

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-1

Midterm

11 de Abril 2025

Instrucciones

- La evaluación consta de dos partes: 20 preguntas de selección múltiple y 2 preguntas de desarrollo.
- Cada una de las secciones (alternativas y desarrollo) son independientes y valen un 50 % de la nota final. Para calcular la nota de cada sección, se considera que todas las preguntas de alternativas valen lo mismo, mientras que la sección de desarrollo indica cuánto vale cada pregunta.
- Recibirás una hoja de respuestas que deberás rellenar con tus datos y las respuestas correspondientes a cada alternativa o pregunta de desarrollo. Sólo se corregirá la hoja de respuestas. **Ten mucha precaución de anotar correctamente tus datos**.
- Cada pregunta de selección múltiple tiene únicamente 1 alternativa correcta. Responder con 2 o más alternativas implicará dejar inválida esa pregunta y se considerará incorrecta. Cada pregunta presenta el mismo puntaje y no se descontará por respuesta incorrecta.
- Debes completar con tus datos personales en cada hoja utilizada. Luego debes entregar de vuelta **todas** las hojas de respuesta (tanto para alternativas como de desarrollo) de la prueba, aunque estas se encuentren en blanco.
- Solo podrás retirarte de la sala una vez hayan transcurrido 60 minutos desde que inició de la evaluación.
- Durante la evaluación se realizarán 2 rondas de preguntas. Estas preguntas únicamente serán respondidas por los profesores del ramo.
- En caso de que sea necesario, podrás solicitar hojas blancas para apoyar al desarrollo de la prueba. Para esto levanta la mano y espera que un ayudante se acerque a tu puesto.

Selección múltiple

1. Respecto a la importación de módulos en Python, ¿cuál de las siguientes afirmaciones es **correcta**?

   A) La importación de módulos siempre se debe realizar en las primeras líneas del archivo o el programa lanzará un error.

   B) No es posible mezclar distintas formas/sentencias de importación de módulos en un mismo archivo.

   C) El código del módulo a importar se ejecuta en cuanto se llega a la sentencia `import` respectiva, y no solo cuando se hace uso de alguna definición de este.

   D) Al importar un módulo, Python automáticamente detecta y no ejecuta los `print()` que puedan existir dentro de este.

   E) Si dos módulos distintos contienen dos funciones distintas pero con el mismo nombre, no será posible importar y hacer uso de ambas funciones en el archivo que las importa.

2. ¿Cuál de las siguientes opciones **NO** corresponde a una ventaja **vista en el curso** de ejecutar archivos de Python desde la terminal?

   A) Saber claramente nuestra ubicación en el sistema de archivos al momento de ejecutar el archivo.

   B) Evitar problemas con la apertura de archivos, los *imports* y con las rutas relativas dentro del código.

   C) Es posible ejecutar más de un archivo a la vez, abriendo múltiples terminales.

   D) Poder ejecutar archivos de *tests* asociados a algún programa de forma configurable.

   E) Siempre poder ejecutar un archivo de Python sin que arroje errores, sin importar el sistema operativo del computador en que se haya desarrollado.

3. ¿Cuál de los siguientes comandos de **Git** implica traer información desde el repositorio **remoto** al computador **local**?

   I. `git clone`

   II. `git pull`

   III. `git commit`

   IV. `git push`

   A) Sólo I

   B) I y II

   C) I y III

   D) I, II y III

   E) I, II y IV

4. En base al siguiente código, ¿**cuántas** reglas **distintas** de PEP8 de las **evaluadas en el curso no** se están cumpliendo? Puedes suponer que el código usa 4 espacios para indentar y las líneas tienen menos de 100 caracteres.

```
from random import randint


class casino:
    def __init__(self):
        self.a = 0 ### Dinero que ha ganado la persona en el casino
        self.b = 100 ### Dinero que puedes ganar en un juego

    def tombola(self,numeros,numero_escogido):
        from time import sleep
        print("Apostando...")
        sleep(1)
        ganador = randint(1, numeros)
        if ganador == numero_escogido:
            self.a = (self.a+self.b)
            return print("¡Ganaste la tombola!")
        else:
            print("Perdiste la tómbola :(")
            return

instancia_casino = casino()
instancia_casino.tombola(100, 42)

```

A) 1

B) 2

C) 3

D) 4

E) 5

5. Un programador está usando **Git** para manejar las versiones de su código. La siguiente figura muestra en la izquierda las carpetas y archivos presentes en su código, marcando con un * a los elementos que fueron modificados desde el *commit* anterior. En la derecha, se muestra la sucesión de comandos de Git utilizados para subir cambios al repositorio remoto.

```
a.py*
data/
    b.txt*
outputs/
    c.txt*
    d.txt*
e.txt
f.txt*
g.uwu*
```

```
git add a.py
git add b.txt
git add /outputs/
git add e.txt
git add g.uwu
git commit -m "Mi entrega"
git add f.txt
git push

```

En base a lo anterior, ¿cuál de las siguientes aseveraciones es **verdadera**?

