

Evaluación Integradora

Examen escrito presencial · Duración estimada: 120 minutos · Sin material ni dispositivos

Apellido y nombre:	Padrón:	Nro. de Curso:
Fecha:	Aula:	Firma:

Instrucciones generales

La evaluación tiene dos partes con igual peso. Parte A — Teoría (50 puntos) y Parte B — Competencia en programación (50 puntos). Total: 100 puntos.

- En los ítems de opción múltiple con una sola respuesta, encerrará en un círculo la única letra correcta.
- En los ítems de respuesta múltiple, cada opción se evalúa como verdadero/falso independiente: encerrará las letras que correspondan. El puntaje del ítem se reparte por igual entre sus opciones; cada opción resuelta correctamente suma su fracción, cada opción marcada incorrectamente resta la mitad de la fracción de una válida.
- En los ítems de emparejamiento, escribí en cada línea la letra de la opción elegida.
- En el problema de programación, respetá las convenciones del curso: estructura modular (Sección Declarativa → Funciones con docstring Pre/Pos → Prólogo / Resolución / Epílogo), manejo de archivos con `with`, `for` para iteración de cantidad conocida y `while` con condición compuesta para la indeterminada, sin `break` ni `continue`.

Escribí con letra clara. Se evalúa el análisis y el diseño, no solo el código final.

Parte A · Teoría

18 ítems · 50 puntos. Cubre los contenidos teóricos de las Semanas 1 a 12. Se deben sumar al menos 20 puntos para aprobar.

1. ¿Cuál afirmación describe correctamente la diferencia esencial entre la arquitectura de Von Neumann y la arquitectura Harvard? (2 pts · una opción correcta)

- En Von Neumann, datos e instrucciones comparten una única memoria y un mismo bus; en Harvard residen en memorias separadas, con buses independientes que permiten accesos simultáneos.
- En Harvard el programa puede modificarse a sí mismo durante la ejecución, mientras que en Von Neumann no.
- Von Neumann es exclusiva de microcontroladores y DSP; Harvard, de los computadores de propósito general.
- Los procesadores x86 actuales son Harvard puros, sin ningún rasgo de Von Neumann.

2. Un estudiante observa que en Python `0.1 + 0.2 == 0.3` devuelve `False`. ¿Cuál es la explicación correcta? (2 pts · una opción correcta)

- Es un error del operador de suma de Python para números `float`.
- Los `float` usan el estándar IEEE 754 de doble precisión y no representan exactamente ciertos decimales; el resultado difiere de `0.3` en un `epsilon`.
- `0.3` se almacena como cadena y `0.1 + 0.2` como número, por eso no son iguales.
- Para que dé `True` hay que declarar las variables como `int`.

3. Sobre la ejecución de un programa Python bajo CPython, ¿cuál afirmación es correcta? (2 pts · una opción correcta)

- El procesador ejecuta directamente el código fuente Python, línea por línea, sin ninguna traducción intermedia.
- El código fuente se compila a `bytecode`, que la máquina virtual de Python interpreta; ese intérprete sí está compilado a código de máquina nativo.
- Python se compila íntegramente a un ejecutable de código de máquina antes de correr, igual que C.
- El `bytecode` de Python es código de máquina del procesador, específico de cada arquitectura.

4. Sobre las estructuras de control y las condiciones lógicas según las convenciones del curso, ¿cuáles afirmaciones son correctas? (3 pts · una o más opciones correctas)

- `if x != 0 and y % x == 0`: aprovecha la evaluación de cortocircuito de `and` para no evaluar el módulo cuando `x` vale 0.
- `for i in range(n)`: es idiomático para iteración de cantidad conocida; `while` con condición compuesta, para iteración indeterminada.
- Por las leyes de De Morgan, `not (a or b)` equivale a `(not a) and (not b)`.
- Reescribir cualquier `while` como `for` siempre mejora la legibilidad y la eficiencia.

5. En el modelo modular del curso, ¿qué describe correctamente la relación entre precondition y postcondition de una función? (2 pts · una opción correcta)

- La precondition es lo que la función garantiza; la postcondition, lo que exige de quien la llama.
- La precondition es lo que la función exige de quien la llama; la postcondition, lo que la función garantiza siempre que la precondition se haya cumplido.
- Ambas deben verificarse obligatoriamente con `assert` o el programa no compila.
- Si la precondition se cumple y la postcondition no, el error es de quien llamó a la función.

6. Sobre cohesión, acoplamiento y paso de parámetros en el diseño modular, ¿cuáles afirmaciones son correctas? (3 pts · una o más opciones correctas)

- a) Una función de alta cohesión hace una sola cosa bien delimitada, lo que favorece su comprensión, reuso y testeo.
- b) Comunicar funciones mediante variables globales reduce el acoplamiento del sistema.
- c) Pasar como parámetros explícitos los datos que la función necesita la hace autocontenida y testeable.
- d) Cohesión y acoplamiento son sinónimos: miden lo mismo desde perspectivas opuestas.

