; De Morgan es correcta. d) es falsa: no todo bucle indeterminado se expresa naturalmente con <code>for</code> .                                                               |
| 5    | S4   | Opc. simple    | <b>b</b>                       | Correcta b). La precondition es exigencia al llamador; la postcondición, garantía de la función bajo esa exigencia. Si la precondition se cumplió y la postcondición no, el error está en la función.                                                                                                 |
| 6    | S4   | Resp. múltiple | <b>a, c</b>                    | Correctas a) y c). b) es falsa: las globales aumentan el acoplamiento; d) es falsa: la cohesión mide unidad interna y el acoplamiento, dependencia entre módulos; se busca alta cohesión y bajo acoplamiento.                                                                                         |
| 7    | S5   | Opc. simple    | <b>b</b>                       | Correcta b). Su precondition es el orden; el descarte de mitades da $O(\log n)$ ; la terminación (rango vacío o elemento hallado) es indeterminada, propia de <code>while</code> .                                                                                                                    |
| 8    | S5   | Emparej.       | <b>1-B, 2-E, 3-C, 4-A, 5-D</b> | 1-B, 2-E, 3-C, 4-A, 5-D. El acceso indexado es constante; la binaria descarta mitades (logarítmica); la lineal recorre $n$ ; merge sort es $n \log n$ ; los ordenamientos cuadráticos comparan todos los pares.                                                                                       |
| 9    | S6   | Resp. múltiple | <b>a, b, c</b>                 | Correctas a), b) y c). d) es falsa: la estabilidad es una propiedad estructural, no de complejidad; merge sort es estable y $O(n \log n)$ .                                                                                                                                                           |
| 10   | S6   | Opc. simple    | <b>b</b>                       | Correcta b). El <code>with</code> cierra el recurso pase lo que pase; declarar el encoding evita depender del locale. <code>'w'</code> trunca; <code>'r'</code> no permite escribir; la iteración conserva el <code>'\n'</code> .                                                                     |
| 11   | S7   | Resp. múltiple | <b>a, b, c</b>                 | Correctas a), b) y c). d) es falsa: la copia superficial duplica solo el contenedor externo; las sublistas siguen compartidas (haría falta <code>copy . deepcopy</code> ).                                                                                                                            |
| 12   | S8   | Opc. simple    | <b>b</b>                       | Correcta b). <code>'&lt;'</code> evita el padding; <code>calcsz</code> es la única fuente del tamaño; los registros fijos habilitan el acceso directo por fórmula. <code>pack</code> rellena con <code>b'\x00'</code> , no con espacios; y hardcodear el tamaño contradice la fuente única de verdad. |
| 13   | S8   | Resp. múltiple | <b>a, b, c</b>                 | Correctas a), b) y c). d) es falsa: sin longitud fija no hay fórmula para el offset del $k$ -ésimo registro; hay que recorrer secuencialmente.                                                                                                                                                        |
| 14   | S9   | Opc. simple    | <b>b</b>                       | Correcta b). Caso base + acercamiento monótono garantizan terminación. Puede haber varias llamadas recursivas (merge sort,                                                                                                                                                                            |

| Ítem | Sem. | Tipo           | Clave                               | Justificación                                                                                                                                                                                      |
|------|------|----------------|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      |      |                |                                     | Hanói); break sale de bucles, no de funciones; el caso base debe evaluarse primero.                                                                                                                |
| 15   | S9   | Resp. múltiple | <b>a, c, d</b>                      | Correctas a), c) y d). b) es falsa: CPython no optimiza la recursión de cola; cada llamada crea un frame y hay un límite (por defecto 1000).                                                       |
| 16   | S10  | Emparej.       | <b>1-A, 2-B, 3-C, 4-D, 5-B, 6-C</b> | 1-A, 2-B, 3-C, 4-D, 5-B, 6-C. La tupla, por inmutable y hasheable, sirve como clave (5); el diccionario agrega frecuencias por clave (6); el conjunto deduplica y opera algebraicamente (4).       |
| 17   | S11  | Resp. múltiple | <b>a, b, c</b>                      | Correctas a), b) y c). d) es falsa: la DP solo gana cuando hay subproblemas superpuestos; si no se repiten, almacenar resultados no acelera nada.                                                  |
| 18   | S12  | Opc. simple    | <b>b</b>                            | Correcta b). La poda es la diferencia esencial con la fuerza bruta. c) invierte los criterios (D&V: independientes; DP: superpuestos); d) es falsa: sin subproblemas superpuestos no hay ganancia. |

Subtotal Parte A: 50 puntos. Opc. simple: 2 ptos c/u · Resp. múltiple: 3 ptos c/u (repartidos por igual entre sus opciones) · Emparejamiento ítem 8: 5 ptos (1 por par) · ítem 16: 6 ptos (1 por par).

Cobertura por semana: S1 ítems 1–2 · S2 ítem 3 · S3 ítem 4 · S4 ítems 5–6 · S5 ítems 7–8 · S6 ítems 9–10 · S7 ítem 11 · S8 ítems 12–13 · S9 ítems 14–15 · S10 ítem 16 · S11 ítem 17 · S12 ítem 18.