A) Los cambios no se subirán al repositorio remoto, ya que faltó el comando `git status` al inicio de la secuencia.

B) El archivo `b.txt` será subido en el nuevo commit ya que Git busca de forma recursiva dentro de carpetas hasta encontrar el archivo.

C) Los archivos `c.txt` y `d.txt` serán subidos ya que se usó un comando para agregar la carpeta completa que los contenía.

D) El archivo `g.uwu` no será subido ya que tiene una extensión inválida que no es usada por ningún programa.

E) Los archivos que contendrá el commit recién subido serán solamente `a.py` y `g.uwu`.

6. Estás desarrollando un bot para un juego online multijugador. Cada jugador puede enviar comandos al bot mientras juega. El bot debe mantener un historial de los comandos pendientes por procesar, priorizando los primeros comandos en llegar. ¿Qué estructura de datos sería la más adecuada para implementar el manejo de respuestas?

A) Stack

B) Tupla

C) Cola

D) Set

E) Diccionario

7. Sea A y B dos *sets* que almacenan *ints*, ¿cuál de las siguientes acciones **no** se pueden hacer sobre ellos?

   A) Identificar elementos repetidos entre A y B.

   B) Ordenar internamente A de forma descendiente.

   C) Encontrar el elemento más grande de B.

   D) Identificar los elementos que solo pertenecen a A, pero no a B.

   E) Comparar si A tiene más elementos que B.

8. Se tiene un diccionario donde cada llave representa a un estudiante, y cada valor es un `set` con los cursos que ha completado.

```
cursos_realizados = {
    "Sofi": {"Programación", "Bases de Datos", "EDD"},
    "Tomás": {"Programación", "EDD", "Software"},
    "Fran": {"EDD", "Programación", "Sistemas Operativos"}
}
```

   ¿Cuál de las siguientes expresiones permite obtener el conjunto de cursos que **han sido completados por todas las personas**?

   A) `set(cursos_realizados.values())`

   B) `cursos_realizados["Sofi"] | cursos_realizados["Tomás"] | cursos_realizados["Fran"]`

   C) `cursos_realizados["Sofi"] & cursos_realizados["Tomás"] & cursos_realizados["Fran"]`

   D) `list(cursos_realizados.keys()) & list(cursos_realizados.values())`

   E) `set.intersection(cursos_realizados)`

9. Considerando el uso de `*args` y `**kwargs` en la definición y ejecución de funciones, ¿cuál de las siguientes afirmaciones es **verdadera**?

   A) Al ejecutar una función, los `*args` y `**kwargs` pueden estar vacíos.

   B) Si la definición de una función presenta `*args` entonces debe presentar `**kwargs` y viceversa.

   C) Al ejecutar una función, los `*args` siempre tendrán la misma información que los `**kwargs` y viceversa.

   D) Al definir una función, no importa el orden de los argumentos, `*args` y `**kwargs`.

   E) Al definir una función, es obligatorio usar los términos `args` y `kwargs` para definir `*args` y `**kwargs`.

10. ¿Cuál de las siguientes opciones corresponde a una acción **inválida** sobre las estructuras de datos vistas en clase, en base a su implementación en Python?

    A) Usar una `namedtuple` como una llave de un diccionario.

    B) Colocar una lista dentro de una `tuple` y modificarla posteriormente.

    C) Editar un elemento que no se encuentre en los extremos en una `cola` o `stack`.

    D) Agregar `deques` vacíos dentro de un `set`.

    E) Crear dos tuplas con los mismos valores internos y agregarlos a un `set`.

11. Respecto a herencia de clases, ¿cuál o cuáles de las siguientes afirmaciones son **verdaderas**?

    I. Corresponde a una relación de especialización y generalización entre clases.

    II. Cuando una subclase hace *overriding* de un método, este afecta directamente al método de la superclase.

    III. En Python, una clase no puede heredar de más de una superclase.

    A) Solo I

    B) Solo II

    C) I y II

    D) I y III

    E) I, II y III

12. ¿Cuál de las siguientes afirmaciones sobre funciones como objetos de primera clase en Python es **correcta**?