7. Sobre la búsqueda binaria, ¿cuál afirmación es correcta? (2 pts · una opción correcta)

- a) Funciona sobre cualquier secuencia, esté ordenada o no.
- b) Exige que la secuencia esté ordenada; su costo es $O(\log n)$ y se implementa idiomáticamente con un `while` de condición compuesta.
- c) Es siempre más rápida que la búsqueda lineal, sin importar el tamaño ni el orden de los datos.
- d) Se implementa con `for i in range(n)` recorriendo todos los elementos.

8. Asocia cada operación o algoritmo con su complejidad temporal asintótica (peor caso). Escribí la letra correspondiente junto a cada número. Cada opción se usa exactamente una vez. (5 pts · emparejamiento)

Opciones: A. $O(1)$ B. $O(\log n)$ C. $O(n)$ D. $O(n \log n)$ E. $O(n^2)$

- | | |
|---|--------|
| 1. Búsqueda binaria en una lista ordenada | → ____ |
| 2. Ordenar con <code>selection sort</code> o <code>bubble sort</code> | → ____ |
| 3. Búsqueda lineal en una lista | → ____ |
| 4. Acceder a <code>datos[i]</code> (por índice en una lista) | → ____ |
| 5. Ordenar con <code>merge sort</code> | → ____ |

9. Sobre los algoritmos de ordenamiento, ¿cuáles afirmaciones son correctas? (3 pts · una o más opciones correctas)

- a) Un algoritmo estable conserva el orden relativo entre elementos de clave igual.
- b) `merge sort` es $O(n \log n)$ en todos los casos; `quick sort` es $O(n \log n)$ en promedio pero $O(n^2)$ en el peor caso.
- c) `selection sort` y `bubble sort` son $O(n^2)$ en el caso general.
- d) La estabilidad hace que un algoritmo sea estrictamente más lento que su versión inestable.

10. Sobre el manejo de archivos de texto en Python según las convenciones del curso, ¿cuál afirmación es correcta? (2 pts · una opción correcta)

- a) El modo `'w'` agrega contenido al final del archivo existente sin borrar lo previo.
- b) `with open(ruta, encoding='utf-8') as f:` garantiza el cierre del archivo aun si ocurre una excepción dentro del bloque, y hace portable la lectura entre configuraciones regionales.
- c) Un archivo abierto en modo `'r'` permite escribir en él si se decide a mitad del programa.
- d) Iterar `for linea in f:` elimina automáticamente el `'\n'` final de cada línea.

11. En el modelo de memoria de Python (toda variable es una referencia a un objeto), ¿cuáles afirmaciones son correctas? (3 pts · una o más opciones correctas)

- a) `is` compara identidad (mismo objeto, mismo `id()`), mientras que `==` compara valor.
- b) Tras `a = [1, 2, 3]`; `b = a`; `b.append(4)`, la lista `a` queda con cuatro elementos por aliasing.
- c) Para comparar contra `None` se recomienda `x is None` en lugar de `x == None`.
- d) Una copia superficial de una lista de listas copia también, de forma independiente, las sublistas internas.

12. Sobre el módulo `struct` y los registros de longitud fija (Semana 8), ¿cuál afirmación es correcta? (2 pts · una opción correcta)

- a) `struct.pack` rellena las cadenas más cortas con espacios en blanco (`b' '`) hasta el tamaño declarado.
- b) Con el formato precedido de `'<'` no hay padding de alineación; `TAM_REGISTRO = struct.calcsize(FORMATO)` es la fuente única de verdad y el offset del `k`-ésimo registro es `k * TAM_REGISTRO`.
- c) `struct` es la única forma de leer bytes de un archivo binario en Python.
- d) El tamaño del registro debe escribirse como literal numérico para evitar recalcularlo en cada ejecución.

13. Sobre el acceso a archivos binarios de registros de longitud fija con `seek` y `tell`, ¿cuáles afirmaciones son correctas? (3 pts · una o más opciones correctas)

- a) Leer el `k`-ésimo registro con `seek(k * TAM_REGISTRO)` tiene costo $O(1)$, independiente de `k`.
- b) `seek` permite posicionarse en cualquier offset, hacia adelante o hacia atrás.
- c) `tell()` devuelve la posición actual del cursor, en bytes desde el inicio del archivo.
- d) El acceso directo por fórmula funciona igual de bien en archivos de registros de longitud variable.

14. Para que una función recursiva sea correcta y termine, ¿cuál condición es necesaria? (2 pts · una opción correcta)

- a) Que tenga exactamente una llamada recursiva en su cuerpo.
- b) Que exista al menos un caso base y que cada llamada recursiva se acerque estrictamente a él según una medida bien definida.
- c) Que use `break` al alcanzar el caso base.

d) Que la llamada recursiva preceda al caso base en el cuerpo de la función.

15. Sobre la ejecución de funciones recursivas y la pila de ejecución, ¿cuáles afirmaciones son correctas? (3 ptos · una o más opciones correctas)

- a)** Cada llamada crea un frame en la pila con parámetros, variables locales y punto de retorno; hay tantos frames simultáneos como la profundidad alcanzada.
- b)** CPython optimiza la recursión de cola reutilizando el mismo frame, por lo que no existe límite de profundidad.
- c)** fib ingenuo genera un árbol de llamadas de tamaño exponencial aunque su profundidad sea lineal en n .
- d)** Superar el límite de profundidad de recursión produce un RecursionError.

