



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

IIC2233 Programación Avanzada (2020-2)

Tarea 00


Entrega

- Tarea
 - **Fecha y hora:** sábado 29 de agosto de 2020, 20:00
 - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T00/`
- `README.md`
 - **Fecha y hora:** lunes 31 de agosto de 2020, 20:00
 - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T00/`

Objetivos

- Aplicar competencias asimiladas en *Introducción a la Programación* para el desarrollo de una solución a un problema.
- Familiarizarse con el proceso de entrega de tareas y uso de buenas prácticas de programación.
- 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 librería estándar de Python.

Índice

1. <i>DCCombateNaval</i>		3
2. Flujo del programa		3
3. Menús		3
3.1. Menú de Inicio		4
3.2. Menú de Juego		4
4. Reglas		5
5. Elementos		5
5.1. Tablero		5
5.1.1. Mapa del Jugador		6
5.1.2. Mapa del Oponente		6
5.2. Barcos		6
5.3. Bombas		7
5.3.1. Bombas Regulares		7
5.3.2. Bombas Especiales		7
5.3.3. Bomba Cruz		7
5.3.4. Bomba X		7
5.3.5. Bomba Diamante		8
6. Partida		8
6.1. Crear		8
6.2. Oponente		9
7. Puntajes		9
8. Archivos Entregados		10
8.1. tablero.py		10
8.2. parametros.py		11
8.3. puntajes.txt		11
9. Buenas prácticas		11
10.README		12
11.Descuentos		12
12..gitignore		13
13.Importante: Corrección de la tarea		14
14.Restricciones y alcances		14

1. *DCCombateNaval*

Luego de unas aburridas vacaciones buscando qué comprar en *AliExpress*, te comienzas a preocupar debido a la enorme tardanza en la llegada de tus compras. Luego de un poco de investigación te enteras de que tus pedidos vienen en camino en barco desde China, y tienen fecha estimada de llegada el 2023-2. Ya sin ganas de esperar más, decides hacerte cargo del problema personalmente, destruyendo la flota de barcos de *AliExpress* para que nunca nadie más deba pasar por tal suplicio.

Para asegurarte de que tu plan no pueda fracasar, y aprovechando que ahora estás cursando IIC2233, te la juegas por crear *DCCombateNaval* para prever el resultado de tu ataque.



Figura 1: Logo de *DCCombateNaval*

2. Flujo del programa

DCCombateNaval consiste en un programa que simula una batalla naval por turnos entre el jugador y la computadora. La ejecución debe ser por **consola**, por lo que debes asegurarte que cualquier instrucción o mensaje destinado al jugador se visualice en ella.

Al ejecutar el programa debe mostrarse el **Menú de Inicio**, en donde se podrá **iniciar una nueva partida** o ver los **puntajes de partidas previas**. En caso de iniciar una partida nueva se mostrará el **Menú de Juego**, en donde se podrá visualizar el tablero del jugador y el oponente, y realizar diferentes acciones como **salir** del programa o realizar una **jugada**.

Si se selecciona el Menú de Juego, el jugador puede optar por la opción **Rendirse**, que implica perder automáticamente la partida y volver al **Menú de Inicio**, o bien optar por **Disparar** una bomba regular o especial, que implica ingresar una coordenada, disparar contra el oponente y descubrir qué oculta esa ubicación del mapa. Una vez el jugador ingrese una coordenada válida, el programa indicará si ahí se ubica un barco o no siguiendo la simbología indicada en la sección **Tablero**.

Tras el disparo del jugador, es el turno del oponente, el cual tendrá durante su turno un comportamiento explicado en la sección **Oponente**. Ganará quien dispare primero a todas las casillas que alberguen barcos del oponente. Tras ganar o perder, el programa indica el resultado junto al puntaje final obtenido en la partida, el cual también se deberá almacenar en un archivo. Luego, el programa regresa al jugador al **Menú de Inicio** para poder empezar una nueva partida o salir del programa.

3. Menús

Para hacer más amigable la experiencia a los usuarios de *DCCombateNaval*, deberás implementar distintos **menús** que permitan realizar la interacción por consola. Cada menú muestra opciones disponibles al usuario, cuya decisión debe ser **recibida** y **procesada** de forma correcta.

