



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 Programación Avanzada (2023-1)

Tarea 0

Entrega

- Tarea y README.md
 - **Fecha y hora:** martes 28 de marzo de 2023, 20:00
 - **Lugar:** Repositorio personal de GitHub — Carpeta: Tareas/T0/

Objetivos

- Desarrollar algoritmos para la resolución de problemas complejos.
- Aplicar competencias asimiladas en *Introducción a la Programación* para el desarrollo de una solución a un problema.
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.
- Trabajar con archivos de texto para leer, escribir y procesar datos.
- Escribir código utilizando paquetes externos (*i.e.* código no escrito por el estudiante), como por ejemplo, módulos que pertenecen a la biblioteca estándar de Python.
- Familiarizarse con el proceso de entrega de tareas y uso de buenas prácticas de programación.

Índice

1. <i>DCCeldas</i>	3
2. Reglas del sistema de defensa	3
3. Flujo del programa	5
3.1. Menú de Inicio	5
3.2. Menú de Acciones	5
4. Funciones	6
5. Archivos	7
5.1. tablero.py	8
5.2. functions.py	8
6. <i>Buenas prácticas</i>	8
7. README	9
8. <i>Bonus</i>	9
8.1. Funciones atómicas (2 décimas)	10
8.2. Regla 5 (4 décimas)	10
9. Descuentos	10
10..gitignore	11
11.Importante: Corrección de la tarea	11
12.Restricciones y alcances	11

1. *DCCeldas*

Hace mucho tiempo, en el reino de las tortugas gobernado por la gran **Lily416** (mejor conocida como profe Dani) existía paz y armonía. Sin embargo, todo comenzó cuando el malvado **Dr. Pinto** atacó, generando terror en el reino. Este plan de ataque consiste en instalar bombas que explotarán y destruirán el castillo de la gobernadora.

Ante esta situación, las tortugas decidieron ubicarse en distintas posiciones llamadas *DCCeldas*, para defender el castillo con sus resistentes caparazones. Lamentablemente, las tortugas no son lo suficientemente inteligentes para tener una buena estrategia de defensa. Es por esto que **Lily416** te encarga a ti, como estudiante de *Programación Avanzada*, que diseñes un plan de defensa y contrarrestar el poderoso ataque del malvado **Dr. Pinto**.



Figura 1: Logo de *DCCeldas*

2. Reglas del sistema de defensa

Para poder diseñar una estrategia de defensa, deberás simular las secciones del castillo de **Lily416**. Este castillo tiene forma de un tablero cuadrado, con n filas y n columnas. Cada una de las celdas puede estar en tres posibles estados:

- "-": Representa una **celda vacía**.
- "T": Representa una **tortuga** en dicha posición.
- **int**: Representa una **bomba**. El número específico indica la cantidad de celdas que la bomba debe ser capaz de “observar” contándose a sí misma. A continuación se darán más detalles.

Por temas de estandarización, el origen del tablero estará ubicada en la esquina superior izquierda, siendo esta la coordenada $(0, 0)$. En el lado opuesto, la esquina inferior derecha será la última celda ubicada en la coordenada $(n - 1, n - 1)$. A continuación se muestra un ejemplo del tablero con $n = 5$:







	0	1	2	3	4
0	2				
1					
2		5		3	
3				6	
4	3				5

Figura 2: Tablero de *DCCeldas* con $n = 5$

Tal como se muestra en el ejemplo, en la coordenada (0, 0) está ubicada una celda con el número 2. En la coordenada (1, 3) está la celda con una "T", la coordenada (3, 4) está la celda vacía ("-") y en la coordenada (4, 4) está el número 5.

La forma de rellenar las celdas con tortugas tiene una variedad de reglas que logran que puedan existir una o más soluciones. Las reglas que se deben cumplir son las siguientes:

Regla 1 Cada número del tablero representa la cantidad de celdas que se ven afectadas por la explosión de una bomba. Una celda es afectada por la explosión de una bomba si dicha bomba está posicionada en la misma fila o columna de la celda, y además no hay tortugas que se interpongan entre ellas.

Por ejemplo, si revisamos el caso de la [Figura 2](#), podemos notar que en la coordenada (0, 0), el número 2 del tablero indica que la bomba en esa celda sólo tiene alcance de dos celdas desde donde esta ubicada, incluyendo en la que está instalada.

