

## Tipos de Datos Abstractos (TAD) en Python

### ¿Qué es un TAD?

Un **Tipo de Dato Abstracto (TAD)** es una *forma lógica de organizar datos y operaciones*, sin importar cómo se implementen por dentro. Es como una receta: no importa si hacés la torta con batidora o a mano, mientras salga rica 🍰.

### Ejemplo de la vida real:

Un TAD sería como un cajón. Vos sabés que podés:

- Guardar cosas
  - Sacar cosas
  - Ver qué hay adentro
- Pero no te importa si tiene un riel, un resorte o magia adentro. Lo importante es lo que *puede hacer*, no cómo lo hace.

---

### ¿Cómo se implementan en Python?

Python no tiene una "interfaz de TAD" como otros lenguajes, pero **¡todo se puede hacer con clases y estructuras!** Vamos a ver los más comunes:

---

### 1. Pila (Stack) – LIFO (Last In, First Out)

La última cosa que guardás, es la primera que sacás. Como una pila de platos 🍽️.

#### Operaciones:

- push(elemento)
- pop()
- peek() (opcional, para ver el tope sin sacarlo)

#### Implementación:

```
class Pila:
```

```
    def __init__(self):
```

```
        self.items = []
```

```
def esta_vacia(self):  
    return len(self.items) == 0
```


```
def apilar(self, item):  
    self.items.append(item)
```

```
def desapilar(self):  
    return self.items.pop()
```

```
def ver_tope(self):  
    return self.items[-1]
```

---

## 2. Cola (Queue) – FIFO (First In, First Out)

La primera persona en la cola del súper es la primera en pagar .

### Operaciones:

- enqueue(elemento)
- dequeue()

### Implementación:

```
from collections import deque
```

```
class Cola:
```


```
    def __init__(self):  
        self.items = deque()
```

```
    def esta_vacia(self):  
        return len(self.items) == 0
```

```
def encolar(self, item):  
    self.items.append(item)  
  
def desencolar(self):  
    return self.items.popleft()
```

---

### 3. Lista Enlazada (Linked List) – Conexión uno a uno

Cada nodo apunta al siguiente. Como un tren .

#### Operaciones:

- Agregar nodo
- Eliminar nodo
- Buscar nodo

#### Implementación:

```
class Nodo:
```

```
    def __init__(self, valor):  
        self.valor = valor  
        self.siguiente = None
```

```
class ListaEnlazada:
```

```
    def __init__(self):  
        self.cabeza = None
```

```
    def agregar_al_inicio(self, valor):  
        nuevo = Nodo(valor)  
        nuevo.siguiente = self.cabeza
```

```
self.cabeza = nuevo
```

```
def imprimir(self):  
  
    actual = self.cabeza  
  
    while actual:  
  
        print(actual.valor, end=" -> ")  
  
        actual = actual.siguiente  
  
    print("None")
```

---

#### 4. Diccionario como TAD Mapa

Asociás claves con valores. Como una guía telefónica .

##### Operaciones:

- Agregar (clave, valor)
- Buscar por clave
- Eliminar por clave

##### Implementación:



```
agenda = {}  
  
agenda["Ana"] = "123-456"  
  
agenda["Juan"] = "789-012"
```



```
print(agenda["Ana"]) # Imprime: 123-456
```

En este caso, Python **ya implementa** este TAD por vos usando dict.

---

#### ¿Por qué usar TAD?

-  **Organizan tu código** como un libro bien escrito.
-  **Evitás repetir código** innecesario.

-  **Ocultan detalles:** sabés *qué hacen*, no *cómo lo hacen*.
-  **Focalizás en la lógica**, no en lo técnico.

---

### **Conclusión**

Los TAD en Python son como superpoderes: te permiten construir programas más ordenados, reutilizables y elegantes. Con ellos podés manejar datos como un profesional 💪🐍.

"No importa si usás una lista, una cola o una pila... lo importante es saber cuándo usarlas."