Todos los menús deben ser **a prueba de errores**, es decir, tu programa no debe caerse al ingresar una opción inválida y debe responder de forma acorde. El formato en que decidas mostrarlos queda a tu criterio, pero debes incluir las opciones de volver al menú anterior y salir del programa cuando corresponda.

Puedes crear distintos submenús si lo consideras necesario, pero como mínimo deberás crear los siguientes:

3.1. Menú de Inicio

Es el menú que primero se mostrará al jugador al iniciar el programa. Debe preguntarle al jugador si desea **Iniciar una partida**, **Ver ranking de puntajes** de partidas anteriores, o **Salir del programa**.

Si se elige una partida nueva, el programa debe preguntar por el **Apodo** del jugador, el cual debe cumplir con ciertos requisitos: una extensión mínima de **cinco** caracteres, los que deben ser **alfanuméricos**¹ (números o letras), pudiendo incluir tanto **mayúsculas** como **minúsculas** a la vez. En caso que el **Apodo** no sea válido, es necesario que se indique mediante un mensaje y se den las opciones de volver a ingresarlo nuevamente o volver al **Menú de Inicio**. Una vez seleccionado un apodo válido, se le debe pedir al jugador el **tamaño del tablero**, pidiendo de forma explícita el número de **filas** y **columnas** de este, siguiendo las restricciones de la sección **Crear**. La forma de pedir el tamaño del tablero quedará a tu criterio.

Si se elige la opción de **ver ranking de puntajes**, deberás mostrar en consola los **apodos** con su respectivo **puntaje** de las **5** mejores puntuaciones ordenadas de **mayor a menor**². En caso de que existan menos de 5 puntajes registrados, deberás mostrar todos los apodos de los jugadores y sus puntajes bajo el mismo criterio de ordenación. Por último, en caso de que no hayan puntajes registrados, deberás indicarlo a través de la consola y volver al **Menú de Inicio**.

```
***** Menu de Inicio *****
```

```
Selecciona una opción:
```

```
[0] Iniciar una Partida
[1] Ver Ranking de Puntajes
[2] Salir
```

```
Indique su opción (0, 1 o 2): (input del usuario)
```

(a) Ejemplo de Menú de inicio

```
***** Ranking de Puntajes *****
```

```
1) misi99: 120 Pts
2) sushi: 36 Pts
3) Opitu0: 0 Pts
```

```
[0] Volver
```

```
Indique su opción (0): (input del usuario)
```

(b) Ejemplo de Ranking de Puntajes

3.2. Menú de Juego

Este menú se despliega luego de ingresar un **apodo** válido y haber seleccionado el **tamaño del mapa**.

Debes dar la opción de **Salir** del juego finalizando el flujo del programa, **Rendirse** para terminar el juego y volver al **Menú de Inicio**, o **Lanzar una bomba**. Al elegir esta última opción el jugador debe seleccionar el tipo de **Bombas** a usar y las coordenadas a donde se quiere lanzar. Estas coordenadas son del tipo **LETRA**, **NÚMERO**, en donde **LETRA** hace referencia a la columna y **NÚMERO** a la fila donde se desea apuntar. Los únicos disparos válidos serán a celdas que aún no han sido descubiertas y se encuentran dentro del mapa. Además, solo se puede disparar a una coordenada válida por turno, a menos que se impacte un barco enemigo, caso en que el jugador actual tendrá un nuevo disparo. Las coordenadas ingresadas corresponden al centro de la bomba.

Una vez ingresadas las coordenadas del disparo, si estas son válidas, se actualizarán los tableros y se

¹La función `isalnum()` podría serte de utilidad para esto.

²Para esta funcionalidad puedes usar la función `sort()` de python.

volverá a mostrar el **Menú de Juego**. En caso de no ser válidas, se debe indicar en consola y volver a pedir las coordenadas hasta que se ingresen coordenadas válidas.

