UNIVERSIDAD NACIONAL
GUILLERMO BROWN

# CLASE 3 - Unidad 2

# Estructuras de Datos Recursivas

**ESTRUCTURAS DE DATOS (271)**

**Clase N. 3. Unidad 2.**

# Clase 3: AGENDA

## AGENDA

- **Temario:**
  - **Revisión de conceptos: Listas, Pilas, Colas, Arboles y Grafos.**
  - Representación y estrategias de implementación de cada estructura, accesos y recorridos.

- **Ejemplos** en Lenguajes **Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

# Clase 3: TEMARIO. Listas

## Listas:

Una **lista** es una colección lineal de elementos a los que podemos acceder de manera aleatoria, es decir podemos acceder a cualquier elemento de la lista.

Las listas pueden ser representas de dos maneras:

- Listas estáticas
- Listas dinámicas

Entonces cuál de los dos tipos de las dos implementaciones usaremos???

La implementación **dinámica** nos permitirá hacer una introducción a otros tipo de estructuras que veremos más adelante en la cursada.

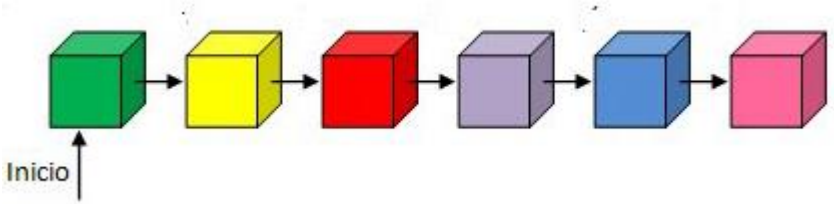
.

*Cap. VIII del Libro Algoritmos y Estructuras de Datos en Python*

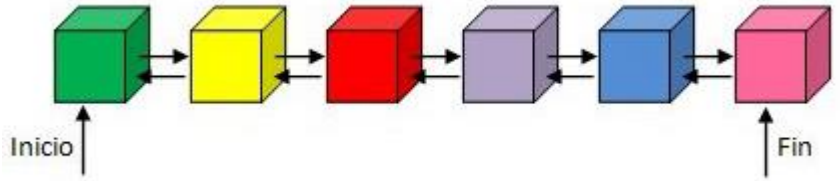
# Clase 3: TEMARIO. Listas

## Representaciones:

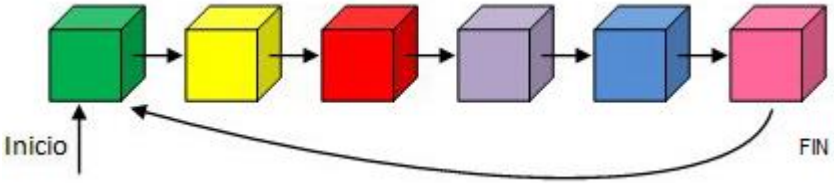
- Lista simplemente enlazada



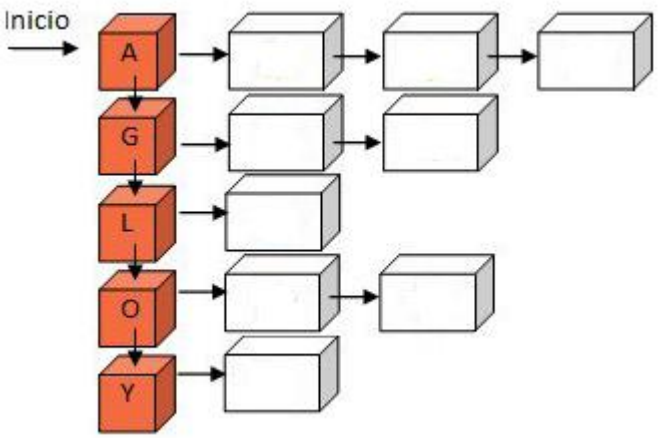

- Lista doblemente enlazada




- Lista circular




- Lista de lista



# Clase 3: TEMARIO. Listas

## Representación:

**inicio** es un puntero que apunta al nodo que esta al principio de la lista

**tamaño**: cantidad de elementos de la lista

```
#ESTRUCTURA DEL TAD LISTA
class Lista(object):
    #clase lista simplemente enlazada
    def __init__(self):
        #crea una lista vacía
        self.inicio = None
        self.tamaño = 0
```

**info**: contiene la información del nodo

**sig**: es una referencia al nodo siguiente

```
class nodoLista(object):
    #clase nodo lista
    info, sig = None, None
```

