Práctica 9: Listas

Índice

Objetivos y planificación

Objetivos

- Creación de listas
- Inserción de elementos
- Paso de listas como argumentos
- Índices e iteración
- Asignación
- Métodos básicos
- Slicing

Planificación

Desarrollaremos:

- Una aplicación para el manejo de cartas.

Abstracción

Trabajaremos con una baraja simplificada.

- Solo 4 tipos de carta (no hay números)


Corazones (hearts)


Picas (spades)

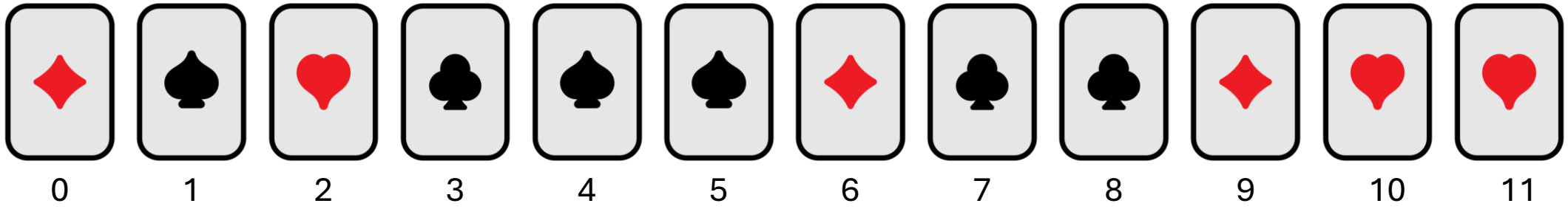

Diamantes (diamonds)


Tréboles (clubs)

Abstracción

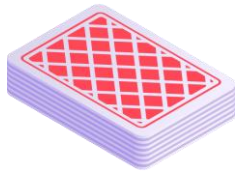
Las cartas se organizarán en secuencia (tienen orden).

Podemos tener cartas repetidas.



Abstracción

Una secuencia podría representar varios conceptos.



El mazo de cartas

La mano del jugador

Las cartas del tablero

Abstracción

Cualquier programa que gestione cartas necesita:

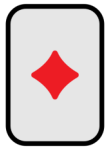
- Crear secuencias de cartas.
- Añadir y/o eliminar cartas de la secuencia.
- Mover cartas de posición.
- Recorrer las cartas de la secuencia.
  - Para mostrar, contar, comparar, etc.

Abstracción

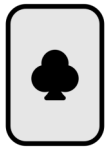
Representaremos los tipos de carta con strings.

`"hearts"`

`"spades"`



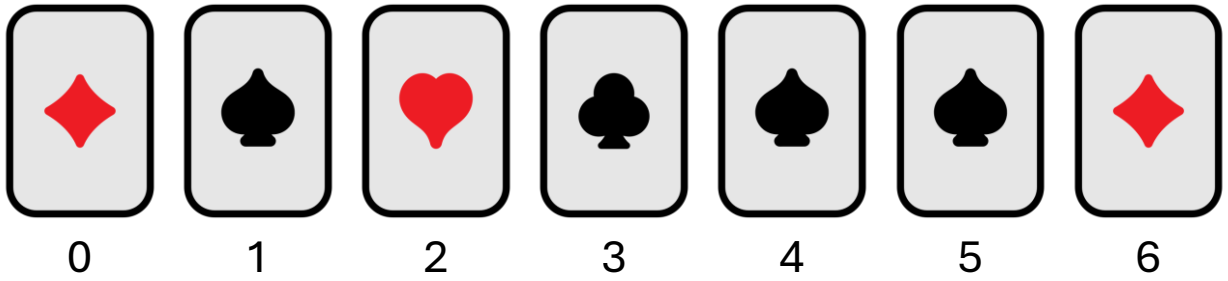
`"diamonds"`



`"clubs"`

Abstracción

Representaremos una secuencia de cartas con una lista.

