Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2024-1

Midterm

12 de abril 2024

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. Además, 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 40 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. ¿En cuál o cuáles de los siguientes contextos es apropiado utilizar *properties*?

   I. Encapsular el acceso a un atributo privado.

   II. Agregar comportamiento a un atributo de una clase.

   III. Validar el valor de un atributo.

   IV. Calcular un atributo de forma dinámica.

   A) Solo I

   B) Solo II

   C) I, III y IV

   D) II, III y IV

   E) I, II, III y IV

2. Se desea implementar el *setter* de una *property* en un código Python, ¿qué es lo **mínimo** que se debe implementar para que este funcione correctamente?

   I. El *getter* asociado a dicha *property*.

   II. Un atributo privado asociado a dicha *property*.

   III. Se debe anteponer `@property.setter` al método que corresponda.

   A) Solo I

   B) Solo II

   C) Solo III

   D) Solo I y III

   E) Solo II y III

A partir del siguiente código, que no presenta errores, responde las preguntas 3 y 4:

```python
import collections

class Banco:
    def __init__(self, dinero):
        self.dinero_setter(dinero)
        self.historial = collections.deque()

    @property
    def dinero(self):
        self.historial.appendleft(self._dinero_guardado)
        self.dinero = 2
        return self.historial[-1]

    @dinero.setter
    def dinero(self, dinero):
        self._dinero_guardado += dinero
        self.historial.append(self._dinero_guardado)

    def dinero_setter(self, dinero):
        self._dinero_guardado = dinero

banco = Banco(1)
banco.dinero = banco.dinero + 2
dinero = banco.dinero
print(banco.historial)
```

3. En base a la ejecución del código, ¿cuántas veces se llama al *getter* de la *property*?

   A) 0

   B) 1

   C) 2

   D) 3

   E) 5

4. A partir del código indicado, ¿cuál es el *output* en la consola producto del último `print`?

   A) `deque([1, 1, 1, 1, 1])`

   B) `deque([5, 1, 3, 5, 7])`

   C) `deque([3])`

   D) `deque([10, 8, 3, 1, 8])`

   E) `deque([8, 1, 3, 8, 10])`

5. ¿A qué concepto de Programación Orientada a Objetos hace referencia el siguiente código?

```
class Planta:
    def crecer(self):
        print('Necesito agua y sol para crecer, pero pronto seré una flor')

class Masa:
    def crecer(self):
        print('Necesito fermentar para crecer, pero pronto seré un pan')

class Huevo:
    def crecer(self):
        print('Necesito calor para crecer, pero pronto seré un pájaro')

for objeto in [Planta(), Masa(), Huevo()]:
    objeto.crecer()
```

A) Herencia

B) Polimorfismo

C) *Properties*

D) Clases abstractas

E) Ninguna de las anteriores

6. Considerando los posibles problemas que se pueden presentar al utilizar Multiherencia, ¿cuál o cuáles contenidos del curso son los **mínimos** y necesarios para enfrentarlos?

I. `*args` y `**kwargs`

II. `super()`

III. Clases abstractas

IV. *Properties*

A) Solo I

B) Solo II

C) I y II

D) II y IV

E) I, II, III y IV

7. En función del siguiente código, ¿cuál o cuáles clases se pueden instanciar sin provocar un error al momento de ejecutar el código?

```python
from abc import ABC, abstractmethod

class A(ABC):
    @abstractmethod
    def saludar(self):
        pass

    @abstractmethod
    def despedir(self):
        pass

class B(A):
    def saludar(self):
        print("Konnichiwa")

class C(B):
    def despedir(self):
        print("Sayonara")
```

A) Solo la clase A

B) Solo la clase B

C) Solo la clase C

D) Las clases B y C

E) Ninguna se puede instanciar

8. Dado un programa escrito en Python 3, el cual recibe el nombre de un archivo como argumento por medio de la línea de comandos. Según lo visto en el curso, ¿cuál es la forma de entregar dicho argumento?

A) `python3 main.py archivo`

B) `python3 main.py`

C) `python3 archivo main.py`

D) `python3 main archivo`

E) No es posible hacer lo pedido.