Regla 2 El mínimo valor que puede tener una bomba en *DCCeldas* será 2 y el valor máximo dependerá del tamaño del tablero, dado por $2n - 1$, donde n es el alto (o ancho) del tablero. Ejemplo:

- Si el tablero es de 2x2, los únicos valores posibles de bombas son 2 o 3.
- Si el tablero es de 4x4, los únicos valores posibles de bombas son números del 2, 3, 4, 5, 6 y 7.
- Si el tablero es de 100x100, los únicos valores posibles de bombas son números enteros entre el 2 y 199.

Regla 3 Las celdas con números no pueden estar tapadas, es decir, no puede haber una tortuga y una bomba en la misma celda.

Regla 4 Dos celdas tapadas, es decir, que tienen una tortuga, no pueden estar juntas en forma horizontal o vertical, solo en diagonal.

Regla 5 (Opcional: *Bonus*) **Todas** las celdas que **no contengan tortugas** (ya sea con una bomba o que se encuentre vacía) deben estar conectadas horizontal o verticalmente, es decir, debe ser posible ir de cualquiera de estas celdas a otra dentro del tablero. En otras palabras, no deben existir "islas" de estas celdas.

A continuación se muestra un ejemplo del tablero del castillo original, y tres posibles soluciones.

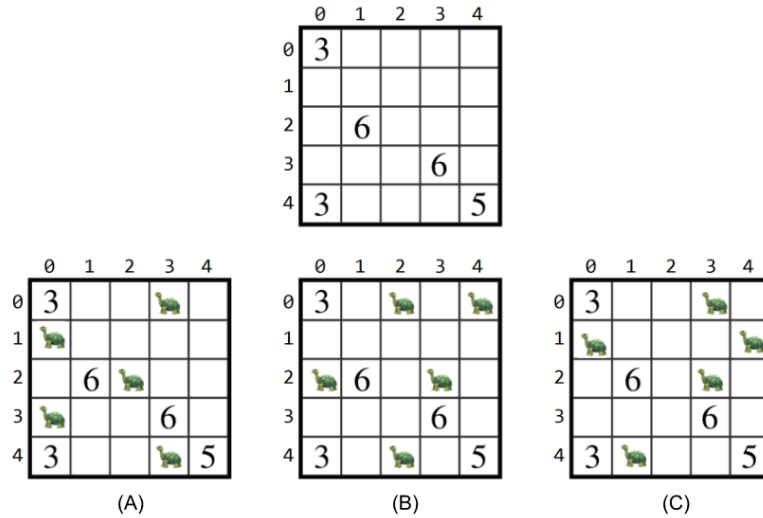


Figura 3: Tablero de *DCCeldas* y tres posibles soluciones.

En la figura anterior se puede observar que existen tres soluciones posibles para un mismo tablero original. Para los casos de (A) y (B) se cumplen las 5 reglas, mientras que en (C) se cumple solo las primeras 4 reglas, ya que existe una celda blanca (0, 4) que **no** se encuentra conectada a las demás celdas blancas.

3. Flujo del programa

Tu objetivo es escribir un programa que permita generar la mejor estrategia de defensa ante el malvado **Dr. Pinto**. La ejecución e interacción del programa será mediante consola, por lo que en esta deberán aparecer todas las instrucciones para el usuario.

Al ejecutar tu programa lo primero que se deberá mostrar en consola es el [Menú de Inicio](#), que le preguntará al usuario cuál es el archivo que desea utilizar como estrategia de defensa. En el caso de elegir uno de los archivos disponibles, se debe cargar dicho archivo¹ y abrir el [Menú de Acciones](#) con sus opciones correspondientes.

3.1. Menú de Inicio

En esta primera parte del programa se le debe preguntar al usuario cuál es el archivo que desea abrir, para verificar si efectivamente el archivo existe o no. En el caso de que la respuesta del usuario sea válida, se debe abrir automáticamente el [Menú de Acciones](#). Por el contrario, en caso de que el archivo elegido no sea valido se debe avisar al usuario y posteriormente cerrar el programa. Para verificar la existencia de un archivo, se recomienda investigar la librería [os](#).