16. Asociá cada escenario con la estructura de datos más apropiada para resolverlo. Escribí la letra junto a cada número. Las opciones pueden repetirse o no usarse. (6 ptos · emparejamiento)

Opciones: A. Lista B. Tupla C. Diccionario D. Conjunto

- 1. Mantener una colección ordenada y modificable de valores, con acceso por posición → ____
- 2. Devolver desde una función varios valores heterogéneos relacionados que no van a cambiar → ____
- 3. Indexar registros por una clave única para acceso $O(1)$ promedio en búsquedas frecuentes → ____
- 4. Mantener una colección sin duplicados sobre la que operar con unión, intersección y diferencia → ____
- 5. Almacenar las coordenadas (x, y) de un punto que se usará como clave de un diccionario → ____
- 6. Contar la frecuencia de cada palabra distinta de un texto → ____

17. Sobre las técnicas algorítmicas iterativas avanzadas (Semana 11), ¿cuáles afirmaciones son correctas? (3 ptos · una o más opciones correctas)

- a)** La técnica de dos punteros puede reducir un algoritmo de $O(n^2)$ a $O(n)$ cuando los datos cumplen una propiedad estructural (por ejemplo, están ordenados).
- b)** La ventana deslizante mantiene un invariante actualizándolo de forma incremental al mover los extremos, en lugar de recalcular desde cero.
- c)** La memorización (top-down) y la programación dinámica iterativa (bottom-up) evitan recomputar subproblemas superpuestos.
- d)** La programación dinámica aporta ganancia de complejidad aunque los subproblemas sean independientes (sin superposición).

18. Sobre las técnicas recursivas avanzadas (Semana 12), ¿cuál afirmación es correcta? (2 ptos · una opción correcta)

- a)** Backtracking y fuerza bruta hacen exactamente el mismo trabajo computacional; difieren solo en estilo.
- b)** La poda en backtracking evita explorar ramas que no pueden conducir a una solución válida, reduciendo el árbol de búsqueda respecto de la fuerza bruta.
- c)** Divide y vencerás se aplica cuando hay subproblemas superpuestos; la programación dinámica, cuando son independientes.
- d)** Toda recursión, sin importar su estructura, puede convertirse en DP iterativa con ganancia de complejidad.

Parte B · Competencia en programación

Un problema · 50 puntos.

Análisis de préstamos de la biblioteca

La biblioteca de la facultad mantiene un archivo de texto `prestamos.txt` donde cada línea registra un préstamo con el formato `fecha;socio;categoria;titulo` (campos separados por punto y coma, sin línea de encabezado). Por ejemplo:

```
2026-03-10;A042;Ingenieria;Estructuras de datos
2026-03-10;B017;Matematica;Calculo I
2026-03-11;A042;Ingenieria;Algoritmos
```

Se pide escribir un programa modular en Python que, leyendo ese archivo, produzca un informe de actividad. El programa debe:

1. Leer el archivo `prestamos.txt` usando `with`.
2. Calcular, mediante un diccionario, la cantidad de préstamos por categoría.
3. Determinar, mediante un conjunto, la cantidad de socios distintos que realizaron al menos un préstamo.
4. Identificar la categoría con mayor cantidad de préstamos (si hay empate, es válido devolver cualquiera de las máximas).
5. Emitir por salida estándar un informe con: total de préstamos, cantidad de socios distintos, listado de categorías con su conteo ordenado de mayor a menor, y la categoría más solicitada.

Requisitos de diseño (convenciones del curso)

- Estructura modular: Sección Declarativa → Funciones (cada una con docstring de precondición y postcondición) → Sección Algorítmica (Prólogo / Resolución / Epílogo).
- Al menos dos funciones con responsabilidad bien delimitada (alta cohesión); comunicación por parámetros y retorno, sin variables globales de estado.
- `for` para recorridos de cantidad conocida; `while` con condición compuesta para iteración indeterminada. Sin `break` ni `continue`.
- Manejo del archivo con `with`. Documentación interna (literate programming).

Qué se evalúa

Las cuatro fases de Pólya: análisis (identificación de entradas/salidas y casos de prueba, incluidos los casos borde: archivo vacío, una sola categoría, empate), diseño (elección de estructuras y descomposición en funciones con su contrato), codificación (corrección y adhesión a las convenciones) y evaluación (qué casos validan la solución).

Rúbrica de evaluación del problema (50 puntos)

Fase (Pólya)	Ptos	Criterio
Análisis	8	Identifica entradas (archivo txt) y salidas (informe); enuncia casos de prueba incluyendo borde (archivo vacío, una categoría, empate).
Diseño	14	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.
Codificación	20	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.
Evaluación	8	Indica con qué casos se valida la solución y verifica el comportamiento en los casos borde.