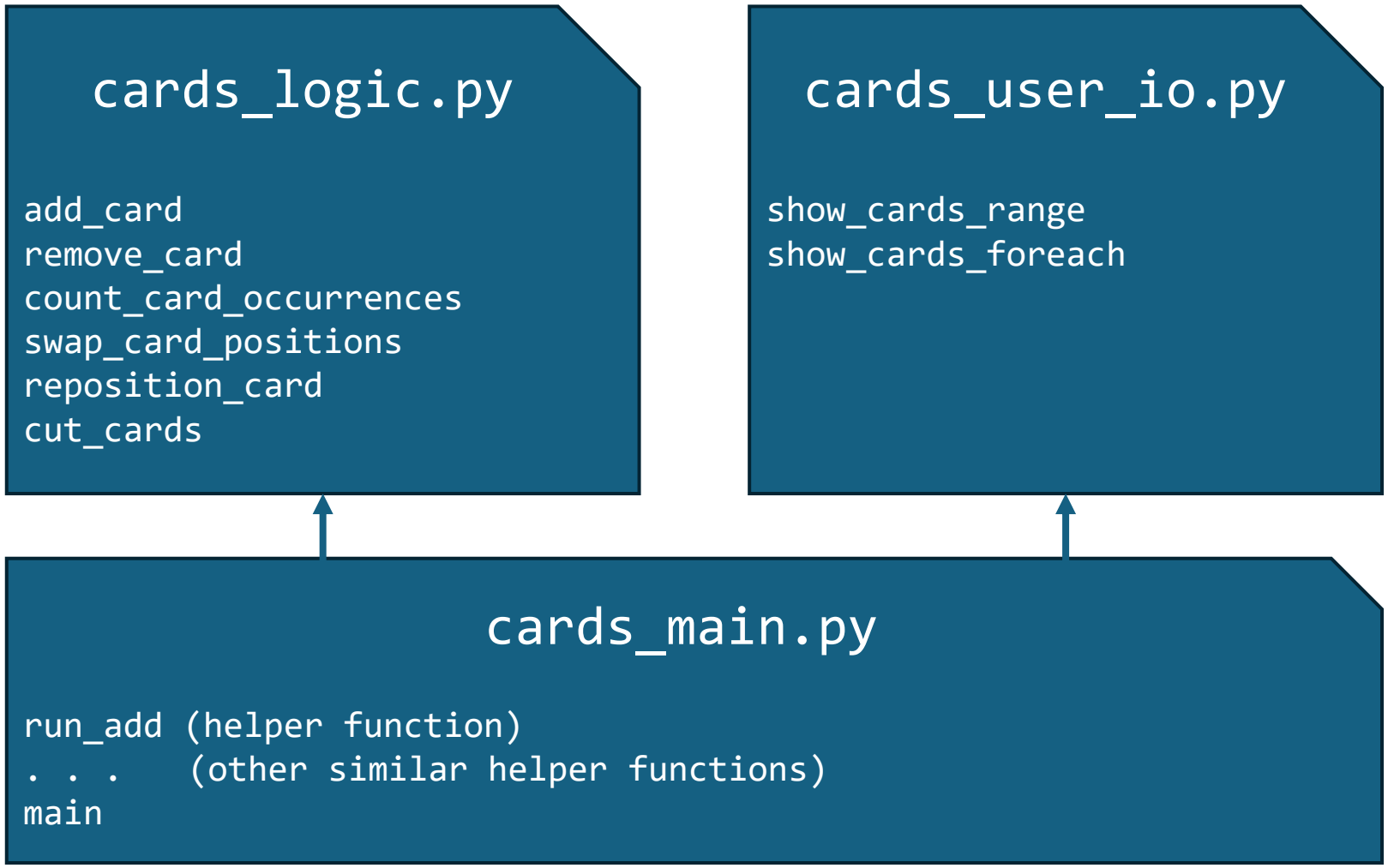



```
card_list = ["diamonds", "spades", "hearts", "clubs", "spades", "spades", "diamonds"]
             0           1         2          3         4          5          6
```

Planificación

- Comenzaremos haciendo pruebas básicas en el módulo main.
- Crearemos un módulo con funciones para manipular cartas.
- Crearemos un módulo de entrada salida para mostrar información de las cartas.
- Iremos creando y manipulando una baraja en el módulo main (sin menú)

Organización del código



Funcionamiento básico

Módulo main

- Crea un módulo `cards_main.py` con su función `main`.
- Probaremos rápidamente las funcionalidades de las listas antes de comenzar a organizar el código en funciones.