3.2. Menú de Acciones

Este menú permitirá al usuario revisar si el tablero escogido cumple con las funciones requeridas para obtener la mejor estrategia de defensa. Este debe incluir las opciones de **mostrar tablero**, **validar bombas y tortugas**, **validar solución**, **solucionar el tablero** y **salir del programa**. Este menú debe ser a prueba de errores ante todo *input* ingresado por el usuario. A continuación, se detalla la función que debe cumplir cada una de las acciones que podrá realizar el usuario sobre el tablero escogido.

1. **Mostrar tablero:** imprime el tablero en la consola.

¹Más información en la sección de [Archivos](#)

2. **Validar bombas y tortugas:** verifica que el tablero cumpla con las [Regla 2](#) y [4](#). Se debe mostrar en consola si el tablero es efectivamente válido o no.
3. **Validar solución:** verifica que el tablero se encuentre resuelto, vale decir, que se cumplan las [Reglas 1, 2, 3 y 4](#). Se debe mostrar en consola si el tablero es efectivamente válido o no.
4. **Solucionar tablero:** completa el tablero asegurando que se cumplan las [Reglas 1, 2, 3 y 4](#). Luego, guarda el tablero en un archivo con el nombre original de este junto con el sufijo **"sol"**. Por ejemplo, si el tablero era el **"Archivo1.txt"**, el nuevo tablero se llamará **"Archivo1_sol.txt"**. Finalmente, se debe notificar por consola si el tablero se pudo solucionar o no. En caso de tener solución, se deberá mostrar el tablero resuelto en consola.
5. **Salir del programa:** termina la ejecución del programa.

A continuación se entrega un ejemplo de cómo se podría ver este menú.

<pre>*** Menú de Inicio ***</pre> <p>Indique el nombre del archivo que desea abrir:</p> <p>(a) Ejemplo de Menú Inicial</p>	<pre>*** Menú de Acciones ***</pre> <pre>[1] Mostrar tablero [2] Validar tablero [3] Revisar solución [4] Solucionar tablero [5] Salir del programa</pre> <p>Indique su opción (1, 2, 3, 4 o 0):</p> <p>(b) Ejemplo de Menú de Acciones</p>
--	---

4. Funciones

En orden de ~~facilitar la corrección a los ayudantes~~ lograr un código más ordenado, se pedirá que implementes como **mínimo**² una serie de **funciones** las cuales deberán ser **llamadas y utilizadas correctamente**³ en el código. Estas funciones estarán definidas en un módulo ([functions.py](#)) que les entregaremos para esta evaluación. Solo debes completar estas funciones, es decir, no se puede cambiar su nombre, alterar los argumentos recibidos o cambiar lo que retorne la función. A continuación se detallan las funciones a implementar:

- **def cargar_tablero(nombre_archivo: str) -> list:**

Esta función recibe como argumento un **str** con el nombre del archivo y retorna una lista de listas con la información del tablero. El formato de estos archivos será indicado en la sección de [Archivos](#).

- **def guardar_tablero(nombre_archivo: str, tablero: list) -> None:**

Esta función recibe como argumentos el nombre del archivo con el cual se va a guardar y el tablero en forma de lista de listas. El formato de guardado debe ser el mismo que el indicado en la sección de [Archivos](#). No debe retornar nada.

- **def verificar_valor_bombas(tablero: list) -> int:**

Esta función recibe como argumento el tablero en forma de lista de listas y se encarga de verificar que la [Regla 2](#) se cumpla, es decir, que los valores de las bombas sean válidos. Retornará la cantidad de bombas inválidas en el tablero.

²Puedes crear más funciones si lo consideras pertinente

³Si las funciones son definidas, pero no están siendo llamadas o utilizadas correctamente para realizar la acción asociada, no se entregará puntaje

- `def verificar_alcance_bomba(tablero: list, coordenada: tuple) -> int:`

Esta función recibe como argumentos el tablero en forma de lista de listas y las coordenadas como una tupla (x, y) , donde x corresponde al número de la fila e y como al número de la columna en forma de `int`. Se debe verificar que la coordenada contenga una bomba y calcular el alcance de dicha bomba, es decir, cuántas celdas blancas, incluida ella misma, alcanza con la explosión antes de colisionar con un borde o una tortuga y retornar ese alcance calculado como `int`. En caso que dicha celda no sea una bomba, esta función debe retornar 0.