9. En base a la ejecución del código presentado a continuación, ¿cuál es el *output* en la consola producto del último `print`?

```python
class MiEstructura(list):
    def append(self, valor):
        self.insert(0, valor)
        self.insert(len(self), valor)
        return self.secreto

    @property
    def secreto(self):
        return self.pop(-1)

especial = MiEstructura()
e1 = especial.append("W")
e2 = especial.append("A")
e3 = especial.append("K")
e4 = especial.append("U")
print([e1, e2, e3, e4], especial)
```

A) `['W', 'A', 'K', 'U'] ['U', 'K', 'A', 'W']`

B) `['W', 'A', 'K', 'U'] ['W', 'A', 'K', 'U']`

C) `['W', 'A', 'K', 'U'] ['U', 'K', 'A', 'W', 'W', 'A', 'K', 'U']`

D) `['W', 'A', 'K', 'U'] <__main__.MiEstructura object at...>`

E) `[None, None, None, None] <__main__.MiEstructura object at...>`

10. ¿Cuál de las siguientes afirmaciones sobre comandos de terminal es **incorrecta**?

A) El comando `cd` permite cambiar el directorio actual.

B) El comando `cd` puede usarse para desplazarse varios directorios hacia adelante o hacia atrás.

C) El comando `mkdir` crea una carpeta nueva en la ruta indicada.

D) No se pueden crear archivos vacíos desde la terminal.

E) Al intentar crear una carpeta nueva con el comando `mkdir`, si la carpeta ya existe no se ejecuta ninguna acción.

11. ¿Cuál de las siguientes expresiones **no** corresponde a una ruta relativa?

A) `./Syllabus/README.md`

B) `../../Syllabus/datos.txt`

C) `../Syllabus/`

D) `/usuario/Escritorio/Syllabus/main.py`

E) `Syllabus/README.md`

12. El siguiente grupo de códigos contempla los archivos `main.py` y `utils.py`:

`main.py`

```python
from utils import analisis

palabra = input()
vocales(palabra)
print(utils.analisis(palabra))
```

`utils.py`

```python
def analisis(palabra):
    ultima_letra = palabra[-1]
    return ultima_letra

def vocales(palabra):
    print([letra for letra in palabra
           if letra in 'aeiou'])
```

Respecto a la importación y uso de `utils.py` en `main.py`, es correcto afirmar que:

I. Existen errores al usar la función `analisis` de `utils.py`.

II. Existen errores al usar la función `vocales` de `utils.py`.

III. No existen errores al usar estas funciones.

A) Sólo I

B) Sólo II

C) I y II

D) II y III

E) I, II y III

13. Con respecto al uso de Git, se presenta esta situación:

> Al momento de subir cambios desde tu repositorio local a tu repositorio remoto, notas que hay modificaciones que no se encuentran presentes en tu computador, pero que sí están presentes en GitHub.

Asumiendo que no hay conflictos, ¿cuál es el flujo de comandos que debes ejecutar para descargar los cambios remotos y después subir las modificaciones locales?

I. `git commit -m "Mensaje descriptivo"`

II. `git add`

III. `git pull`

IV. `git push`

A) III, II, I y IV

B) II, I y IV

C) II, I y III

D) IV, II, I y III

E) No es posible realizar lo pedido

14. Dado dos *threads* que se ejecutan concurrentemente, ¿en cuál o cuáles de los siguientes casos es **innecesario** el uso de *locks*?

    I. Uno de los *threads* aumenta un contador, y el otro lo disminuye.

    II. Ambos *threads* leen el contenido de un archivo `data.txt`.

    III. Ambos *threads* agregan elementos a la misma lista.

    A) Solo I

    B) Solo II

    C) Solo III

    D) I y III

    E) Ninguna de las anteriores.

15. ¿Cuál de las siguientes afirmaciones es **correcta** sobre *threads*?

    A) El fin de la de ejecución de un *thread daemon* depende del fin de ejecución del programa.

    B) Un *thread daemon* tiene prioridad cuando solicita un `lock`.

