



Taller 8

Fecha: Junio de 2021

Profesor: Fray León Osorio Rivera

Competencia a evaluar: Aplicar los conceptos básicos de la orientación a objetos y las estructuras de datos en el desarrollo de una aplicación con interfaz gráfica de usuario.

NOTAS:

- Este taller se debe hacer como preparación para examen o prácticas. En ningún caso representará una calificación.
- El primer ejercicio se entrega resuelto como ejemplo para el desarrollo de los demás.

Elaborar el diagrama de clases básico y la respectiva aplicación en el lenguaje *Python* para los siguientes enunciados:

1. Un palíndromo es una frase que se lee igual de izquierda a derecha y viceversa. Ejemplos:

- Amad a la dama
- Anita lava la tina
- Dábale arroz a la zorra el abad
- Somos o no somos
- Yo hago yoga hoy
- Roma amor

Elaborar una aplicación que dada una frase diga si es palíndromo o no.

R/

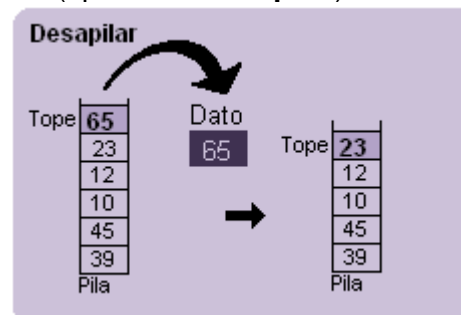
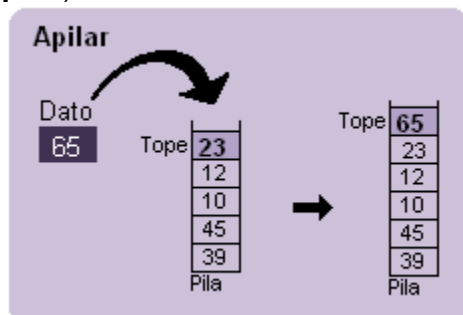
Para comprender este diagrama y el ejercicio, es importante tener en cuenta los siguientes fundamentos:

Concepto de Pila

Una **Pila** es una lista de elementos en la cual siempre por un extremo, denominado **Tope**, se pueden insertar nuevos elementos o retirar otros. Es una lista en la que el último que entra es el primero que sale (estructura LIFO).

Son dos los cambios básicos que se pueden hacer en una pila:

- Agregar un nuevo elemento (operación **Apilar**)
- Quitar el último elemento agregado (operación **Desapilar**)





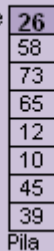
Para realizar estas operaciones existen algunas restricciones:

- No se puede *desapilar* si no hay elementos en la pila (condición **Pila Vacía**).
- En algunos casos, al apilar no se puede sobrepasar la capacidad máxima de la pila (condición **Pila Llena**)

Pila Vacía

Tope 
Pila

Pila Llena

Tope  Maximo
58
73
65
12
10
45
39
Pila

La siguiente es la descripción de la clase *Pila*:

Pila
apilar() desapilar(): Objeto vacía(): boolean valorTope(): Objeto

- El método *apilar()* permite apilar un dato en la pila.
- El método *desapilar()* permite desapilar un objeto de la pila
- El método *valorTope()* permite obtener el elemento que está en el tope de la pila (sin necesidad de desapilar)
- El método *vacía()* devuelve un valor booleano que indica si la pila está vacía

El siguiente sería el respectivo código en *Python*:

```
class Pila:
    #Metodo constructor
    def __init__(varClase):
        varClase.datos=[]

    def apilar(varClase, dato):
        varClase.datos.append(dato)

    def desapilar(varClase):
        dato=varClase.datos.pop()
        return dato

    def valorTope(varClase):
        if varClase.datos:
            return varClase.datos[-1] #retorna el ultimo elemento
        else:
            return None

    def vacia(varClase):
        if len(varClase.datos)==0:
            return True
        else:
            return False
```



Ahora bien, el código que resuelve el problema sería el siguiente:

```
from Pila import Pila

texto = input("Frase a validar?")

texto=texto.lower();

p=Pila()
#Quitar lo espacios en blanco
texto=texto.replace(" ", "")

mitad = int(len(texto)/2)

for i in range(0,mitad):
    p.apilar(texto[i]);

esPalindromo=True
i = mitad
if len(texto)% 2!=0:
    i += 1
while not p.vacia() and esPalindromo:
    caracter=p.desapilar()
    if texto[i] != caracter:
        esPalindromo=False;
    i += 1
if esPalindromo:
    print("La frase es palindromo")
else:
    print("La frase no es palindromo")
```

Cuya ejecución luciría así cuando la frase es un palíndromo:

```
Frase a validar?Anita lava la tina
La frase es un palíndromo
>>> |
```

Y así, cuando no lo es:

```
Frase a validar?Me gusta la programación
La frase NO es un palíndromo
>>>
```

2. Los archivos XML son archivos sin formato compuestos por un conjunto de datos y una serie de códigos denominados **Etiquetas** que sirven para describir qué significan dichos datos. El creador del archivo puede utilizar las etiquetas que se consideren necesarias, siguiendo un convencionalismo establecido. Es por ello que XML ha resultado un lenguaje muy útil y por la que se considera un lenguaje extensible.