# Clase 3: TEMARIO. Listas



## Operaciones:

1. **insertar**(lista, elemento). Agrega el elemento a la lista de manera que el mismo quede ordenado;
2. **eliminar**(lista, clave). Elimina y devuelve de la lista si encuentra un elemento que coincida con la clave dada –el primero que encuentre–, si devuelve *None* significa que no se encontró la clave en la lista, y por ende no se elimina ningún elemento;
3. **lista_vacia**(lista). Devuelve verdadero (*true*) si la lista no contiene elementos;
4. **buscar**(lista, clave). Devuelve un puntero que apunta al nodo que contiene un elemento que coincida con la clave –el primero que encuentra–, si devuelve *None* significa que no se encontró la clave en la lista;
5. **tamaño**(lista). Devuelve la cantidad de elementos en la lista;
6. **recorrido**(lista). Realiza un recorrido de la lista mostrando la información de los elementos almacenado en la lista.

# Clase 3: TEMARIO. Listas



Implementación:

```
class nodoLista(object):
    #clase nodo lista
    info,sig = None,None

class lista(object):
    #clase lista
    def __init__(self):
        #crea una lista vacia
        self.inicio = None
        self.tamaño = 0
def insertar(lista,dato):
        #inserta el dato en la lista de manera ordenada
        nodo = nodoLista()
        nodo.info = dato
        if(lista.inicio is None) or (lista.inicio.info > dato):
            #el dato nuevo es menor que los elementos insertados en la lista
            nodo.sig = lista.inicio
            lista.inicio = nodo #actualizo el comienzo de la lista
        else:
            #recorrer la lista buscando donde insertar el elemento
            anterior = lista.inicio
            actual = lista.inicio.sig
            while(actual is not None) and (actual.info < dato):
                anterior = anterior.sig #anterior = actual
                actual = actual.sig
            anterior.sig = nodo
            nodo.sig = actual
        lista.tamaño = lista.tamaño+1 #actualizo la cantidad de elementos de la lista
```

1. ***insertar****(lista, elemento). Agrega el elemento a la lista de manera que el mismo quede ordenado;*

## Implementación:

```
def imprimir(lista):
    cantidad = 0
    actual = lista.inicio
    while(cantidad < lista.tamaño):
        print(actual.info)
        actual = actual.sig
        cantidad = cantidad +1

lista_enlazada = lista() #creamos la instancia
insertar(lista_enlazada,8) #insertamos un dato
insertar(lista_enlazada,4) #insertamos otro dato
insertar(lista_enlazada,2) #insertamos y otro dato mas
imprimir(lista_enlazada)
```

**insertar(lista, elemento).** *Probamos el código con varios datos y controlamos que los datos queden ordenados*

**Definimos una función que nos permita recorrer la lista e imprimir el contenido, cual debería ser el resultado por pantalla??**

# Implementación:

```
def eliminar(lista,valor):
    #si el valor se encuentra en la lista
    # elimina el elemento de la lista y lo retorna
    dato = None
    if(lista.inicio.info == valor):
        dato = lista.inicio.info
        lista.inicio = lista.inicio.sig
        lista.tamaño = lista.tamaño -1
    else:
         #recorremos la lista
         anterior = lista.inicio
         actual = lista.inicio.sig
         while(actual is not None) and (actual.info != valor):
             anterior = anterior.sig #anterior = actual
             actual = actual.sig
         if(actual is not None):
             #suponemos que el elemento puede no existir en la lista
             anterior.sig = actual.sig
             dato = actual.sig.info
             lista.tamaño = lista.tamaño -1
    return dato

lista_enlazada = lista() #creamos la instancia
insertar(lista_enlazada,8) #insertamos un dato
insertar(lista_enlazada,4) #insertamos otro dato
insertar(lista_enlazada,2) #insertamos y otro dato mas
imprimir(lista_enlazada)
eliminar(lista_enlazada,4)#eliminamos el segundo elemento de la lista
imprimir(lista_enlazada) #verificamos que la lista queda ordenada
```