    C) Un `Timer` es un tipo de *thread* que detiene su ejecución después de una cierta cantidad de tiempo.

    D) Un *thread* **no** *daemon* es aquel que siempre se ejecuta primero si existe más de un *thread*.

    E) Solo puede ejecutarse un único *thread* **no** *daemon* en el programa.

16. Respecto a *threading*, ¿cuál o cuáles afirmaciones son **incorrectas**?

    I. Si dentro de una función se utiliza un `lock.acquire()`, no es necesario hacer `lock.release()` porque al momento de finalizar la función, el *lock* será liberado automáticamente.

    II. Varios *threads* pueden esperar a un mismo `threading.Event`.

    III. El método `.join()` es utilizado por la clase `threading.Event` para indicarle al *thread* que debe esperar hasta que el evento finalice.

    A) Solo I

    B) Solo II

    C) Solo III

    D) I y III

    E) II y III

17. Respecto a las estructuras de datos vistas en el curso, ¿cuál de las siguientes estructuras **no** permite alterar el largo de esta una vez instanciada, y solo permite contener elementos inmutables?

    A) Tupla

    B) *Set*

    C) Lista

    D) Cola

    E) Ninguna de las anteriores

18. Se te entrega el siguiente código para representar las conexiones de una red de metro mediante una lista de listas:

```
1  red = [[0] * 3] * 3
2
3  red[0][1] = 1
4  red[1][2] = 1
5  red[2][0] = 1
```

    ¿Cuál es el **principal** problema presenta el código anterior?

    A) No se indica el nombre de las estaciones.

    B) Las sublistas de `red` comparten un mismo espacio de memoria.

    C) No se puede utilizar el operador $*$ sobre listas.

    D) Se debería modelar las conexiones de la red como un diccionario.

    E) El código debería ser modelado con clases.

19. Se pide modelar la simulación de un puesto de peajes en la carretera y se han elegido diferentes estructuras de datos para cada situación. ¿Cuál de los siguientes usos **no** es adecuado?

    A) `deque()` para la cola de autos esperando pagar el peaje en una cabina.

    B) `set()` para registrar todos los montos de los pagos realizados.

    C) `namedtuple()` para representar a los funcionarios de cada cabina.

    D) `dict()` para relacionar el valor del peaje según el tipo de vehículo.

    E) `list()` para almacenar todos los ingresos de la jornada.

20. El siguiente código sin errores presenta una base de datos de películas y un código de consulta. ¿A qué dato accede?

```
dccflix_data = {
    "Inception": {
        "director": "Christopher Nolan",
        "anio": 2010,
        "genero": ["accion", "aventura", "sci-Fi"],
        "rating": 8.8,
        "cast": ["Leonardo DiCaprio", "Joseph Gordon-Levitt", "Elliot Page"]
    },
    "The Godfather": {
        "director": "Francis Ford Coppola",
        "anio": 1972,
        "genero": ["crimen", "drama", "politica"],
        "rating": 9.2,
        "cast": ["Marlon Brando", "Al Pacino", "James Caan"]
    }
}

dccflix_data.get("Inception").get("genero", 0)[1]
```