- `def verificar_tortugas(tablero: list) -> int:`

Verifica que no hay dos o más tortugas contiguas vertical u horizontalmente. Retornará un `int` con la cantidad de Tortugas que no cumplan con lo anterior, es decir, la [Regla 4](#).

- `def solucionar_tablero(tablero: list) -> list:`

Completa el tablero recibido con las tortugas necesarias para asegurar que se cumplan las [Reglas 1, 2, 3 y 4](#) de *DCCeldas*. Finalmente retorna el tablero solucionado como una lista de listas. En caso que el tablero no tenga solución, esta función debe retornar `None`.

Finalmente, para la impresión del tablero, deberás importar el módulo `tablero.py` y utilizar la función `imprimir_tablero(tablero: list, utf8=True)`. Esta función recibe una lista de listas, donde cada lista corresponde a una fila del tablero.

5. Archivos

Para poder entender la arquitectura del castillo de **Lily416**, se te facilitarán distintos archivos que contendrán información sobre las dimensiones y el alcance de las bombas instaladas por el **Dr. Pinto**. Dado que el castillo está encantado, esta puede tomar distintas dimensiones acorde al archivo seleccionado.

Dentro de la carpeta **Archivos** podrás encontrar información de los posibles tableros del castillo. El contenido de cada archivo estará dado por una línea de texto, el cual tendrá el siguiente formato:

- El primer número N antes de la primera coma (",") representa un `int` con la dimensión del largo del tablero cuadrado.
- Luego vendrán $N \times N$ elementos separados por una coma (","), donde por cada N elementos se interpretará como una fila del tablero.
- Cada uno de estos elementos puede ser: un `int` que representa un número que indica la cantidad de celdas que puede alcanzar una bomba, una `"T"` que representa una tortuga, o bien un `"-"` que representa una celda vacía.

Un ejemplo de esto una línea de archivo y su representación en el tablero sería de la siguiente forma:

1

5,2,-,T,-,-,T,-,-,T,-,-,5,T,3,-,-,-,6,-,3,T,-,T,5







2				
				
	5		3	
			6	
3				5

Figura 5: Representación del archivo de texto en forma de tablero

5.1. tablero.py

Para facilitar tu trabajo, los esclavos de ~~Lily416~~ **maestros ayudantes** te harán entrega de módulo `tablero.py` que contiene la función `imprimir_tablero(tablero: list, utf8=True)` encargado de imprimir el tablero en pantalla de una forma ordenada. El parámetro `tablero` corresponde a una lista de listas con la información del tablero, mientras que `utf8` es un booleano (es decir **True** o **False**) que indica si el tablero se imprimirá con caracteres UTF-8 o no, ya que dependiendo de la fuente que utiliza tu consola, el tablero puede presentar problemas para mostrar los caracteres UTF-8 o deformarse. El valor por defecto de esta variable es **True**, en dicho caso el tablero imprimirá los caracteres UTF-8, mientras que si es **False**, estos no se imprimirán. Por lo tanto, si es que tu tablero no se está mostrando correctamente te recomendamos dejar el valor de este argumento como **False**. Recuerda notificar en tu [README](#) en caso de cambiar este valor, para que el ayudante corrector no tenga problemas.

Será tú deber importar **correctamente** la función `tablero.py` y hacer uso de ella para imprimir el tablero en consola. A continuación se muestra un ejemplo del tablero y su representación en consola al llamar la función:

```

1 tablero = [
2     ['2', '-', 'T', '-', '-'],
3     ['T', '-', '-', 'T', '-'],
4     ['-', '5', 'T', '3', '-'],
5     ['-', '-', '-', '6', '-'],
6     ['3', 'T', '-', 'T', '5']
7 ]

```

(a) Ejemplo de lista de listas del tablero

	0	1	2	3	4
0	2	-	T	-	-
1	T	-	-	T	-
2	-	5	T	3	-
3	-	-	-	6	-
4	3	T	-	T	5

(b) Tablero en consola con `utf8=True`

5.2. functions.py

Este módulo contiene la base de las funciones solicitadas en la sección [Funciones](#). Este archivo será revisado para asignar el puntaje correspondiente por función. Es tu responsabilidad asegurarte de completarlo con las indicaciones dadas en el enunciado y que funcione correctamente. En caso de definir las funciones en otro archivo, **no se entregará el puntaje correspondiente**.