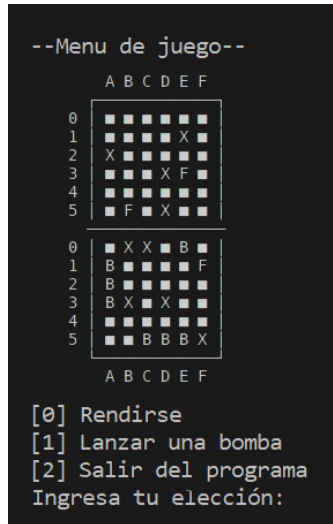


Figura 3: Ejemplo de menú de juego.

4. Reglas

El objetivo del juego es hundir todas las naves del oponente antes de que este acabe con las del jugador. En pantalla se mostrarán dos tableros: el del jugador y el del oponente. En el tablero del jugador podrás ver la distribución de los barcos del jugador y dónde ha disparado el oponente. En el tablero del oponente verás donde ha disparado el jugador y si ha dado al blanco a alguno de los barcos del oponente.

Las reglas básicas del juego son:

- Una vez posicionados los barcos e iniciada la partida, nunca cambiarán de posición.
- No pueden haber dos barcos en la misma casilla.
- No puedes volver a disparar en lugares donde ya disparaste.
- No se pueden deshacer disparos.
- Si no aciertas a un barco oponente, se pasa al turno del contrario. Si le das a un barco oponente, puedes volver a disparar.
- La partida acaba cuando un jugador se rinde, ha hundido la flota completa del oponente, o cuando todos los barcos del jugador han sido destruidos por este.

5. Elementos

A continuación se muestran los elementos que componen una partida de *DCCombateNaval*:

5.1. Tablero

Cada vez que se inicie el **Menú de Juego** o se ejecute un **disparo**, deberás representar en consola el tablero de juego. Este tablero contiene dos mapas: **Mapa del Jugador** y **Mapa del Oponente**. Cada uno de ellos deberá ser representado en forma de listas de listas, cumpliendo con las dimensiones ingresadas por el

jugador en el [Menú de Inicio](#). Estas listas formarán un conjunto de celdas indicando las coordenadas de cada elemento del mapa. Cada celda se compondrá de un **str de largo uno**.

5.1.1. Mapa del Jugador

- Las celdas que no han sido descubiertas y **no** contienen barcos están representadas por un espacio (' ').
- Las celdas que no han sido descubiertas y **sí** contienen un barco están representadas por el **str 'B'**.
- Las celdas que han sido descubiertas y **no** contienen barcos están representadas por el **str 'X'**.
- Las celdas que han sido descubiertas y **sí** contienen barcos están representadas por el **str 'F'**.

5.1.2. Mapa del Oponente

- Las celdas que **no** han sido descubiertas están representadas por un espacio (' ').
- Las celdas que no han sido descubiertas y **sí** contienen un barco están representadas por el **str 'B'**. Al momento de mostrar estas celdas en la consola, deberán verse como si fueran celdas **no descubiertas**.
- Las celdas que han sido descubiertas y **no** contienen barcos están representadas por el **str 'X'**.
- Las celdas que han sido descubiertas y **sí** contienen barcos están representadas por el **str 'F'**.

Afortunadamente, tendrás a disposición el módulo [tablero.py](#), el cual te permitirá imprimir el tablero actual en el siguiente formato:

```
mapa_oponente = [
    [' ', ' ', ' ', ' ', 'X', ' ', ' '],
    [' ', 'B', 'B', 'B', 'X', ' ', ' '],
    ['X', ' ', ' ', ' ', 'B', ' ', ' '],
    ['X', ' ', ' ', 'X', 'F', ' ', ' '],
    [' ', 'B', 'B', 'X', 'F', ' ', ' '],
    [' ', 'F', 'B', 'X', ' ', ' ', ' '],
]

mapa_jugador = [
    [' ', 'X', 'X', ' ', 'B', 'F'],
    ['B', ' ', ' ', ' ', 'F', 'F'],
    ['B', 'X', ' ', ' ', ' ', ' '],
    ['B', 'X', ' ', 'X', ' ', ' '],
    [' ', ' ', ' ', ' ', ' ', 'X'],
    [' ', ' ', 'B', 'B', 'B', 'X'],
]
```

Figura 4: Mapas de jugadores en lista de listas.