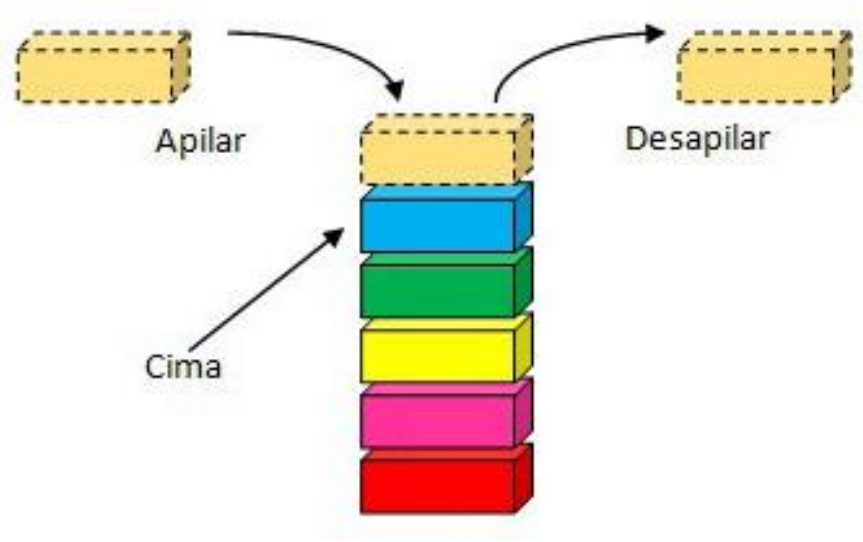
2. ***eliminar****(lista, clave). Elimina y devuelve de la lista si encuentra un elemento que coincida con la clave dada –el primero que encuentre–, si devuelve None significa que no se encontró la clave en la lista, y por ende no se elimina ningún elemento;*

# Clase 3: TEMARIO. Pilas

## Pilas:

Una pila es una colección de elementos que se agregan y se eliminan siguiendo el principio de **ultimo en entrar-primero en salir** (LIFO, *Last In-First Out*), es decir, el ultimo elemento insertado en la pila es el primero en ser eliminado.

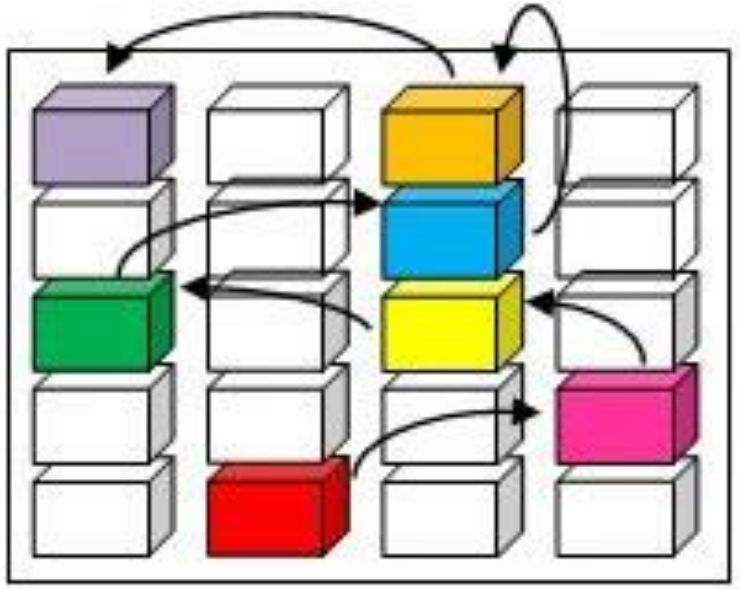
- Un elemento puede ser apilado en cualquier momento a una pila.
- Solo se puede desapilar(acceder) el elemento que este en la cima o tope de la pila,



# Clase 3: TEMARIO. Pilas

## Pilas representación:

Si usamos una implementación dinámica con nodos enlazados disponemos de toda la memoria para crear nodos, por lo cual si la misma se llenara el sistema operativo continuara utilizando memoria virtual (intercambiando la memoria a disco) extendiendo la memoria principal y por ende la capacidad de la pila.



# Clase 3: TEMARIO. Pilas

## TAD Pila:

**1. apilar(pila, elemento).** Agrega el elemento sobre la cima de la pila;

**2. desapilar(pila).** Elimina y devuelve el elemento almacenado en la cima de la pila;

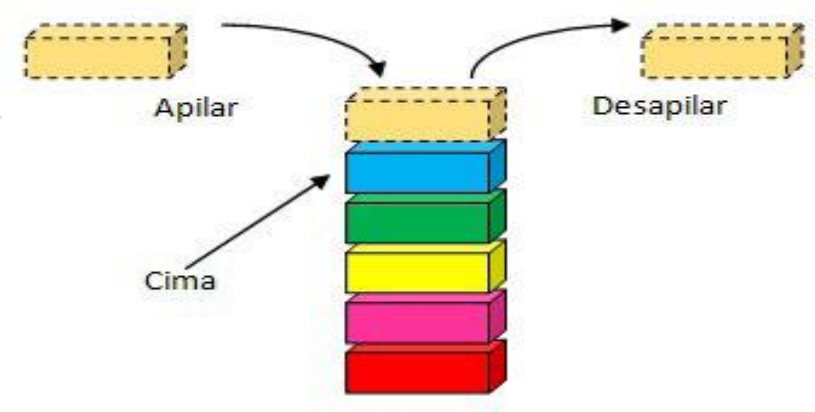
**3. pila_vacia(pila).** Devuelve verdadero  si la pila no contiene elementos;