6. Buenas prácticas

Se espera que durante todas las tareas del curso, se empleen buenas prácticas de programación. Esta sección detalla dos aspectos que deben considerar a la hora de escribir sus programas, que buscan mejorar la forma en que lo hacen.

■ PEP8:

PEP8 es una guía de estilo que se utiliza para programar en Python. Es una serie de reglas de redacción al momento de escribir código en el lenguaje y su utilidad es que permite estandarizar la forma en que se escribe el programa para sea más legible⁴. En este curso te pediremos seguir un pequeño apartado de estas reglas, el cual puede ser encontrado en la [guía de estilo](#).

■ Modularización:

Al escribir un programa complejo y largo, se recomienda organizar en múltiples módulos de poca extensión. Se obtendrá puntaje si ningún archivo de tu proyecto contiene más de 400 líneas de código.

Normalmente, estos dos aspectos son considerados como descuentos. A manera de excepción, para esta tarea serán parte del puntaje de la tarea, buscando que los apliques y premiando su correcto uso. Ten en cuenta que en las siguientes tareas, funcionarán como cualquier otro descuento de la sección [Descuentos](#).

7. README

Para todas las tareas de este semestre deberás redactar un archivo `README.md`, un archivo de texto escrito en Markdown, que tiene por objetivo explicar su tarea y facilitar su corrección para el ayudante. Markdown es un lenguaje de marcado (como \LaTeX o HTML) que permite crear un texto organizado y simple de leer. Pueden encontrar un pequeño tutorial del lenguaje en este [link](#).

Un buen `README.md` debe **facilitar la corrección de la tarea**. Una forma de lograr esto es explicar de forma breve y directa el **idioma** en qué programaste (puedes usar inglés o español), incluir las referencias, e indicar **qué cosas fueron implementadas, se encuentran incompleta, presentan errores o no fueron implementaron**, usualmente **siguiendo la pauta de evaluación**. Esto permite que el ayudante dedique más tiempo a revisar las partes de tu tarea que efectivamente lograste implementar, lo cual permite entregar un *feedback* más certero. Para facilitar la escritura del README, se entregará una [plantilla](#) (*template*) a rellenar con la información adecuada.

Se **recomienda fuertemente** ir completando este archivo a medida que se desarrolla la tarea. De este modo, se asegura la creación de un buen y completo `README.md`. Aprovecha este documento para dejar un registro de las funcionalidades que has completado y las que se encuentran incompletas o presentan un error, así tendrás más claridad de qué puntos de la tareas has completado y cuáles presentan problemas.

8. Bonus

En esta tarea habrá una serie de *bonus* que podrás obtener. Cabe recalcar que necesitas cumplir los siguientes requerimientos para poder obtener *bonus*:

1. La nota en tu tarea (sin bonus) debe ser **igual o superior a 4.0**⁵.
2. El bonus debe estar implementado **en su totalidad**, es decir, **no se dará puntaje intermedio**.

Finalmente, la cantidad máxima de décimas de *bonus* que se podrá obtener serán 6 décimas. Deberás indicar en tu README si implementaste alguno de los bonus, y cuáles fueron implementados.

⁴Símil a como la ortografía nos ayuda a estandarizar la forma es que las palabras se escriben

⁵Esta nota es sin considerar posibles descuentos

8.1. Funciones atómicas (2 décimas)

En orden de mantener un código más ordenado, se pide que cada **función o método** definido dentro del código contenga un **máximo de 15 líneas** de código. Esto incluye **todas** las funciones solicitadas en la sección [Funciones](#), además de cualquier otra función o método creado por el estudiante.

8.2. Regla 5 (4 décimas)

En orden de tener *DCCeldas* programado de forma completa, se pide mejorar las funciones necesarias para considerar la [Regla 5](#) al momento de validar y solucionar un tablero. En particular, para optar a este *bonus* se deben cumplir los siguientes 3 requisitos:

- La opción 2 del menú de acciones ([Subsección 3.2 - Validar tablero](#)) deberá incluir en su validación que la [Regla 5](#) también se cumpla para retornar **True**.
- La opción 4 del menú de acciones ([Subsección 3.2 - Solucionar tablero](#)) deberá asegurar que la solución respete todas las reglas, incluyendo la 5.
- Deberás implementar y utilizar la siguiente función: `def verificar_islas(tablero: list) -> bool:` Deberá retornar un valor booleano (**True** o **False**) que indique si es que existen celdas blancas aisladas.