	A	B	C	D	E	F
0	■	■	■	■	X	■
1	■	■	■	■	X	■
2	X	■	■	■	■	■
3	X	■	■	X	F	■
4	■	■	■	X	F	■
5	■	F	■	X	■	■

	A	B	C	D	E	F
0	■	X	X	■	B	F
1	B	■	■	■	F	F
2	B	X	■	■	■	■
3	B	X	■	X	■	■
4	■	■	■	■	■	X
5	■	■	B	B	B	X

Figura 5: Representación del tablero en consola

5.2. Barcos

Cada jugador dispondrá de una cantidad de barcos definida por una variable [NUM_BARCOS](#) almacenada en [parametros.py](#). Al momento de [Crear](#) una partida, tanto los barcos del jugador como los barcos del

oponente deberán ubicarse de manera **aleatoria** sobre el tablero. Cada barco es de tamaño 1×1 y solo podrá estar ubicado en una única celda. Además, deberás verificar que la ubicación asignada a cada uno de los barcos sea válida, es decir, que no se sobreponga con otro y que no sea posicionado fuera del mapa.



Figura 6: Esto pasa si permites que se sobrepongan.

5.3. Bombas

A lo largo de la partida, deberás hacer uso de bombas para poder derribar los barcos del oponente. Tendrás dos categorías de bombas: Las **bombas regulares**, y las **bombas especiales**.

5.3.1. Bombas Regulares

Las **Bombas Regulares** corresponden a ataques **puntuales** a la celda ubicada en la posición donde se planea enviar el explosivo.

5.3.2. Bombas Especiales

Además de las **Bombas Regulares**, dispondrás de tres tipos de bombas las cuales podrán ser utilizadas una **única vez** por partida. Esto significa que solo podrás hacer uso de **una** bomba especial a lo largo de la partida, independiente del tipo que hayas escogido. Todas las bombas especiales tienen un **radio de explosión**³ definido por una variable almacenada en `parametros.py`, la cual corresponde a su alcance.

5.3.3. Bomba Cruz

Esta bomba tiene forma de cruz.

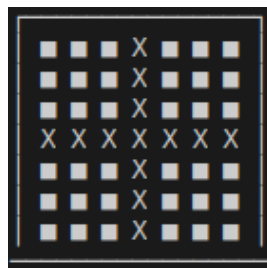


Figura 7: Ejemplo de Bomba Cruz con $r = 4$

5.3.4. Bomba X

Esta bomba tiene forma de X.

³Variable de tipo `int` el cuyo valor será siempre **mayor o igual** a 1.

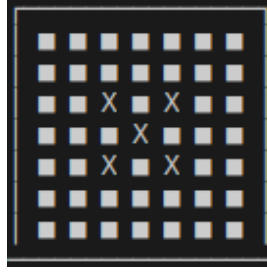


Figura 8: Ejemplo de Bomba X con $r = 2$

5.3.5. Bomba Diamante

Esta bomba tiene forma de diamante.

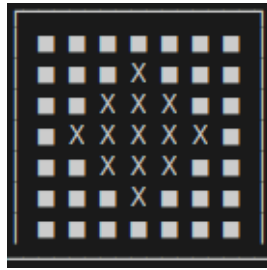


Figura 9: Ejemplo de Bomba Diamante con $r = 3$

Cabe mencionar que deberás manejar todos los casos en donde las celdas descubiertas se encuentren fuera del tablero. Además, para los casos en que el **radio de explosión** sea igual a 1, independiente del tipo de bomba, los efectos de la bomba serán iguales a los de una bomba regular.