    A) Las funciones en Python solo pueden llamarse, no pueden ser almacenadas como datos.

    B) Las funciones en Python pueden ser pasadas como argumentos, retornadas desde otras funciones y asignadas a variables, lo que permite construir decoradores.

    C) Los decoradores solo se pueden usar si son métodos dentro de clases, no si son funciones individuales.

    D) El decorador sólo puede ser la función interna.

    E) Las funciones no pueden ser almacenadas en estructuras como listas o diccionarios, solo en módulos.

13. Dado el siguiente código, ¿cuál de las siguientes aseveraciones es **incorrecta**?

```
from abc import ABC, abstractmethod


class Instrumento(ABC):
    def afinar(self):
        print("Afinando el instrumento...")

    @abstractmethod
    def tocar(self):
        pass

class Guitarra(Instrumento):
    def tocar(self):
        print("La guitarra toca un solo.")

class Piano(Instrumento):
    def afinar(self):
        return "Piano afinado por un profesional."

    def tocar(self):
        print("El piano interpreta una pieza clásica.")

gibson = Guitarra()
```

A) Si una subclase no implementa todos los métodos abstractos de su superclase, no se puede instanciar y se lanza un `TypeError`.

B) En el ejemplo, si `Guitarra` no implementara el método `tocar()`, se lanzaría un error al intentar instanciar una Guitarra.

C) En cuanto a `afinar()`, sí se puede hacer *override* y se puede cambiar su comportamiento como se hizo en `Piano`.

D) El método `afinar()` de la clase `Instrumento` es abstracto, por lo tanto, debe ser implementado por todas las subclases.

E) La clase `Piano` hace *override* correctamente tanto del método `tocar` como del método `afinar`.

14. En el siguiente fragmento, un objeto `LiderTecnico` utiliza su método `trabajar()`, el cual llama directamente a los métodos `trabajar()` de las clases `Contador` e `Ingeniero`. Aunque ambas heredan de `Empleado`, se observa que el método base se ejecuta más de una vez.

```
class Empleado:
    llamadas_empleado = 0
    def trabajar(self):
        Empleado.llamadas_empleado += 1

class Contador(Empleado):
    llamadas_contador = 0
    def trabajar(self):
        Empleado.trabajar(self)
        Contador.llamadas_contador += 1

class Ingeniero(Empleado):
    llamadas_ingeniero = 0
    def trabajar(self):
        Empleado.trabajar(self)
        Ingeniero.llamadas_ingeniero += 1

class LiderTecnico(Contador, Ingeniero):
    llamadas_lider = 0
    def trabajar(self):
        Contador.trabajar(self)
        Ingeniero.trabajar(self)
        LiderTecnico.llamadas_lider += 1

lt = LiderTecnico()
lt.trabajar()
# Resultado: Empleado.llamadas_empleado == 2
```

Se observa que `Empleado.llamadas_empleado` toma el valor 2 tras la ejecución anterior. ¿Cuál de las siguientes afirmaciones explica por qué el valor mostrado de `llamadas_empleado` es mayor a 1?

A) Porque `LiderTecnico` hereda de ambas clases y Python ejecuta automáticamente todos los métodos de las clases base.

B) Porque el método se ejecuta una sola vez por clase, pero las llamadas se repiten como si fueran múltiples debido a la estructura de herencia.

C) Porque el método `trabajar` de `Empleado` se invoca más de una vez desde rutas distintas de la jerarquía de herencia.

D) Porque los atributos de clase como `llamadas_empleado` se comparten entre subclases y causan duplicación.

E) Porque Python reutiliza la misma función varias veces durante la ejecución al detectar herencia múltiple.

15. ¿Qué imprimirá el siguiente código?

```
class CajaSecreta:
    def __init__(self, valor):
        self.valor_secreto = valor
        self.multiplicador_secreto = 1

    @property
    def multiplicador(self):
        return self.multiplicador_secreto * 2

    @property
    def valor(self):
        return self.valor_secreto * self.multiplicador

    @valor.setter
    def valor(self, otro):
        self.multiplicador_secreto = otro

caja = CajaSecreta(1)
caja.valor_secreto *= 2
caja.valor += caja.multiplicador
print(caja.valor)
```

A) 2

B) 4

C) 8

D) 12

E) 24

16. ¿Cuál o cuáles son los beneficios de manejar excepciones con `try/except`?

