

## 2. Práctica TAD - Parte 2

TUIA - Programación 2

2025 - 2C

### Pila

#### Ejercicio 1

Defina una clase `Pila` que implemente el TAD Pila utilizando listas de Python.

#### Ejercicio 2

Defina una clase `PilaEnlazada` que implemente el TAD Pila utilizando una estructura enlazada.

#### Ejercicio 3

Defina una clase `PilaConMaximo` que implemente las operaciones de Pila (`push(item)` y `pop()`) y el método `obtener_maximo()` que devuelva el elemento máximo de la Pila, sin sacarlo de la misma y que funcione en **tiempo constante**.

**Ayuda:** usar dos pilas, una para guardar los elementos y otra para guardar los máximos.

#### Ejercicio 4

Escriba una función `balanceado` que reciba una expresión matemática (en forma de string) y devuelve `True` si los paréntesis `()`, corchetes `[]` y llaves `{}` están correctamente balanceados, `False` en caso contrario.

```
>>> balanceado('(x+y)/2')
True
>>> balanceado('[8*4(x+y)]+[2/5]')
True
>>> balanceado('(x+y]/2')
False
>>> balanceado('1+)2(+3')
False
```

### Cola

#### Ejercicio 5

Defina una clase `Cola` que implemente el TAD Cola utilizando listas de Python.

## Ejercicio 6

Defina una clase `ColaEnlazada` que implemente el TAD Cola utilizando una estructura enlazada.

## Cola Generalizada.

### Ejercicio 7

Hace un montón de años había una viejísima sucursal del correo que tenía un cartel que decía “No se recibirán más de 5 cartas por persona”. Es decir, la gente entregaba sus cartas (hasta la cantidad permitida) y luego tenía que volver a hacer la cola si tenía más cartas para despachar.

Modelar una cola de correo generalizada, donde en la inicialización se indica la cantidad (no necesariamente 5) de cartas que se reciben por persona.

```
class Cliente:
    def __init__(self, nombre: str, cant_cartas: int = 1) -> None:
        self.nombre = nombre
        self.cant_cartas = cant_cartas

class ColaGeneralizada:
    """Implementar esta clase"""
    pass
```

Ejemplo de uso:

```
>>> correo = ColaGeneralizada()
>>> correo.push(Cliente("Ana", 1))
>>> correo.push(Cliente("Facu", 1))
>>> correo.push(Cliente("Seba", 2))
>>> correo.push(Cliente("Joe", 6))
>>> correo.push(Cliente("Pablo", 9))
>>> correo.push(Cliente("Ana", 1))
>>> correo.push(Cliente("Facu", 1))
>>> correo.push(Cliente("Seba", 2))
>>> while not correo.isEmpty():
...     correo.remove()
...
Atendido cliente Ana, despachadas 1 cartas
Atendido cliente Facu, despachadas 1 cartas
Atendido cliente Seba, despachadas 2 cartas
Atendido cliente Joe, despachadas 5 cartas
Atendido cliente Pablo, despachadas 5 cartas
Atendido cliente Ana, despachadas 1 cartas
Atendido cliente Facu, despachadas 1 cartas
Atendido cliente Seba, despachadas 2 cartas
Atendido cliente Joe, despachadas 1 cartas
Atendido cliente Pablo, despachadas 4 cartas
```

## Ejercicios adicionales

### Ejercicio 7

Dado un **Stack** de números, reordenarlos para que estén abajo los impares y arriba los pares, pero que entre números del mismo tipo preserven el orden.

**Ayuda:** utilizar dos **Stacks** auxiliares de números pares e impares respectivamente.

**Ejemplo:**

4	4
8	8
5 =>	2
9	5
2	9
1	1

### Ejercicio 8

Una implementación alternativa del TAD Cola puede implementarse internamente utilizando dos **Stacks** de la siguiente manera:

- Inserta por uno de los stacks.
- Remueve por el otro stack.
- Cuando queremos remover de stack vacío, primero volcamos el stack de inserción en este y seguimos normalmente.

Definir una clase **FastQueue** que realice esta implementación del TAD Cola.