**4. cima(pila).** Devuelve el valor del elemento que está almacenado en la cima de la pila pero sin eliminarlo;

**5. tamaño(pila).** Devuelve la cantidad de elementos en la pila.

# Clase 3: TEMARIO. Pilas

## Estructura de TAD Pila:




```
#estructura del TAD pila

class Nodo_pila(object):
    #clase nodo pila
    info = None
    siguiente = None

class Pila(object):
    #clase pila
    def __init__(self):
        #referencia al nodo que está en la cima de la pila
        self.cima = None
        self.tamaño = 0 #pila sin datos
```

# Clase 3: TEMARIO. Pilas

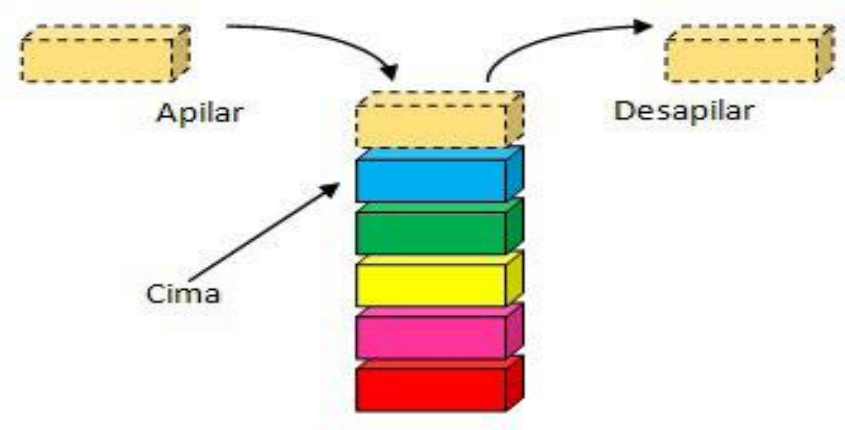
## Estructura de TAD Pila:

```
def apilar(pila,dato):
    #agrego el dato en la pila actualizando los punteros
    nodo = Nodo_pila() #creo el nodo
    nodo.info = dato #agrego al nodo los datos
    nodo.siguiente = pila.cima #actualizo el puntero
    pila.cima = nodo #actualizo la cima
    pila.tamaño = pila.tamaño+1

def desapilar(pila):
    #elimino el elemento de la cima de la pila
    nodo = pila.cima #guardo el elemento a sacar
    pila.cima = nodo.siguiente #actualizo el elemento de la cima
    pila.tamaño = pila.tamaño-1 #actualizo el tamaño
    return nodo

def imprimir_nodo(nodo):
    print(nodo.info)

#programa principal
mi_pila = Pila()
apilar(mi_pila,25)
apilar(mi_pila,15)
dato = desapilar(mi_pila)
imprimir_nodo(dato)
```

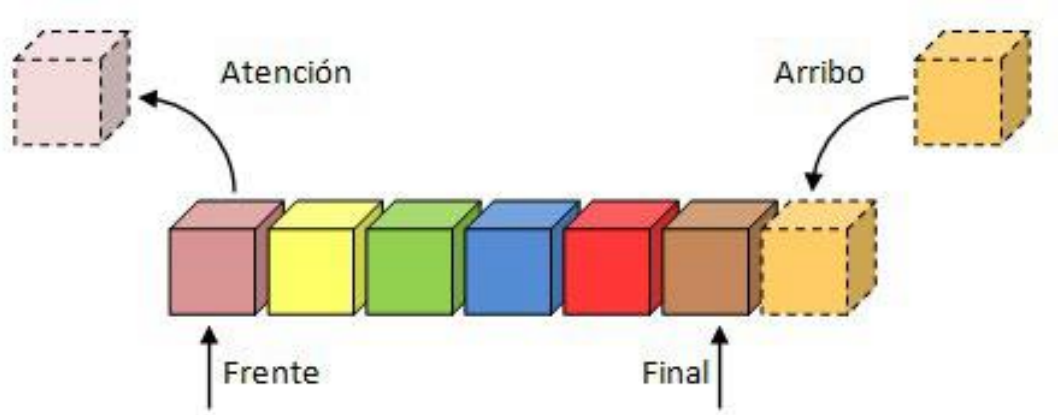


# Clase 3: TEMARIO. Colas

## Colas:

Una cola es una colección de elementos que se agregan y quitan basandose en el principio de primero en entrar-primero en salir (FIFO, *First In-First Out*).

