Guía de Ejercicios 9: Clases y objetos

Objetivos:

- Incorporar la conveniencia de definir clases que encapsulen entidades de los problemas que queremos resolver, con sus datos y operaciones específicos.
- Apreciar la claridad y la simpleza que pueden ganar los programas cuando se usan clases.
- Tener un primer acercamiento al diseño de clases como herramienta para mejorar la complejidad temporal de los programas.

Ejercicio 1. Definir la clase `Punto`, para representar puntos en el plano, que tenga al menos los siguientes métodos:

(a) `Punto(x:float, y:float)`: construye e inicializa un nuevo objeto de la clase `Punto` (para esto, definir el método `__init__`).

(b) `p.distancia_a(q:Punto)`: devuelve la distancia euclidiana entre los puntos `p` y `q`.

(c) `p.distancia_al_origen()`: devuelve la distancia euclidiana del punto `p` al origen.

(d) `p + q`: devuelve un nuevo objeto de la clase `Punto`, resultado de sumar los puntos `p` y `q`, componente a componente (para esto, definir el método `__add__`).

(e) `p == q`: determina si los puntos `p` y `q` son iguales, componente a componente (para esto, definir el método `__eq__`).

(f) Representación `str` de un punto; ejemplo: `"(1.23, -2.00)"` (para esto, definir `__repr__`).

Ejercicio 2.

(a) Definir la clase `Dado`, para representar un dado con algún número de caras, que tenga al menos los siguientes atributos y métodos:

(I) `Dado(caras:int)`: construye e inicializa un nuevo objeto de la clase `Dado`, con la cantidad de caras indicada (mayor o igual que 2).

(II) `d.valor_actual`: atributo entero (entre 1 y la cantidad de caras del dado) correspondiente a la cara superior del dado en este momento.

(III) `d.tirar()`: método que simula una tirada del dado. Elige un número al azar entre 1 y la cantidad de caras, y deja el valor obtenido en el atributo `valor_actual`. Para esto, usar la función `random.choice(xs)`, que permite elegir un elemento al azar de la lista `xs` (requiere hacer `import random`).

(IV) Representación `str` de un dado: simplemente su `valor_actual` como string.

Ejemplo de uso:

```
d:Dado = Dado(6)        # Crea un dado con 6 caras.
d.tirar()               # Tira el dado.
print(d.valor_actual)   # Imprime su valor actual (por ejemplo, 3)
d.tirar()               # Tira el dado.
print(d.valor_actual)   # Imprime su valor actual (por ejemplo, 5)
```

(b) Definir la clase `CubileteDeGenerala`, para representar los 5 dados de 6 caras necesarios para jugar a la Generala. Esta clase debe tener al menos los siguientes atributos y métodos:

(I) `CubileteDeGenerala()`: construye e inicializa un nuevo objeto `CubileteDeGenerala`.

(II) `p.dados`: atributo de tipo lista, con 5 dados de 6 caras.

(III) `p.tirar_todos()`: hace una tirada que incluye a todos los dados (es decir, tira cada uno de los 5 dados).

(IV) Representación `str` de un `CubileteDeGenerala`, que muestre los valores actuales de los 5 dados.

(c) Definir una función `generala()`, que interactúe con el usuario/a para simular un turno de la Generala: 1) comenzar con un `CubileteDeGenerala`; 2) tirar todos los dados y mostrar sus valores; 3) preguntar al usuario/a (con la instrucción `input`) cuáles dados quiere volver a tirar (numerados del 1 al 5 y separados por comas); 4) tirar esos dados y mostrar sus valores; 5) repetir una vez los pasos 3 y 4; 6) fin del turno.

Ejercicio 3. Considerar la definición de las clases `Naipe` y `Mazo` (de la baraja española) disponible en el archivo adjunto `naipes.py`

(a) Sea el siguiente código:

```
1  from naipes import Naipe, Mazo
2
3  mazo:Mazo = Mazo()
4  mazo.agregar(Naipe(7,'oros'))
5  mazo.agregar(Naipe(1,'espadas'))
6  mazo.agregar(Naipe(12,'copas'))
7  mazo.agregar(Naipe(4,'bastos'))
8  mazo.ordenar()
9
10 print(mazo)
```

Entre las líneas 3 y 8 se construye una mazo con 4 naipes y se lo ordena de menor a mayor. La línea 10 imprime como resultado `[4 de bastos, 12 de copas, 1 de espadas, 7 de oros]`, pero el resultado deseado hubiera sido `[7 de oros, 12 de copas, 1 de espadas, 4 de bastos]`. Encontrar el error (en `naipes.py`) y corregirlo.

(b) Demostrar que el método `naipe_mas_alto` tiene complejidad lineal en la cantidad de cartas del mazo. Luego, modificar la clase `Mazo`, de tal manera que el método `naipe_mas_alto` sea O(1), y el resto de los métodos no empeoren su orden de complejidad algorítmica.

Ejercicio 4. Definir una clase `ConjuntoDePuntos` que encapsule un conjunto de puntos bidimensionales representados por la clase `Punto` definida en el Ejercicio 1. Debe ofrecer un método que devuelva el **centro del conjunto**, definido como un nuevo punto con componentes `x` e `y` iguales al promedio de dichas componentes sobre todos los puntos del conjunto. Por ejemplo, si el conjunto tiene los puntos (1.0, 5.0), (3.0, -5.0) y (0.5, 9.9), entonces su centro es (1.5, 3.3). Este método debe tener **complejidad algorítmica constante**. La clase debe tener además un método de inicialización y un método para agregar un punto al conjunto.

Ejercicio 5.

(a) Definir la clase `Tateti`, que provea métodos para crear un juego, para determinar de quién es el turno, para hacer una jugada (cruz o círculo), para determinar si el juego terminó, y (cuando el juego terminó) para determinar quién ganó o si hubo empate. Por simplicidad, suponer que siempre empieza jugando la cruz.

Sugerencia: Representar al tablero con una lista de listas de strings (tres listas de longitud tres), con valores posibles `'vacío'`, `'cruz'` o `'círculo'`.

(b) Escribir un programa que permita al usuario jugar interactivamente al ta-te-tí contra la computadora, mostrando por pantalla el tablero y preguntando la siguiente movida mediante la instrucción `input`. Las jugadas de la computadora pueden realizarse al azar.

Sugerencias: Definir el método `__repr__` de `Tateti` para imprimir el tablero por pantalla. Las jugadas al azar de la computadora pueden lograrse con la función `random.choice(xs)` presentada en el Ejercicio 2.

Ejercicio 6. El archivo adjunto `provincias.csv` tiene información demográfica de las 24 provincias argentinas (considerando a la Ciudad de Buenos Aires como una provincia, por simplicidad).

(a) Definir una clase `Provincia` que encapsule los datos de una provincia, y que cuente con un método para comparar (por menor) según la densidad de población (población / área).

(b) Escribir (fuera de la clase `Provincia`) una función que lea el contenido de `provincias.csv` en una lista de objetos de tipo `Provincia`. Usar el módulo `csv` para la lectura del archivo.

(c) Imprimir por pantalla los nombres de las provincias, ordenadas de mayor a menor según su densidad de población.