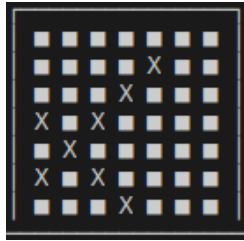


Figura 10: Bomba X con $r = 4$

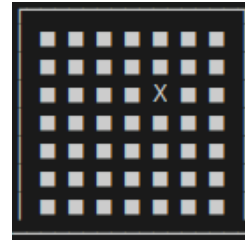


Figura 11: Bomba Especial con $r = 1$

6. Partida

6.1. Crear

El programa debe permitir iniciar nuevas partidas. Al momento de iniciar una, se debe preguntar por el tamaño que tendrá el tablero de ambos participantes. Para esto el jugador deberá ingresar dos enteros N y M pertenecientes al rango $[3,15]$ ⁴ que representarán el número de filas y columnas del mapa, respectivamente. Una vez ingresados los valores, el programa deberá crear un mapa con las dimensiones respectivas para tanto el jugador como el oponente. Luego, deberá posicionar los barcos de manera aleatoria⁵ en ambos mapas, verificando que cada casilla contenga sólo un barco. La cantidad de barcos por mapa estará definida

⁴Debes verificar que los números estén dentro del rango establecido, incluyendo ambos extremos.

⁵Para esta funcionalidad puedes usar la librería [random](#) de python.

por el parámetro `NUM_BARCOS`. Podrás asumir que este valor siempre será menor a la cantidad de celdas del tablero, por lo que no deberás preocuparte de los casos donde hayan más barcos que celdas.

6.2. Oponente

Para poder jugar adecuadamente es necesario la presencia de un contrincante que responda a los ataques del jugador, para lo cual tendrás que diseñar un **oponente** que interactúe con el jugador. Luego de cada turno, el oponente tendrá que jugar de forma **automática** con las mismas posibilidades que el jugador, es decir, deberá acatar las reglas generales del juego (explicadas en la sección [Reglas](#)) y las mismas restricciones que el jugador, con la única diferencia de que **nunca hará uso de bombas especiales**, sino que solo atacará con bombas regulares. Por último, al acabar el turno del oponente se deberá imprimir las **coordenadas** de la casilla a la que lanzó una bomba, seguido del mapa de ambos jugadores con las casillas actualizadas.

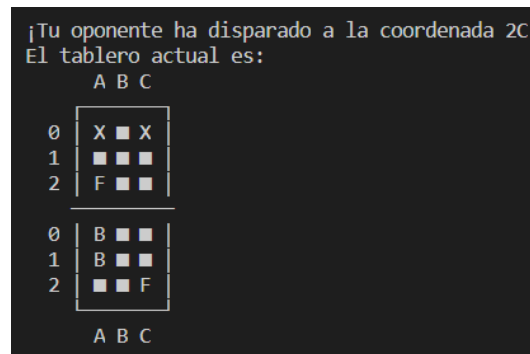


Figura 12: Ejemplo de ataque del oponente

7. Puntajes

Al concluir la partida, ya sea con una victoria, derrota, o porque el jugador se ha rendido, se le deberá asignar un puntaje al jugador. Este se calculará utilizando la siguiente fórmula:

$$\text{max}(0, (\text{filas} \times \text{columnas} \times \text{NUM_BARCOS} \times (\text{enemigos_descubiertos} - \text{aliados_descubiertos})))$$

Donde:

- *filas*: cantidad de filas en cada tablero de la partida.
- *columnas*: cantidad de columnas en cada tablero de la partida.
- `NUM_BARCOS` (número de barcos): constante que se almacena en el archivo `parametros.py`.
- *enemigos_descubiertos*: celdas de barcos enemigos descubiertas.
- *aliados_descubiertos*: celdas de barcos aliados descubiertos.

Por ejemplo, para el tablero con 6 filas y 6 columnas, mostrado a continuación, y tomando como parámetro `NUM_BARCOS` = 10 el puntaje sería:

	A	B	C	D	E	F
0					X	
1					X	
2	X					
3	X			X	F	
4		F		X	F	
5		F		X		

	A	B	C	D	E	F
0		X	X		B	F
1	B				B	B
2	B	X				
3	B	X		X		
4						X
5			B	B	B	X

$$Puntaje = \text{max}(0, (6 \times 6 \times 10 \times (4 - 1))) = 1080$$

Figura 13: Tablero y su respectivo puntaje.

Una vez calculado el puntaje, este deberá automáticamente guardarse en el archivo `puntajes.txt`.

