Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-2

Publicación: 4 de septiembre de 2025
Actividad evaluada

Actividad 3
Iterables e Iteradores

Entrega

- **Lugar:** Repositorio personal de GitHub — Carpeta: `Actividades/AC3`
- **Fecha máxima de entrega:** 4 de septiembre 17:20
- **Ejecución de actividad**: La Actividad será ejecutada **únicamente** desde la terminal del computador. **Los *paths* relativos utilizados en la Actividad deben ser coherentes con esta instrucción, y no pueden modificarse.**

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Objetivos de la actividad

- Crear y utilizar una estructura nodal.
- Crear y utilizar una lista ligada.
- Crear un iterable y un iterador personalizado definiendo correctamente los métodos `__iter__` y `__next__`.

DCCruzVerde

Preocupado porque no tienes suficiente dinero para salir a celebrar durante el 18 de septiembre, te llega una interesante oferta de trabajo.

Te ha contactado la nueva farmacia del mercado –DCCruzVerde– y te piden a ti –gran estudiante de Programación Avanzada– que los ayudes a implementar un sistema de colas que maneje a los miles de clientes que llegan cada día a esta farmacia.

Feliz, aceptas este desafío y decides utilizar tus conocimientos de Iterables e Iteradores para resolver el problema y ganes las lucas que te faltan para las fiestas patrias.

Flujo de la actividad

El objetivo de esta actividad es que implementes un sistema que permita organizar los clientes que llegan a una farmacia, permitiendo que los clientes preferenciales sean atendidos antes que los normales.

Deberás crear una estructura nodal que permita almacenar y organizar los clientes mediante colas, y crear un sistema que permita manejar las colas de clientes. Además, deberás asegurarte que el sistema de colas se adapte y cumpla el patrón iterable/iterador.

Te entregaremos el archivo `main.py` el cual tiene estructuras básicas que deberás completar con el código adecuado. Debes asegurarte de entregar, como mínimo, los archivos que tengan el *tag* de Entregar en la siguiente sección. Los demás archivos no es necesario subir, pero tampoco se penalizará si se suben al repositorio personal.

Archivos y entidades

- Entregar Modificar `main.py`: Archivo principal a ejecutar, se encarga de llamar a todas las clases a implementar. Además, contiene las clases `NodoCliente`, `SistemaColas` y `IteradorSistemaColas`.

  La clase `NodoCliente` representa un nodo que almacena la información de un cliente. Esta clase se utiliza para crear cadenas de nodos, similares a una lista ligada. Posee el siguiente atributo:

  - `self.identificador` Identificador único que permite identificar el nodo a partir de un entero. Este se basa en el atributo de clase `identificador`.
  - `self.preferencial` Booleando que indica si el cliente es preferencial (`True`) o normal (`False`).
  - `self.siguiente` Referencia al nodo siguiente. Inicialmente este atributo parte con el valor `None` ya que el nodo no presenta un sucesor.

  La clase `SistemaColas` se encarga de almacenar sistema de colas de la farmacia. Posee el siguiente atributo:

  - `self.cola_preferencial` Lista ligada correspondiente a la cola preferencial. Inicialmente parte con el valor `None`.
  - `self.cola_normal` Lista ligada correspondiente a la cola normal. Inicialmente parte con el valor `None`.

  La clase `IteradorSistemaColas` se encarga recorrer los elementos del `SistemaColas`. Posee el siguiente atributo:

  - `self.cola_preferencial` Copia de la cola preferencial del `SistemaColas` que está siendo recorrida por el iterador.
  - `self.cola_normal` Copia de la cola normal del `SistemaColas` que está siendo recorrida por el iterador.
  - `self.contador_preferencial` Contador que cuenta cuántos clientes preferenciales han sido atendidos de corrido. Inicialmente su valor es `0`.

  Estas tres clases -`NodoCliente`, `SistemaColas`, `IteradorSistemaColas`- posee distintos métodos que deberán ser implementados o modificados. Esto será explicado en las siguientes partes del enunciado.

Parte I. NodoCliente

La primera parte de tu trabajo consistirá en crear la estructura que permitirá guardar la información de los distintos clientes de la farmacia y organizarlos mediante la creación de cadenas de nodos.

