PREGUNTAS SOBRE PRIORITY QUEUE

a) ¿Qué estructura de datos subyacente se utiliza comúnmente para implementar una cola de prioridad en Python?

La implementación más habitual es un **heap binario** (binary heap) almacenado internamente como una **lista** de Python. Un binary heap cumple la "propiedad de heap": en un *min-heap*, cada nodo es menor o igual que sus hijos (en un *max-heap*, cada nodo es mayor o igual); y se representa muy eficientemente en un array, poniendo el elemento raíz en índice 1 y calculando los hijos de la posición i en 2*i y 2*i+1 .

---

b) ¿Cuál es la diferencia entre una cola FIFO tradicional y una cola de prioridad?

- **Cola FIFO (First-In First-Out):** atiende los elementos en el orden exacto en que llegaron. El primero en entrar es el primero en salir, sin distinguir ninguna otra característica.

- **Cola de prioridad:** cada elemento viene acompañado de una "prioridad" (un número, un nivel, etc.). A la hora de extraer un elemento, no importa tanto el orden de llegada como la prioridad: siempre se saca primero aquel con mayor (o menor) prioridad, independientemente de cuándo fue añadido.

---

c) ¿Qué módulo proporciona Python para trabajar fácilmente con colas de prioridad?

- El módulo **heapq** de la librería estándar ofrece funciones como heappush, heappop, heapify, que implementan un min-heap sobre listas.

- Además, el paquete **queue** incluye la clase **queue.PriorityQueue**, que envuelve internamente a heapq y añade bloqueo/thread-safe para programación concurrente.

---

d) ¿Qué ventajas tiene el uso de una cola de prioridad sobre una lista ordenada manualmente?

1. **Eficiencia:**

    - **Heap**: inserción y extracción en *O(log n)*.

- **Lista ordenada**: para insertar en posición correcta hay que desplazar elementos, costando *O(n)*; extraer el máximo/mínimo es *O(1)* si está al final, pero el coste total de mantenerla ordenada crece linealmente con cada inserción.

2. **Simplicidad de código:** con heapq basta llamar a heappush / heappop; una lista ordenada requiere buscar la posición de inserción y usar list.insert, lo cual es más propenso a errores.

3. **Uso de memoria contigua:** el heap en lista aprovecha bien la caché y evita sobrecarga de nodos enlazados.

---

e) Si dos elementos tienen la misma prioridad, ¿cómo decide la cola cuál atender primero?

- Con heapq, si almacenamos simplemente tuplas (prioridad, dato), Python compara primero la prioridad y, al ser idénticas, compara el dato—lo que puede dar un orden "arbitrario" (o lanzar TypeError si los datos no son comparables).

Para garantizar **estabilidad** (mantener orden de llegada entre iguales), se suele añadir un **contador incremental**, por ejemplo:

```
contador = itertools.count()
heappush(heap, (prioridad, next(contador), dato))
```

- De ese modo, la segunda posición de la tupla rompe empates según el orden de inserción.

---

f) ¿Qué se debe hacer para que los elementos personalizados puedan ser almacenados en una cola de prioridad en Python?

1. **Hacerlos comparables:** implementar al menos el método __lt__ (menor que) en tu clase para que heapq pueda compararlos.

2. **O bien** envolverlos en una tupla (prioridad, objeto) donde la prioridad sea un tipo nativo (int, float…) y garantice comparación, dejando el objeto como segundo elemento para desempaquetar tras la extracción.

---

g) ¿Qué situaciones del mundo real se pueden modelar con colas de prioridad? Mencione al menos dos.

1. **Planificación de CPU** en sistemas operativos: tareas más “urgentes” o con mayor prioridad deben ejecutarse antes.

2. **Simulaciones de eventos discretos** (p. ej. simulación de redes, tráfico): el “evento” con la marca de tiempo más temprana—la prioridad—se procesa primero.

3. (Adicional) **Gestión de impresoras en red**, **gestión de emergencias** (triage), **rutas de mínimo coste** en grafos (algoritmos de Dijkstra/A*).

---

h) En un sistema de atención médica, ¿cómo se puede usar una cola de prioridad para organizar a los pacientes?

Cada paciente recibe un **nivel de urgencia** (por ejemplo, del 1 al 5). Al llegar, se inserta en la cola con su prioridad. Cuando hay un hueco, se extrae siempre el paciente con la mayor urgencia (prioridad numérica más alta o más baja según convención), garantizando que los casos críticos sean atendidos antes que los menos graves.

---

i) ¿Cómo afectaría al rendimiento usar una lista simple en lugar de una estructura especializada como heapq para manejar prioridades?

- **Inserción** en lista ordenada: *O(n)* por desplazamiento de elementos.

- **Extracción** del mejor elemento: si está al final, *O(1)*, pero si está al principio, *O(n)* por desplazamiento.
  En escenarios con muchas inserciones y extracciones frecuentes, el coste *O(n)* empobrece el rendimiento frente al *O(log n)* del heap, especialmente con grandes volúmenes de datos.

---

j) ¿Qué complejidad tiene la inserción y extracción en una cola de prioridad basada en heap?

- **Inserción** (heappush / swim): *O(log n)*
- **Extracción** (heappop / sink): *O(log n)*
- **Consulta** del elemento de mayor prioridad sin extraer (heap[0]): *O(1)*.