Esto quiere decir que el primer elemento en ser *insertado* es el primero en ser *eliminado*.

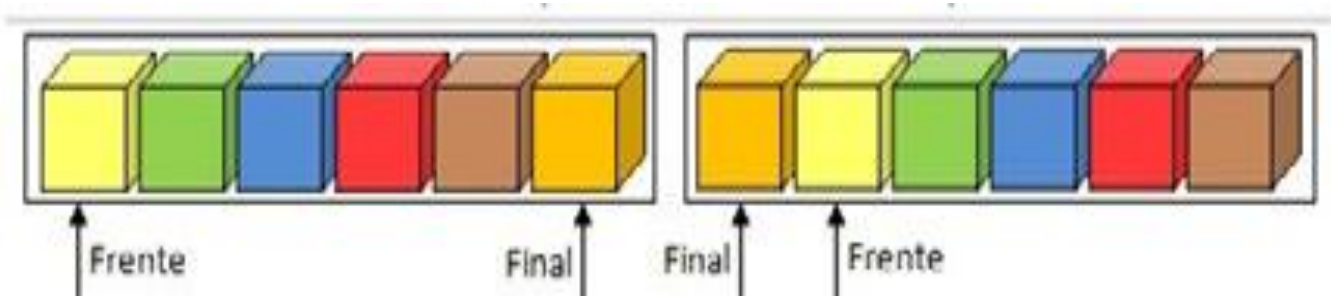


# Clase 3: TEMARIO. Colas

## Colas representación:

Usaremos nodos enlazados al igual que en el TAD pila.

## TAD Colas :




1. **push(cola, elemento).** Agrega el elemento a la final de la cola;
2. **pop(cola).** Elimina y devuelve el elemento almacenado en el frente de la cola;
3. **cola_vacia(cola).** Devuelve verdadero si la cola no contiene elementos;
4. **en_frente (cola).** Devuelve el valor del elemento que esta almacenado en el frente de la cola sin eliminarlo;
5. **tamano(cola).** Devuelve la cantidad de elementos en la cola;
6. **mover_al_final(cola).** Elimina el elemento en el frente de la cola y lo inserta en el final de la misma;

## Estructura del TAD Cola:

```
class NodoCola(object):
    #clase nodo cola
    info = None
    siguiente = None

class Cola(object):
    #clase cola
    def __init__(self):
        #crea una cola vacia
        self.frente = None
        self.final = None
        self.tamaño = 0
```

```
def push(cola,dato):
    #agrega el elemento en la cola
    nodo = NodoCola()#creo el nodo
    nodo.info = dato #agrego el dato
    if(cola.frente == None):#si la cola esta vacia
        cola.frente = nodo #actualizo
    else:
        cola.final.siguiente = nodo #actualizo el punterodel siguiente
    cola.final = nodo #ahora el nodo agregado nos indica el final
    cola.tamaño = cola.tamaño+1 #actualizo el tamaño de la cola
```

# Clase 3: TEMARIO. Colas



## Estructura del TAD Cola:

```
def pop(cola):
    #saco un elemento de la cola
    dato = cola.frente.info #saco el elemento
    cola.frente = cola.frente.siguiente #actualizo el frente
    if (cola.frente == None):
        cola.final = None #se quedo sin elementos
    cola.tamaño = cola.tamaño -1
    return dato

def imprimir_cola(cola):
    i = 0
    elemento = cola.frente
    while(i < cola.tamaño):
        print(elemento.info)
        elemento = elemento.siguiente
        i = i +1

#programa principal
mi_cola = Cola()
push(mi_cola,25)
push(mi_cola,30)
push(mi_cola,2)
push(mi_cola,78)
imprimir_cola(mi_cola)
dato = pop(mi_cola)
print("sin el elemento")
imprimir_cola(mi_cola)
```

# Clase 3: TEMARIO. Colas



# Clase 3: CIERRE DE LA CLASE

## Temas a desarrollar la próxima clase

- ❏ Árboles
- ❏ Grafos