Para lograrlo, deberás completar los siguientes métodos de la clase `NodoCliente`:

- `def agregar_nodo(self, nuevo_nodo: NodoCliente) -> None:`

  Agrega la instancia de `NodoCliente` al final de la cadena de nodos.

- `def __str__(self) -> str:`

  Retorna un *string* que representa a la instancia de `NodoCliente` y la cadena de nodos que le sucede.

  El texto a retornar debe cumplir con el siguiente formato:

```
'C({identificador}) -> {siguiente_nodo}'
```

  donde `identificador` corresponde al identificador del nodo, y `siguiente_nodo`, al *string* que representa a su sucesor.

  Por ejemplo, una cadena de 4 nodos se verá de la siguiente manera:

```
C(0) -> C(1) -> C(2) -> C(3) -> None
```

- `def __len__(self) -> int:`

  Retorna un entero que indica la cantidad de nodos que contiene la cadena.

Parte II. SistemaColas

Ahora que hemos logrado implementar los nodos que almacenan y organizan la información, debemos preparar el sistema de colas que maneja los clientes preferenciales y normales.

Para lograrlo, deberás completar los siguientes métodos de la clase `SistemaColas`:

- `def agregar_persona(self, preferencial: bool) -> None:`

  Recibe un booleano que indica si ha llegado un cliente preferencial. Crea una instancia de `NodoCliente` y lo agrega al final de la cola que le corresponde.

- `def __len__(self) -> int:`

  Retorna un entero que indica la cantidad total de clientes en el sistema de colas.

  Por ejemplo, el sistema que se presenta a continuación retornaría 7:

```
Preferencial: C(0) -> C(1) -> C(2) -> C(3) -> C(4) -> None
Normal:       C(5) -> C(6) -> None
```

Parte III. Iterable e iterador de SistemaColas

Finalmente, el último paso que queda para hacer que el sistema de cola funcione es implementar el patrón iterable/iterador.

Para lograrlo, deberás completar el siguiente método de la clase `SistemaColas`:

- `def __iter__(self) -> IteradorSistemaColas:`

  Retorna una instancia de `IteradorSistemaColas`. Debes asegurarte que el iterador no modifique la información de `SistemaColas`, para esto investiga las funciones `copy` y `deepcopy` de la módulo `copy` y utiliza el más adecuado.

Además, deberás completar los siguientes métodos de la clase `IteradorSistemaColas`:

- `def __iter__(self) -> Self:`

  Retorna la instancia misma del iterador (`self`).

- `def __next__(self) -> NodoCliente:`

  Retorna la siguiente instancia de `NodoCliente` que debe ser atendido.

  El sistema de colas de esta farmacia prioriza la elección de clientes preferenciales por sobre los normales, salvo que se hayan atendido 3 clientes preferenciales de forma consecutiva o que no hayan clientes preferenciales.

  Un vez que no queden clientes en la cola preferencial y normal, se levanta la excepción correspondiente a esta situación.

Notas

- **No puedes hacer *import* de otras librerías o módulos externos a las entregadas en el archivo.**
- Recuerda que la ubicación de tu entrega es en **tu repositorio de Git**. En la rama (*branch*) por defecto del repositorio: `main`.
- Se recomienda completar la actividad en el orden del enunciado.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- Si aparece un error inesperado, ¡léelo y revisa el código del *test*!

Ejecución de *tests*

En esta actividad se provee de varios archivos `.py` los cuáles contiene diferentes *tests* que ayudan a validar el desarrollo de la actividad. Para ejecutar estos *tests*, **primero debes posicionar tu terminal/consola en la carpeta de la actividad (`Actividades/AC3`)**. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests* de la actividad:

- `python3 -m unittest discover tests_publicos -v`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo si escribes lo siguiente en la terminal/consola:

- `python3 -m unittest -v tests_publicos.test_nodo_cliente`
  Para ejecutar solo el subconjunto de *tests* relacionado a la Parte II.
- `python3 -m unittest -v tests_publicos.test_sistema_colas`
  Para ejecutar solo el subconjunto de *tests* relacionado a la Parte II.
- `python3 -m unittest -v tests_publicos.test_iterable_iterador`
  Para ejecutar solo el subconjunto de *tests* relacionado a la Parte III.
- `python3 -m unittest -v tests_publicos.test_programa_completo`
  Para ejecutar solo el subconjunto de *tests* que revisa la correcta implementación de todo lo anterior.

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.12`.