En caso de optar al *bonus* de Funciones atómicas. Esta función también deberá estar definida en un máximo de **15 líneas**.

9. Descuentos

En todas las tareas de este ramo habrá una serie de descuentos que se realizarán para tareas que no cumplan ciertas restricciones. Estas restricciones están relacionadas con malas prácticas en programación, es decir, formas de programar que hacen que tu código sea poco legible y poco escalable (difícil de extender con más funcionalidades). Los descuentos tienen por objetivo que te vuelvas consciente de estas prácticas e intentes activamente evitarlas, lo cual a la larga te facilitará la realización de las tareas en éste y próximos ramos donde tengas que programar. Los descuentos que se aplicarán en esta tarea serán los siguientes:

- **README:** (5 décimas)
 - Se descontará 1 décima si no se indica(n) los archivos principales que son necesarios para ejecutar la tarea o su ubicación dentro de su carpeta.
 - Se descontará hasta 4 décimas si se sube el README, pero este se encuentra incompleto, sin especificar los aspectos implementados, incompletos y no implementados o sin mencionar posibles errores dentro del código. Para más información, pueden revisar la sección de [README](#).
- **Formato de entrega:** (hasta 5 décimas)

Se descontarán hasta cinco décimas si es que no se siguen las reglas básicas de la entrega de una tarea, como son: el uso de groserías en su redacción, archivos sin nombres aclarativos, no seguir restricciones del enunciado, entre otros⁶. Esto se debe a que en próximos ramos (o en un futuro trabajo) no se tiene tolerancia respecto a este tipo de errores.
- **Cambio de líneas:** (hasta 5 décimas)

Se permite cambiar **hasta 20 líneas de código** por tarea, ya sea para corregir un error o mejorar una funcionalidad. El cambio de líneas debe pedirse oportunamente **en el README o mediante el**

⁶Uno de los puntos a revisar es el uso de *paths* relativos, para más información revisar el siguiente [material](#)

formulario de corrección, si no se pide en alguna de estas instancias, no se cambiarán líneas en el código entregado. Este descuento puede aplicarse si se requieren cambios en el código después de la entrega (incluyendo las entregas atrasadas). Dependiendo de la cantidad de líneas cambiadas se descontará entre una y cinco décimas.

- **Built-in prohibidos:** (entre 1 a 5 décimas)

En cada tarea se prohibirán algunas funcionalidades que Python ofrece y se descontarán entre una y cinco décimas si se utilizan, dependiendo del caso. Para cada tarea se creará una *issue* donde se especificará qué funcionalidades estarán prohibidas. Es tu responsabilidad leerla.

En la [guía de descuentos](#) se puede encontrar un desglose más específico y detallado de los descuentos.

10. .gitignore

Para esta tarea **deberás utilizar un .gitignore** para ignorar los archivos indicados, este deberá estar dentro de tu carpeta Tareas/T0/. Puedes encontrar un ejemplo de .gitignore en el siguiente [link](#).

Acá se deben indicar los archivos a ignorar.

Se espera que no se suban archivos autogenerados por las interfaces de desarrollo o los entornos virtuales de Python, como por ejemplo: la carpeta `__pycache__`.

Para este punto es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos no **deben** subirse al repositorio debido al archivo `.gitignore` y no debido a otros medios.

11. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del programa será el pilar de la corrección, es decir, **sólo se corrigen tareas que se puedan ejecutar**. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y *push* en sus repositorios.

Cuando se publique la distribución de puntajes, se señalará con color **amarillo** cada ítem que será evaluado a nivel de código, todo aquel que no esté pintado de amarillo significa que será evaluado si y sólo si se puede probar con la ejecución de su tarea.

En tu archivo `README.md` deberás señalar el archivo y la línea donde se encuentran definidas las funciones o clases relacionados a esos ítems.

Finalmente, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante Coordinador de Bienestar o al ayudante de Bienestar de tu sección, puedes hacerlo escribiéndoles a sus respectivos [correos](#).

12. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el [Código de honor de Ingeniería](#).
- Tu programa debe ser desarrollado en Python 3.10.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py`.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del [foro](#) si es que es posible utilizar alguna librería en particular.

- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. El no incluir este archivo o bien se encuentra vacío conllevaría un **descuento** en tu nota.
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0).