## Parte B · Solución de referencia (verificada) y rúbrica

La solución de referencia procesa el archivo transaccional en una única pasada: no lo carga entero en memoria, sino que construye simultáneamente el diccionario de préstamos por categoría, el conjunto de socios distintos y el conteo total. Delega todos los cálculos en procesar\_prestamos y la presentación en informar. La categoría más solicitada no requiere una función aparte: al ordenar el ranking de mayor a menor, la posición 0 ya es una categoría de máxima frecuencia. El orden descendente se obtiene por el orden natural de tuplas (cantidad, categoría), sin función lambda. Fue ejecutada y produce, sobre el archivo de ejemplo de 7 préstamos: total = 7, socios distintos = 4, Ingeniería = 4, Matemática = 2, Literatura = 1, categoría más solicitada = Ingeniería. Casos borde verificados: archivo vacío (informa 0 y «No hay préstamos registrados») y empate (devuelve una de las máximas).

```
# =====
# Analisis de prestamos de la biblioteca
# Algoritmos y Programacion I / Fundamentos de Programacion - FIUBA
# =====

# ----- Seccion Declarativa -----
ARCHIVO = 'prestamos.txt' # ruta del archivo de entrada
SEP = ';'                 # separador de campos

# ----- Funciones -----
def procesar_prestamos(ruta):
    """
    Pre: 'ruta' refiere a un archivo de texto transaccional, con una
        línea por préstamo y formato 'fecha;socio;categoría;título'.
    Pos: recorre el archivo UNA sola vez, sin cargarlo entero en memoria,
        y devuelve la tupla (total, cantidad_socios, ranking), donde
        'ranking' es la lista de pares (cantidad, categoría) ordenada
        de mayor a menor cantidad de préstamos.
    """
    # --- Prologo: acumuladores de la única pasada ---
    total = 0          # cantidad de préstamos leídos
    socios = set()      # socios distintos (deduplica)
    conteo = {}         # préstamos por categoría

    # --- Resolución: una sola pasada, sin materializar la colección ---
    with open(ruta, encoding='utf-8') as archivo:
        for linea in archivo:
            linea = linea.strip() # quita salto de línea
            if linea != '': # se ignoran líneas vacías
                fecha, socio, categoria, titulo = linea.split(SEP)
                total = total + 1
                socios.add(socio)
                conteo[categoria] = conteo.get(categoria, 0) + 1

    # --- Epílogo: ranking por natural ordering de tuplas (mayor a menor) ---
    pares = [(cantidad, categoria) for categoria, cantidad in conteo.items()]
    ranking = sorted(pares, reverse=True)
    return total, len(socios), ranking

def informar(total, cantidad_socios, ranking):
    """
    Pre: 'total' y 'cantidad_socios' son enteros >= 0; 'ranking' es la
        lista de pares (cantidad, categoría) ordenada de mayor a menor.
    Pos: imprime por salida estándar el informe de actividad; si no hubo
        préstamos, informa esa condición.
    """
    print('Total de préstamos:', total)
    print('Socios distintos:', cantidad_socios)
    print('Préstamos por categoría (mayor a menor):')
```

```

for cantidad, categoria in ranking:
    print(' ', categoria, '->', cantidad)
if ranking != []: # el primero es una de las categorías máximas
    print('Categoría mas solicitada:', ranking[0][1])
else:
    print('No hay préstamos registrados.')

# ----- Sección Algorítmica -----
def main():
    # --- Prólogo: la lectura y el cómputo se delegan por completo ---
    #     en procesar_prestamos (una única pasada sobre el archivo).

    # --- Resolución ---
    total, cantidad_socios, ranking = procesar_prestamos(ARCHIVO)

    # --- Epílogo ---
    informar(total, cantidad_socios, ranking)

main()

```

*Variantes aceptables: para el orden descendente, `sorted(conteo.items(), key=operator.itemgetter(1), reverse=True)` o la forma con `key=Lambda` `par: par[1]` empleada en el material de la Semana 14; obtener el máximo con `max(conteo, key=conteo.get)` en lugar de leerlo de la posición 0. Es igualmente válido separar la lectura de los cómputos en dos funciones (la resolución del epílogo de `procesar_prestamos`). Se penaliza el uso de `break/continue`, leer sin `with`, o usar variables globales para comunicar funciones.*

### Rúbrica del problema (50 puntos)

| Fase (Pólya)        | Ptos      | Criterio                                                                                                                                                                                                                                   |
|---------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Análisis</b>     | <b>8</b>  | Identifica entradas (archivo CSV) y salidas (informe); enuncia casos de prueba incluyendo borde (archivo vacío, una categoría, empate).                                                                                                    |
| <b>Diseño</b>       | <b>14</b> | Elige diccionario para la agregación por categoría y conjunto para deduplicar socios; descompone en funciones cohesivas con pre/postcondición; define el contrato antes de codificar.                                                      |
| <b>Codificación</b> | <b>20</b> | Programa correcto y con la estructura modular del curso (Declarativa / Funciones / Prólogo-Resolución-Epílogo); usa <code>with</code> , <code>for/while</code> según corresponde, sin <code>break/continue</code> ; documentación interna. |
| <b>Evaluación</b>   | <b>8</b>  | Indica con qué casos se valida la solución y verifica el comportamiento en los casos borde.                                                                                                                                                |

### Grilla de puntaje

| Componente                  | Máximo     | Obtenido |
|-----------------------------|------------|----------|
| Parte A · Teoría (18 ítems) | <b>50</b>  |          |
| Parte B · Programación      | <b>50</b>  |          |
| <b>TOTAL</b>                | <b>100</b> |          |

Para aprobar se debe obtener al menos 20 puntos en cada parte.