8. Archivos Entregados

Para facilitar tu trabajo se te hará entrega de los siguientes módulos: `parametros.py` y `tablero.py`.

Estos módulos **no** deben ser modificados, ni debe escribirse código en ellos. Además, para acceder a los valores o funciones que éstos contienen debes **importarlos correctamente**⁶

8.1. `tablero.py`

En este módulo se entrega la función `print_tablero(tablero_rival, tablero_propio, utf8)` la cual permite imprimir el tablero actual de la partida. La función recibe los siguientes argumentos:

- **tablero_rival**: lista de listas que contienen el estado del tablero del oponente. Esta tiene el siguiente formato:
 - Cada sub-lista representa una **fila** del tablero, donde cada elemento de la sub-lista es una celda.
 - Las celdas donde no se haya disparado se representan con un espacio (' ').
 - Las celdas descubiertas se simbolizan con su **str** correspondiente.
- **tablero_propio**: lista de listas que contienen el estado del tablero del jugador. Esta tiene el mismo formato que la lista del argumento anterior.
- **utf8**: 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**.

⁶Para mas información revisar el [material de modularización](#) de la semana 0.

```

mapa_rival = [
    [' ', 'X', ' ', ' ', 'X'],
    [' ', 'B', ' ', ' ', 'B'],
    ['B', 'X', ' ', ' ', 'F'],
]

mapa_jugador = [
    [' ', 'X', 'B', ' ', 'X'],
    [' ', 'B', ' ', ' ', ' '],
    ['X', ' ', ' ', 'B', 'F'],
]

```

(a) Representación del tablero

```

  A B C D
0  [X][ ][X]
1  [ ][ ][ ][ ]
2  [X][ ][F]

  X B X
  [B][ ][ ]
X [ ][B] F
  A B C D

```

(b) Tablero con utf-8

```

  A B C D
0  - X - X
1  - - - -
2  - X - F

0  - X B X
1  - B - -
2  X - B F
  A B C D

```

(c) Tablero sin utf-8

8.2. parametros.py

Al momento de escribir programas de mayor complejidad, como lo son las tareas del curso, una buena práctica es **parametrizar** ciertos datos o variables que pueden variar entre ejecuciones de este, o que dependen de la estructura de archivos del computador del usuario. El tener todos los parámetros almacenados en un único archivo permite identificarlos y modificarlos de forma simple y rápida.

Para esta tarea te entregamos un archivo `parametros.py` ya relleno. En este archivo se encontrarán los parámetros mencionados anteriormente en el enunciado, en donde cada línea almacena una constante con su respectivo valor.

Particularmente, este archivo contiene:

```

1  NUM_BARCOS = 3  # Número de Barcos iniciales de cada jugador
2  RADIO_EXP = 3  # Radio de explosión de las bombas especiales

```

También se te entregará un archivo de ejemplo `main.py`, en el cual se encontrarán los parámetros ya importados y ejemplos de cómo utilizarlos.

8.3. puntajes.txt

Finalmente, se te entregará un archivo `puntajes.txt` con puntajes almacenados. Tu deber será continuar rellenoando este archivo conforme los jugadores vayan completando partidas. Recuerda que **todos** los puntajes de las partidas finalizadas deberán estar contenidas en este. El formato de guardado es **usuario,puntaje** como en el siguiente ejemplo:

```

1  misi99,120
2  sushi,36
3  0pitu0,0

```

9. 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 legible⁷. En este curso te pediremos seguir un pequeño apartado de estas reglas, el cual puede ser encontrado en la [guía de estilos](#).

- **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](#).

10. 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) y **qué cosas fueron implementadas, y qué cosas no**, 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 te entregará una [plantilla](#) (*template*) a rellenar con la información adecuada.

Finalmente, como forma de motivarte a redactar buenos READMEs, todas las tareas tendrán **décimas de des-descuento** si el ayudante considera que tu README fue especialmente útil para la corrección. Estás décimas anulan décimas de descuento que les hayan sido asignadas hasta un máximo de cinco.