Se podría pensar en almacenar la información de la colección musical personal en este tipo de archivos. Allí se incluirían los artistas y sus canciones.



```
<?xml version="1.0" encoding="UTF-8"?>
- <Artistas>
  - <Artista>
    <Nombre>Ekhymsys</Nombre>
    <Tipo>Grupo</Tipo>
    <Pais>Colombia</Pais>
  - <Cancion>
    <Titulo>Solo</Titulo>
    <Duracion>4:50</Duracion>
    <Año>1989</Año>
    <Genero>Balada</Genero>
  </Cancion>
  - <Cancion>
    <Titulo>La Tierra</Titulo>
    <Duracion>4:00</Duracion>
    <Año>1997</Año>
    <Genero>Pop</Genero>
  </Cancion>
</Artista>
- <Artista>
  <Nombre>Guns and Roses</Nombre>
  <Tipo>Grupo</Tipo>
  <Pais>USA</Pais>
  - <Cancion>
    <Titulo>November Rain</Titulo>
    <Duracion>8:57</Duracion>
    <Año>1989</Año>
    <Genero>Rock</Genero>
  </Cancion>
  - <Cancion>
    <Titulo>Dont cry</Titulo>
    <Duracion>4:45</Duracion>
    <Año>1989</Año>
    <Genero>Rock</Genero>
  </Cancion>
</Artista>
</Artistas>
```

Cada dato está rodeado por una combinación de etiquetas que juntas conforman un nodo. Cada etiqueta es un texto en medio de corchetes angulares. A cada etiqueta de apertura le corresponde una de cierre que tiene el mismo texto antecedido de un */slash*:

```
<Pais> ... </Pais>
```

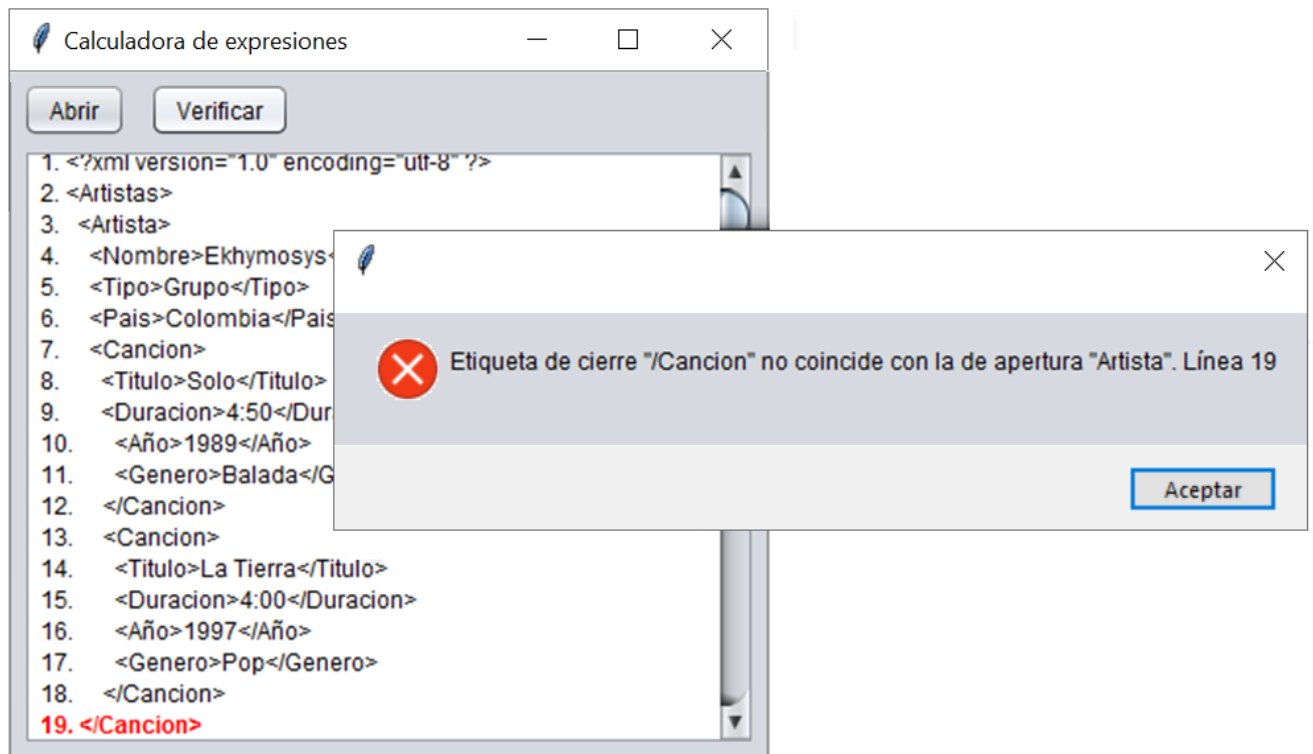
Un nodo puede contener a otros nodos que se pueden denominar subnodos:

```
<Cancion>
  < Titulo> ... </Titulo>
```



<Cancion>

Elaborar una aplicación que abra un archivo XML y valide si está bien configurado



Se debe validar:

- Que el archivo inicie con la etiqueta

```
<?xml version="1.0?>
```

- Que toda etiqueta de apertura tenga la respectiva etiqueta de cierre
- Que las etiquetas estén bien construidas