Funcionalidades básicas

Crea una función `run_basic_functionalities` que haga lo siguiente:

1. Crea una lista `card_list` con 5 cartas.
2. Utiliza la función `print` para imprimir la lista.
3. Añade una carta al final de la lista utilizando el método `append` e imprímela.
4. Crea una variable `other_card_list` y asígnale la lista original `card_list` e imprime ambas.
5. Crea una variable `card` y asígnale la carta de la posición 3 de `card_list`.
6. Asigna el valor de la variable `card` a la posición 0 de `other_card_list`. Imprime ambas listas ¿por qué ambas muestran ese resultado?
7. Crea una lista vacía, inserta las cartas de las posiciones 2 y 5 de `other_card_list`, e imprime todas las listas.
8. Utiliza la función `len` para mostrar el número de elementos de cada lista.

Añadir cartas

Añadir carta

Crea el modulo `cards_logic.py`

Crea en él la función `add_card`, que añade una carta a una lista.

- ¿Qué argumentos debería recibir?
- ¿Qué debería devolver?

```python
def add_card(???):
    """Documentation"""
    # Code...
    return ???
```

Utilizando add_card

En el modulo main, importa `cards_logic.py`, crea la función `run_add_card` y llámala desde el `main`.

```
def run_add_card_mutating(card_list, card_name):
    """Documentation"""
    print(f"card_list: {card_list}")
    returned_card_list = cards_logic.add_card(card_list, card_name)
    print(f"card_list: {card_list}")
    print(f"returned_card_list: {returned_card_list}")
```

Listas como argumentos

Las listas son objetos **mutables.**

Cuando pasamos una lista como argumento, si dentro de la función es mutada, mutará a la lista fuera de la función.

Listas como argumentos

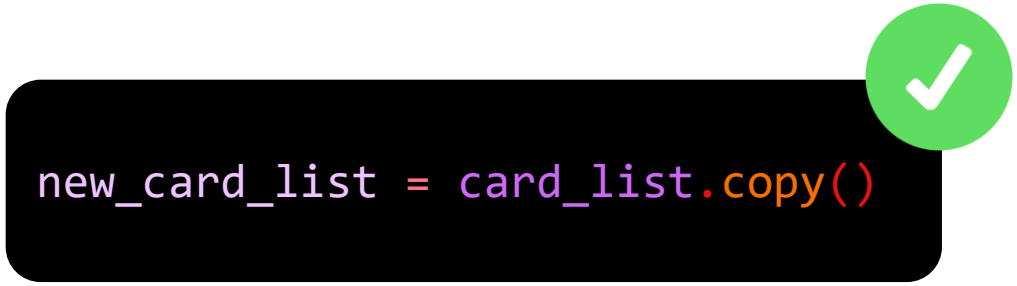
Debemos evitar que las funciones muten los argumentos que reciben.

La solución más sencilla es:

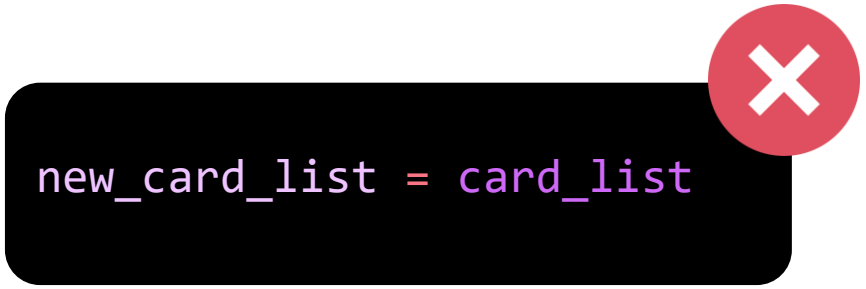
1. Copiar la lista que hemos recibido como argumento.
2. Mutar únicamente la copia.
3. Devolver la copia.