Escribir código cliente para verificar qué tanto **FastQueue** como **Queue** se comportan efectivamente como Colas.

**Ayuda:** puede serle útil implementar un método extra **\_volcar**.

### Ejercicio 9

Escribir una clase **TorreDeControl** que modele el trabajo de una torre de control de un aeropuerto con una pista de aterrizaje. Los aviones que están esperando para aterrizar tienen prioridad sobre los que están esperando para despegar. La clase debe funcionar conforme al siguiente ejemplo:

```
>>> torre = TorreDeControl()
>>> torre.nuevo_arribo('AR156')
>>> torre.nueva_partida('KLM1267')
>>> torre.nuevo_arribo('AR32')
>>> torre.ver_estado()
Hay vuelos esperando para aterrizar.
Hay vuelos esperando para despegar.
>>> torre.asignar_pista()
El vuelo AR156 aterrizó con éxito.
>>> torre.asignar_pista()
El vuelo AR32 aterrizó con éxito.
>>> torre.asignar_pista()
El vuelo KLM1267 despegó con éxito.
>>> torre.asignar_pista()
No hay vuelos en espera.
```

## Ejercicio 10

Escribir las clases `Impresora` y `Oficina` que permita modelar el funcionamiento de un conjunto de impresoras conectadas en red.

Una impresora:

- Tiene un nombre, y una capacidad máxima de tinta.
- Permite encolar un documento para imprimir (recibiendo el nombre del documento).
- Permite imprimir el documento que está al frente de la cola.
  - Si no hay documentos encolados, se muestra un mensaje informándolo.
  - Si no hay tinta suficiente, se muestra un mensaje informándolo.
  - En caso contrario, se muestra el nombre del documento, y se gasta 1 unidad de tinta.
- Permite cargar el cartucho de tinta

Una oficina:

- Permite agregar una impresora
- Permite obtener una impresora por nombre.
- Permite quitar una impresora por nombre.
- Permite obtener la impresora que tenga menos documentos encolados.

Para facilitar el ejercicio, supondremos que todos los documentos a imprimir consumen la misma cantidad de tinta, es decir la cantidad de tinta la representaremos como un número entero y cada documento impreso la disminuye en uno. Al inicializar una impresora, esta siempre tiene la tinta al máximo.

**Ejemplo:**

```
>>> o = Oficina()
>>> o.agregar_impresora(Impresora('HP1234', 1))
>>> o.agregar_impresora(Impresora('Epson666', 5))
>>> o.impresora('HP1234').encolar('tp1.pdf')
>>> o.impresora('Epson666').encolar('tp2.pdf')
>>> o.impresora('HP1234').encolar('tp3.pdf')
>>> o.obtener_impresora_libre().encolar('tp4.pdf') # se encola en Epson666
>>> o.impresora('HP1234').imprimir()
tp1.pdf impreso
>>> o.impresora('HP1234').imprimir()
No tengo tinta :(
>>> o.impresora('HP1234').cargar_tinta()
>>> o.impresora('HP1234').imprimir()
tp3.pdf impreso
>>> o.impresora('HP1234').imprimir()
No hay nada en la cola
```

## Ejercicio 11

En la parada del colectivo 133 pueden ocurrir dos eventos diferentes:

- Llega una persona.
- Llega un colectivo con  $n$  asientos libres, y se suben al mismo todas las personas que están esperando, en orden de llegada, hasta que no quedan asientos libres. Asumimos que no pueden viajar personas de pie.

Cada evento se representa con una tupla que incluye:

- El instante de tiempo (cantidad de segundos desde el inicio del día).
- El tipo de evento, que puede ser **p** (persona) o **c** (colectivo).
- En el caso de un evento de tipo **c** hay un tercer elemento que es la cantidad de asientos libres.

Escribir una función que recibe una lista de eventos, ordenados cronológicamente, y devuelva el promedio de tiempo de espera de los pasajeros en la parada.

**Ejemplo:**

```
>>> promedio_espera([(35, 'p'), (43, 'p'), (80, 'c', 1), (98, 'p'), (142, 'c', 2)])
62.6667 # calculado como (45 + 99 + 44) / 3
```