A) El código queda más limpio y fácil de leer.

B) Obliga al programador a tratar las excepciones.

C) Le da más flexibilidad al programa.

D) Permite "notificar" a otras partes del programa sobre estos errores.

E) Todas las alternativas son correctas.

17. A partir del siguiente *traceback*, ¿qué alternativa muestra el flujo que siguió el código desde el inicio de su ejecución hasta que se produjo el error?

```
Traceback (most recent call last):
  File "main.py", line 13, in <module>
    func2(2, 2)
  File "main.py", line 7, in func2
    return func1(var1 + 1, var2 - 1)
           ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "main.py", line 3, in func1
    return func2(var1, var2)
           ^^^^^^^^^^^^^^^^^
  File "main.py", line 7, in func2
    return func1(var1 + 1, var2 - 1)
           ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "main.py", line 4, in func1
    return func3(var1, var2)
           ^^^^^^^^^^^^^^^^^
  File "main.py", line 10, in func3
    return var1 / var2
           ~~~~~^~~~~
ZeroDivisionError: division by zero
```

A) `func3`, **`ZeroDivisionError`**

B) `func1`, `func3`, **`ZeroDivisionError`**

C) `func2`, `func3`, **`ZeroDivisionError`**

D) `func1`, `func2`, `func1`, `func3`, **`ZeroDivisionError`**

E) `func2`, `func1`, `func2`, `func1`, `func3`, **`ZeroDivisionError`**

18. ¿Cuál de los siguientes tipos de datos **no** presenta un iterador?

A) *Strings*

B) Conjuntos

C) Tuplas

D) Diccionarios

E) Todos los tipos de datos anteriores presentan un iterador.

19. Sin considerar saltos de líneas, ¿qué imprime el siguiente código?

```
try:
    try:
        print('TRY' + 1)
    except:
        print('EXCEPCION1')
except:
    print('EXCEPCION2')
else:
    print('ELSE')
finally:
    print('FINALLY')
```

A) `TRY1 ELSE FINALLY`

B) `EXCEPCION1 EXCEPCION2 FINALLY`

C) `EXCEPCION1 ELSE FINALLY`

D) `EXCEPCION2 FINALLY`

E) `FINALLY`

20. De las siguientes opciones, ¿cuál contiene la **mínima** cantidad de información para poder implementar una estructura **iterable** que se pueda recorrer directamente mediante el uso de `for`?

A) Una clase Iterable con el método `__iter__`, y una clase Iterador con el método `__next__`.

B) Una clase Iterable con el método `__iter__`, y una clase Iterador con los métodos `__iter__` y `__next__`.

C) Una clase Iterable con el método `__next__`, y una clase Iterador con el método `__iter__`.

D) Una clase Iterable con los métodos `__iter__` y `__next__`, y una clase Iterador con el método `__next__`.

E) Una clase Iterable con los métodos `__iter__` y `__next__`, y una clase Iterador con los métodos `__iter__` y `__next__`.

Preguntas desarrollo

Pregunta 1 - Modelación OOP [6 puntos]

Imagina un universo en el que existe un ser legendario llamado **Mark**, cuya existencia trasciende los límites de la realidad. Mark además se manifiesta en diversas formas, las cual se conocen como **Variantes**. Cada una con habilidades y características especiales que las definen. En este mundo ficticio, Mark y cada una de sus Variantes poseen una **Realidad** de origen, la cual consiste en un entorno con un **nombre**, un **estado** que determina si se encuentra en guerra actualmente o no y una **descripción** que narra la esencia del lugar.

Además de poseer una Realidad de origen, Mark cuenta con **características fundamentales**: una **fuerza**, **agilidad** y **defensa**, que se combinan para formar su **nivel de poder**, el cual se calcula de manera dinámica utilizando dichas características de Mark como base. Ahora bien, este poder base se ve modificado por la naturaleza de cada variante de Mark. Finalmente, Mark puede atacar a sus enemigos, pero como recién está comenzando su aventura, aún la gente no lo percibe ni como **SuperHeroe** ni como **SuperVillano**.

Existen **tres variantes** principales de Mark.