A) `"accion"`

B) `"aventura"`

C) `"c"`

D) `["accion", "aventura", "sci-Fi"]`

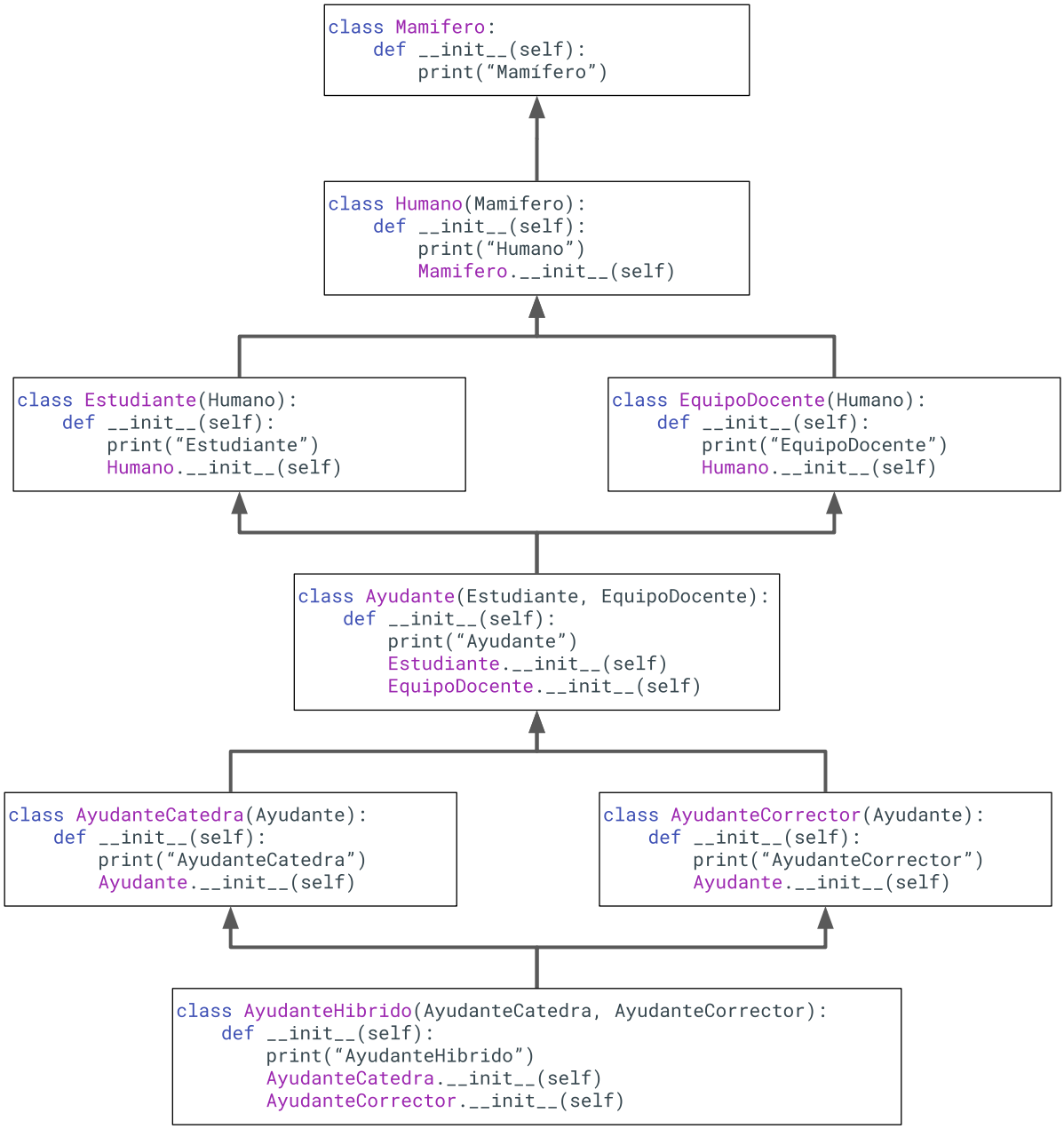
E) `"genero"`

Preguntas desarrollo

Pregunta 1 - Modelación OOP

Suponga que existe un programa que busca simular el comportamiento de una universidad.

A continuación se presenta un diagrama que representa la relación de herencia existente entre algunas clases del programa y parte del código relacionado a sus métodos inicializadores. A partir de la figura, conteste las preguntas correspondientes.



1. **(2 puntos)** Suponga que se instancia un objeto de la clase `AyudanteHibrido`. En base al diagrama anterior y sus conocimientos de OOP en Python, escriba en orden **los *prints* que deberían aparecer en consola** de los métodos inicializadores de clases respectivas, comenzando desde `AyudanteHibrido` e incluyendo su print. Puede ignorar los saltos de línea.

2. **(1 punto)** Actualmente existe un problema en los inicializadores del código incluido en el diagrama. Indique cuál es dicho problema, de qué se trata, y cuál sería la solución adecuada para este caso en particular.