11. 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:** (1 décima)

Se descontará una décima si no se indica(n) los archivos principales que son necesarios para ejecutar la tarea o su ubicación dentro de su carpeta. También se descontará una décima si es que no se hace entrega de un README. Esto se debe a que este archivo facilita considerablemente la corrección de las tareas.

- **Formato de entrega:** (hasta 5 décimas)

Se descontarán hasta cinco décimas si es que no se siguen 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 otros⁸. Esto se debe a que en próximos ramos (o en un futuro trabajo) no se

⁷Símil a como la ortografía nos ayuda a estandarizar la forma es que las palabras se escriben.

⁸Uno de los puntos a revisar es el uso de *paths* relativos, para más información revisar el siguiente [material](#).

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. 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.

- **Adicionales:** (hasta 5 décimas)

Se descontarán hasta cinco décimas a criterio del ayudante corrector en caso de que la tarea resulte especialmente difícil de corregir, ya sea por una multitud de errores o porque el programa sea especialmente ilegible. Este descuento estará correctamente justificado.

- **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.

- **Malas prácticas:** (hasta 5 décimas)

Al igual que los *built-ins* prohibidos, también se prohibirán ciertas malas practicas y se descontarán entre una y cinco décimas si se realizan. Para cada tarea se creará una *issue* donde se especificará qué malas prácticas estarán prohibidas. Es tu responsabilidad leerla y preguntar en caso de tener dudas sobre las malas prácticas establecidas.

- **Entrega atrasada:** (entre 5 a 20 décimas)

Las tareas serán recolectadas automáticamente y no se considerará ningún avance realizado después de la hora de entrega. Sin embargo, se puede optar por entregar la tarea de forma atrasada y se descontarán 5 a 20 décimas dependiendo de cuánto tiempo de diferencia haya entre la hora de entrega y la entrega atrasada.

- **Des-descuento:** (entre 1 a 5 décimas)

Finalmente, se des-descontarán hasta 5 décimas por un README especialmente útil para la corrección de la tarea.

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

12. .gitignore

Cuando estés trabajando con repositorios, muchas veces habrán archivos y/o carpetas que no querrás subir a la nube. Por ejemplo, puedes estar trabajando con planillas de Excel muy pesadas, o tal vez estás utilizando un Mac y no quieres subir la carpeta `__MACOSX`, o el archivo `.DS_Store`, entre otros.

Una posible solución es simplemente tener cuidado con lo que subes a tu repositorio. Sin embargo esta “solución” es extremadamente vulnerable al error humano y podría terminar causando que subas muchos *gigabytes* de archivos y carpetas no deseados a tu repositorio.⁹

Para solucionar esto, `git` nos da la opción de crear un archivo `.gitignore`. Éste es un archivo **sin nombre, y con extensión .gitignore**, en el cual puedes detallar **archivos y carpetas a ser ignoradas por git**. Esto quiere decir que todo lo especificado en este archivo **no será subido a tu repositorio accidentalmente**, evitando los problemas anteriores.

⁹Lamentablemente basado en una historia real.

En esta ocasión el uso de este archivo **no será evaluado**, pero se recomienda su realización para que aprendan a crearlo y utilizarlo ya que será evaluado en las siguientes tareas.

Se recomienda utilizar el archivo `.gitignore` para ignorar archivos en tu entrega, específicamente el enunciado, los archivos indicados en [Archivos Entregados](#) y todos los que no sean pertinentes para el funcionamiento de tu tarea. El archivo `.gitignore` debe encontrarse dentro de tu carpeta T00. Puedes encontrar un ejemplo de `.gitignore` en el siguiente [link](#).

13. 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, mientras que 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. En caso de que no logres implementar cierto ítem, prueba saltarlo e implementar los que te falten, procurando mantener tu tarea funcional.

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 jefe de Bienestar al correo bienestar.iic2233@ing.puc.cl.

14. 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 3.7.
- 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. **Tendrás hasta 48 horas después del plazo de entrega** de la tarea para subir el `README` a tu repositorio.
- Tu tarea podría sufrir los descuentos descritos en la [guía de descuentos](#).
- Entregas con atraso de más de 24 horas tendrán calificación mínima (1,0).
- 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).