> La primera, conocida como **MarkAlpha**, es aquella que siempre se alinea con el heroísmo, por lo que en todos lados es percibido como un **SuperHeroe**. Además, esta versión se caracteriza por siempre portar un **accesorio único** que le entrega ciertas características. Este accesorio puede ser de un tipo especifico y poseer un poder único asociado, el cual incrementa su nivel de poder.
>
> La segunda, llamada **MarkBeta**, se distingue por tener un tipo de **piel singular**. Esta variante se caracteriza por aborrecer a cualquier otro tipo de criatura que no posea su tipo de piel, por lo cual, su corazón se inclina hacia el lado oscuro, convirtiéndolo en un **SuperVillano** implacable a ojos de todos. Además, este rasgo viene acompañado de un conjunto de factores que alteran el valor de sus características bases, modificando la forma en que se calcula su nivel de poder.
>
> La tercera, **MarkGamma**, es la más enigmática, pues se caracteriza por portar un **símbolo místico** en la frente, el cual tiene una figura específica y un nivel de intensidad específico cuando brilla. Este **no influye en el cálculo del nivel de poder**. Sin embargo, luego de un proceso de canalización de energía, el cual puede ocurrir de manera independiente a un combate, éste símbolo se enciende y genera un comportamiento único. Ahora bien al estar encendido durante una pelea, puede alterar el resultado de esta misma debido a dicho comportamiento. Gracias a este rasgo único, en él se entrelazan tanto el espíritu del héroe como el del villano, haciendo de su conducta algo dependiente de la naturaleza del enemigo al que se enfrenta.

Actualmente, todas estas Variantes han atacado la Realidad de Mark, por lo tanto los **combates** que se realicen entre Mark y sus Variantes o viceversa, son exclusivamente entre ellos.

Respecto al **combate**, aquellos Marks que actúan como **superhéroes**, además de dejar a su enemigo con vida, lo destinan a la cárcel. En contraste, los Marks con tendencias **supervillanas** se aseguran de que sus rivales nunca sobrevivan. En el caso de los Marks que mezclan ambas facetas, el destino del oponente **dependerá de la identidad de este mismo**: si el enemigo se trata de un SuperVillano, se aplicará la compasión heroica y se le encarcelará; pero si se trata de un SuperHeroe, la lógica impasible del villano se hará cargo y dicho enemigo no quedará con vida.

Usando conceptos de Programación Orientada a Objetos (POO), diseñe las clases que necesitaría un programa de Python que simule el contexto presentado anteriormente. **Realice un punteo o una tabla** acoplándose al siguiente formato para cada clase que cree. No es necesario implementar ningún tipo de código, sólo con escribir en palabras es suficiente:

1. **Clase.** Por cada clase que añada, indique:
    a. Nombre de la clase.
    b. Si es abstracta o no.
    c. Si hereda de otra clase, en caso de que corresponda, indica de cuál clase hereda.
2. **Atributos.** Por cada atributo que añada, indique:
    a. Nombre del atributo.
    b. Tipo de dato: *str*, *int*, *float*, *bool*, instancia de algún objeto, etc.
    c. Si es una *property* o no.
3. **Métodos.** Por cada método que añada, indique:
    a. Nombre del método.
    b. Argumentos que necesite, junto a cada uno de sus tipos de dato.
    c. Tipo de dato que retorna.
    d. Descripción breve de lo que hace el método (máximo 2 líneas).

En caso de estar describiendo una clase que hereda de otra, no es necesario repetir los atributos y/o métodos de la superclase respectiva **a menos** de que dicho atributo y/o método cambie su comportamiento en la subclase.

A continuación se entregan un par de **ejemplos de formatos válidos** para esta pregunta. El ejemplo es sobre una modelación entre dos clases que presentan una relación de herencia entre ellas. De ser necesario extrapole los datos para el resto de los tipos pedidos (atributos y métodos):

(a) Formato punteo

Clase: Perro

- ¿Es abstracta?: Sí
- ¿Presenta herencia?: No
- Hereda de: -

Clase: Golden

- ¿Es abstracta?: No
- ¿Presenta herencia?: Sí
- Hereda de: Perro

(b) Formato tabla

Clase	¿Es abstracta?	¿Presenta herencia?	Hereda de
Perro	Sí	No	-
Golden	No	Sí	Perro

Pregunta 2 - Estructuras de Datos [6 puntos]

Esta pregunta se divide en dos partes independientes. Ambas te presentan un problema de programación y deberás escribir pseudocódigo para solucionarlo. No es requerimiento que este pseudocódigo siga estrictamente la sintáxis de Python, sino que sea entendible cada paso que siga tu código.