3. **(1 punto)** ¿Cuál de las clases del diagrama deberían ser clases abstractas y cuáles no? Argumente para cada posible clase abstracta que identifique.

4. **(2 puntos)** Usando el diagrama anterior, suponga que se quiere agregar el atributo `hambre`. Este atributo debe ser un número entero, entre 0 y 100, que corresponde al porcentaje actual de hambre de su instancia. Indique, de forma justificada, la clase donde sería más adecuado incorporar este nuevo atributo, y describa con palabras, usando conceptos del curso, cuál sería la mejor forma de programarlo.

Pregunta 2 - Análisis de código

El siguiente código corresponde a la modelación de un reproductor de música utilizando la materia de estructuras de datos. Lamentablemente, el código tiene dos errores que impiden su ejecución correcta. Primero, la estructura de datos `self.cola` no se encuentra definida. Segundo, una de las otras dos estructuras no se está utilizando correctamente.

```python
import collections


Cancion = collections.namedtuple('Cancion', ['nombre', 'artista'])


class DCCancionero:
    def __init__(self) -> None:
        self.cola = None
        self.historial = list()
        self.favoritos = set()

    def agregar_a_la_cola(self, cancion: Cancion) -> None:
        self.cola.append(cancion)
        print(f'Se agregó {cancion.nombre} de {cancion.artista} a la cola')

    def escuchar_siguiente_cancion(self) -> None:
        siguiente = self.cola.popleft()
        self.historial.append(siguiente)
        print(f'Escuchando {siguiente.nombre} de {siguiente.artista}')

    def agregar_favorito(self, cancion: Cancion) -> None:
        self.favoritos.append(cancion)
        print(f'Se agregó {cancion.nombre} de {cancion.artista} a favoritos')

    def volver_a_escuchar(self) -> None:
        cancion = self.historial.pop()
        self.cola.append(cancion)
        print(f'Se agregó {cancion[0]} de {cancion[1]} a la cola')


if __name__ == '__main__':
    cancionero = DCCancionero()
    cancionero.agregar_a_la_cola(Cancion('Fellas in London', 'Ye y JeySi'))
    cancionero.escuchar_siguiente_cancion()
    cancionero.volver_a_escuchar()

    cancion_favorita = Cancion('Might give you up', 'Rick Roller')
    cancionero.agregar_favorito(cancion_favorita)
    cancionero.agregar_favorito(cancion_favorita)
```

Indique la veracidad o falsedad de las siguientes afirmaciones. Argumente, en máximo 4 líneas, cada respuesta **aplicando correctamente los términos de estructuras de datos y relacionando la argumentación realizada con el código entregado**. Solo se permite referenciar el código presentado, es decir, **no se puede asumir** que existen más funciones, clases o variables fuera de las contenidas en el código anterior.

A continuación se presenta una afirmación con su respuesta y una argumentación acorde a la instrucción del párrafo anterior.

0. La variable `cancion_favorita` corresponde a una Cancion correctamente instanciada.

(V)/ F

En la línea 3 del código se define la `namedtuple` "`Cancion`", con dos campos: `'nombre'` y `artista`; y con el nombre de tipo `'Cancion'`. Dado que en la línea 37 se está instanciando `namedtuple` con dos argumentos, se creará correctamente una instancia de tipo `Cancion` con nombre `'Might give you up'` y artista `'Rick Roller'`.

Finalmente, las afirmación que deberás evaluar y argumentar son:

1. **(1.5 puntos)** Para que el método `agregar_a_la_cola` funcione correctamente, el atributo `self.cola` debe ser un `deque`.

2. **(1.5 puntos)** El atributo `self.historial` corresponde a un *stack*.

3. **(1.5 puntos)** Suponga que `self.cola` es una estructura de datos apropiada. Al final de la ejecución del programa, la canción `'Might give you up'` se encontrará dos veces en favoritos.

4. **(1.5 puntos)** Suponga que `self.cola` es una estructura de datos apropiada y que se soluciona el segundo error del código. Las líneas 34 y 36, correspondientes a `agregar_a_la_cola` y `volver_a_escuchar`, imprimirán lo mismo en consola.