

## Ejercicio 1

Crea una clase Pet que represente una mascota registrada en una clínica veterinaria.

La clase tendrá atributos privados para el nombre, especie y edad. Debes validar en el constructor que el nombre no sea vacío, que la especie sea una cadena de texto válida y que la edad sea un número entero mayor que 0. Si algún dato es incorrecto, lanza un ValueError con un mensaje adecuado.

- Implementa propiedades para poder modificar la edad y el nombre, aplicando las mismas validaciones.
- Agrega un método llamado vacunar() que marque internamente a la mascota como “vacunada”. Este método SOLO se puede llamar de forma interna en la clase.
- Implementa \_\_str\_\_ para mostrar el estado de la mascota, su nombre, especie, edad y si está vacunada o no.
- En el programa principal, crea varias mascotas, prueba a modificar datos válidos e inválidos y muestra los errores capturados con try/except.

## Ejercicio 2

Diseña una clase Participant que gestione los datos de una persona inscrita a un evento educativo.

La clase tendrá atributos privados para el nombre, el email y un estado booleano que indica si ha pagado la inscripción.

- En el constructor, valida que el nombre no esté vacío y que el email contenga el carácter "@". Si alguno no cumple, lanza ValueError.
- Implementa propiedades para permitir modificar el email con las mismas validaciones.
- Agrega los métodos pay() para marcar al participante como pagado, y cancel() para revertirlo.
- El método \_\_str\_\_ debe mostrar el nombre, email y si el participante ha completado el pago.

En el programa principal, prueba varios participantes válidos e inválidos, capturando excepciones. Los datos pueden ser estáticos o dinámicos (introducidos manualmente).

## Ejercicio 3

Crea una clase `Movie` que represente una película guardada en una lista de favoritos. La clase tendrá los atributos privados `título`, `género` y `puntuación`.

El constructor debe validar:

- El título no puede estar vacío.
- El género debe ser uno de: "acción", "comedia", "drama", "terror".
- La puntuación debe ser un número flotante entre 0 y 10.
- Si alguien introduce datos incorrectos, lanza el `ValueError` correspondiente.
- Implementa propiedades para poder modificar los atributos con las mismas validaciones y `__str__` para mostrar la película en una línea.

En el programa principal, crea una lista de varias películas y utiliza **`filter()`** para obtener solo las que tengan una puntuación igual o superior a 7, mostrando el resultado.

Captura todas las excepciones que se generen al crear películas inválidas.

## Ejercicio 4

Diseña una clase `Employee` que tenga los atributos privados `nombre`, `horas trabajadas` y `precio por hora`.

El constructor debe validar:

- El nombre no puede ser numérico ni estar vacío.
- Las horas deben ser un valor entero entre 1 y 200.
- El precio por hora debe ser mayor que 5.
- Si algo no es válido, lanza `ValueError`.
- Implementa una propiedad para modificar las horas con validación.
- El `__str__` mostrará el nombre y lo que ha ganado ese mes.

En el programa principal:

- Crea varios empleados.
- Usa `map()` para transformar la lista de empleados en una lista con sus salarios finales del mes.
- Muestra los resultados.

Gestiona los errores con `try/except` si algún empleado no es válido.

## Ejercicio 5

Crea una clase Food que represente un producto alimenticio con atributos privados nombre, calorías y cantidad (en gramos).

Validaciones:

- El nombre debe ser texto no vacío.
- Las calorías deben ser un número positivo.
- La cantidad debe ser un valor entero entre 10 y 500.
- Implementa propiedades para modificar la cantidad y las calorías con validación y “\_\_str\_\_” para mostrar la información del alimento.

En el programa principal, crea una lista de alimentos y usa reduce() (de functools) para calcular:

- El total de calorías consumidas sumando  $(\text{calorías} * \text{cantidad} / 100)$  de cada alimento.

Gestiona todas las excepciones necesarias (por ejemplo si algún alimento no es válido).

## Ejercicio 6: Ejercicio de examen

Vas a desarrollar un pequeño sistema que gestione usuarios y registre sus actividades dentro de una plataforma digital.

Debes implementar dos clases: User y Activity, trabajar con listas de objetos y emplear validaciones con raise, además de usar filter, map o reduce en alguna parte del programa principal.

### 1. Clase User

Cada objeto representará un usuario de la plataforma. Debe contener atributos privados:

- `__username`
- `__email`
- `__age`
- `__active` (booleano)

Validaciones obligatorias:

- El nombre de usuario no puede estar vacío, ni contener espacios al inicio o final, y no puede ser numérico.
- El email debe contener "@" y "."; si no, lanza `ValueError("Email inválido")`.
- La edad debe ser un valor entero entre 14 y 120.
- `__active` será `True` por defecto.

Propiedades obligatorias:

- `email` con setter validado.
- `age` con setter validado.

Métodos obligatorios:

- `deactivate()` que ponga `__active` a `False`.
- `activate()` que ponga `__active` a `True`.
- `__str__()` que muestre: "Usuario: Ana (22 años) — Activo"

### Clase Activity

Cada objeto representará una actividad realizada por un usuario dentro de la plataforma.

Atributos privados:

- `__user` (objeto User)
- `__type` (ej: "login", "compra", "comentario")
- `__duration` (duración en minutos)

#### Validaciones:

- user debe ser obligatoriamente un objeto de la clase User (si no, lanza ValueError("Usuario inválido")).
- El tipo debe ser uno de: "login", "compra", "comentario", "visita".
- La duración debe ser un número mayor que 0.

#### Propiedades:

- duration con setter validado.

#### Método obligatorio:

- \_\_str\_\_() que muestre la información así:
  - "Actividad: login — Usuario: Ana — Duración: 5 min"

#### Debes:

- a) Crear varios usuarios, algunos válidos y otros inválidos para comprobar que las validaciones funcionen. Usa try/except y muestra los errores.
- b) Guardar los usuarios válidos en una lista users.
- c) Crear varias actividades, vinculadas a los usuarios creados. Asegúrate de probar actividades inválidas (duraciones incorrectas, tipos no permitidos, usuario no válido, etc.)
- d) Guardar las actividades válidas en una lista activities.

#### Herramientas obligatorias:

En algún punto del programa debes usar de forma obligatoria:

- filter() para obtener todas las actividades de usuarios activos.
- map() para obtener solo la duración de cada actividad (lista de enteros).
- reduce() (de functools) para calcular el total de minutos registrados en actividades.