En cada una de estas preguntas debes usar estructuras de datos apropiadas para resolver el problema de forma eficiente, y debes indicar de forma **explícita** las que usas. Las estructuras que puedes usar son: `list`, `tuple`, `namedtuple`, `dict`, `set` y `deque`. **NO** puedes usar clases ni funciones de **aleatoriedad**. Debes indicar explícitamente cuando uses alguna de las estructuras.

Parte A [3 puntos]

Se te ha pedido simular un **sistema que atiende a clientes**. Estos clientes tienen dos categorías excluyentes: **normales** y **preferentes**. Cada cliente está representado por un `string` que guarda su identificador único.

Deberás atender a los clientes por **orden** de llegada, pero teniendo en cuenta su **categoría**. Lo anterior se aplica tal que que si hay un preferente esperando, este debe ser atendido antes que cualquier cliente, a menos que vaya a ser el quinto preferente seguido que sea atendido y hayan normales esperando, en cuyo caso se atiende a un normal y se reinicia el contador de preferentes seguidos atendidos.

Para simular el sistema de recursos deberás usar una estructura, cuya instancia llamaremos "`sistema`", que te permita acceder a los atributos por su nombre: `normales` que guardará los usuarios por **orden** de llegada, `preferentes` que almacena los preferentes por **orden** de llegada, y `preferentes_seguidos` que almacena la **cantidad** de preferentes seguidos que han sido atendidos.

Deberás programar las siguientes funciones:

- `crear_sistema()`: No recibe parámetros. Retorna el sistema con los tres atributos con su estado inicial tal que luego se puedan usar. Inicialmente no hay `preferentes`, `normales`, ni se han atendido `preferentes_seguidos`.
- `recibir_cliente(sistema, cliente, es_preferente)`: Recibe el sistema, cliente y si es preferente o no. Agrega el cliente al sistema, añadiéndolo a `preferente` si es que `es_preferente` es `true` y en el otro caso lo agrega a `normales`. Retorna el sistema actualizado.
- `atender_cliente(sistema)`: Recibe el sistema. Simula el atender un cliente. Retorna el sistema actualizado.

Parte B [3 puntos]

Se te ha pedido simular un torneo en el que participan millones de personas, con cada persona representada por medio de su identificador universal único. El torneo se desarrolla mediante rondas que poseen partidas entre dos jugadores elegidas aleatoreamente y cada jugador compite una vez por ronda, tal que si hay m que siguen participando, se realizan $m/2$ partidas. El ganador permanece en el torneo, y el perdedor es eliminado. En cada ronda se realizan enfrentamientos entre los ganadores de la ronda anterior. El juego termina cuando solo queda una persona. Para facilitar la modelación del problema puedes suponer que la cantidad de personas es una potencia de 2.

Se te proporcionan las siguientes funciones:

- `obtener_personas()`, que retorna una tupla que permite acceder a cada persona individualmente, pero debido al tiempo que toma llamar esta función, **solo podrás llamarla una vez** en tu programa.
- `obtener_ganador(una_persona, otra_persona)`, que retorna uno de los dos argumentos como ganador.

Debes tener consideración de usar una estructura no secuencial para almacenar las personas que obtengas a partir de `obtener_personas()`, tal que con solo eso puedas simular la **aleatoriedad** de escoger cada persona.

Finalmente, tendrás que guardar los jugadores en la `distribucion_jugadores` de manera tal que con solo saber la **cantidad de victorias** se puedan obtener **directamente** todos los jugadores que poseen exactamente esa cantidad de victorias, y que se pueda verificar **rápidamente** si hay un jugador dentro de los que ganaron exactamente esa cantidad.

Nombre completo: _____

RUT: _____ Sección: _____ N° lista: _____ Página: ____ de ____

IIC2233 - Pregunta 1 (Modelación OOP)

Nombre completo: _____

RUT: _____ Sección: _____ N° lista: _____ Página: ____ de ____

IIC2233 - Pregunta 1 (Modelación OOP)

Nombre completo: _____

RUT: _____ Sección: _____ N° lista: _____ Página: ____ de ____

IIC2233 - Pregunta 2 (Estructuras de Datos)

Nombre completo: _____

RUT: _____ Sección: _____ N° lista: _____ Página: ____ de ____

IIC2233 - Pregunta 2 (Estructuras de Datos)