Copiar listas

El método `copy` de la clase lista crea una nueva copia de la lista y la devuelve.



```
new_card_list = card_list.copy()
```



```
new_card_list = card_list
```

¡Importante! La asignación no crea una nueva lista. Ambas variables apuntarían a la misma lista.

Listas como argumentos

Podemos verificar en los doctest que nuestra función no muta la lista que recibe.

```
Examples:
        >>> card_list = ["diamonds", "hearts"]
        >>> add_card (card_list, "hearts")
        ['diamonds', 'hearts', 'hearts']
        >>> card_list
        ['diamonds', 'hearts']
```

- Asignamos la lista a una variable.
- Pasamos la lista como argumento.
- Comprobamos que devuelve una lista actualizada.
- Usamos de nuevo la variable original.
- Comprobamos que no ha cambiado.

Funciones que devuelven nuevas listas

Cuando llamamos a funciones que devuelven nuevas listas podemos asignar el valor de retorno a una nueva variable. Así conservamos la lista original.

```
new_card_list = cards_logic.add_card(card_list, card_name)
```

Si no necesitamos conservar la lista original, podemos asignar el valor de retorno a la misma variable.

```
card_list = cards_logic.add_card(card_list, card_name)
```

Eliminar cartas

- Crea la función `remove_card` asegurándote que no muta la lista que recibe.
  - Recibe un tipo de carta y elimina la primera carta de ese tipo que encuentre (la que esté en el índice más bajo).
    - Puedes utilizar el método `remove` de la clase list.

Recorrer cartas

Iteración

Crea un modulo `cards_user_io.py` y crea en el la función `show_cards_range` .

Utiliza un bucle for, y las funciones `range` y `len` para recorrer todas las cartas de la lista e imprimirlas una a una.

Iteración

Crea ahora una función `show_cards_foreach`

Utiliza el bucle for de la siguiente manera:

- ¿Cuál es la diferencia?

```
def show_cards_foreach(card_list):
    for card in card_list:
        # Code
```

Iteración

Crea en `cards_logic.py` una función `count_card_occurrences` que cuente el número de veces que se repite un tipo de carta en la lista.

(Aunque en la clase list existe el método `count`, resuelve la función utilizando un bucle)

Intercambiar posiciones

Intercambiar posiciones

En el modulo `cards_logic` crea la función `swap_card_positions`.

- Recibe la lista y dos índices.
- Devuelve una nueva lista con posición de las cartas en dichos índices intercambiadas.

Reposicionar cartas

Reposicionar cartas

En el modulo `cards_logic.py` crea una función `reposition_card`.

- Recibe la lista, un tipo de carta, y un índice.
- Busca la primera carta de dicho tipo, y la recoloca en el índice proporcionado.
- Utiliza los métodos `index`, `pop` e `insert` de la clase list.

Cortar la baraja

Cortar la baraja

En el modulo `cards_logic.py` crea la función `cut_card_list_in_two`.

- Recibe una lista
- La corta en dos mitades, e intercambia sus posiciones.
- Utiliza el operador de slicing [:] para resolverla.

Trabajo en casa

Contar parejas

En el modulo `cards_logic.py` crea la función count_pairs

- Consideramos una pareja dos cartas del mismo tipo en posiciones consecutivas.
- Dos cartas solo pueden pertenecer a una sola pareja:
  - Tres cartas del mismo tipo consecutivas solo contarían como 1 pareja, no 2.
  - Cuatro cartas del mismo tipo serían 2 parejas, no 3.

Contar escaleras

En el modulo `cards_logic.py` crea la función `count_flush`.

- Cuenta el número de secuencias de 4 cartas distintas.
  - 4 cartas solo pueden pertenecer a una escalera.

Varios cortes

En el modulo `cards_logic.py` crea la función `cut_cards`

- Recibe la lista de cartas y un entero.
- Divide la lista en el número recibido, y reordena los cortes en orden inverso.

(Una vez funcione, refactoriza la función `cut_cards_in_two` para llamar a esta función)