Introducción a la Programación

Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2023

Programación Imperativa: Otros TAD

Tipos Abstractos de Datos

Repasando

Un Tipo Abstracto de Datos (TAD) es un modelo que define valores y las operaciones que se pueden realizan sobre ellos.

- Se denomina abstracto ya que la intención es que quien lo utiliza, no necesita conocer los detalles de la representación interna o bien el cómo están implementadas sus operaciones.

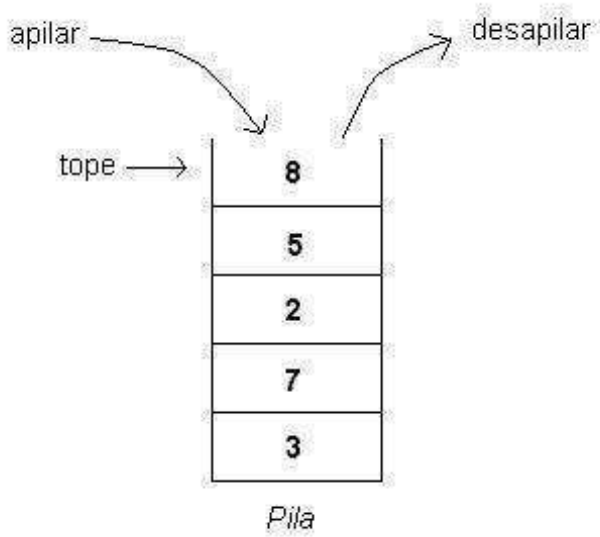
El tipo lista que estuvimos viendo es un TAD:

- Se define como una serie de elementos consecutivos
- Tiene diferentes operaciones asociadas: append, remove, etc
- Desconocemos cómo se usa/guarda la información almacenada dentro del tipo

Pila

Una pila es una lista de elementos de la cual se puede extraer el último elemento insertado.

- También se conocen como listas LIFO (Last In - First Out / el último que entra es el primero que sale)
- Operaciones básicas
  - apilar: ingresa un elemento a la pila
  - desapilar: saca el último elemento insertado
  - tope: devuelve (sin sacar) el ultimo elemento insertado
  - vacia: retorna verdadero si está vacía



Pila

- En Python, el tipo lista provee los métodos necesarios para poder usar una lista como una pila
- También, podemos importar el módulo LifoQueue que nos da una implementación de Pila

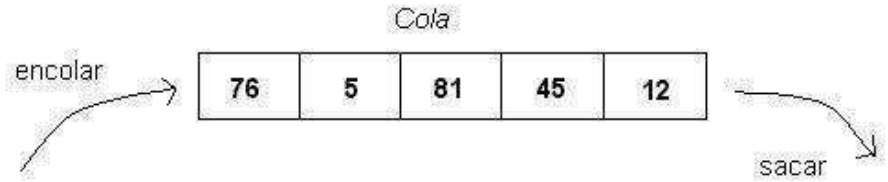
```
from queue import LifoQueue
pila = LifoQueue()
```

- Operaciones implementadas en el tipo:
  - apilar: ingresa un elemento a la pila
    - put
  - desapilar: devuelve y quita el último elemento insertado
    - get
  - tope: devuelve (sin sacar) el ultimo elemento insertado
    - No está implementado
  - vacia: retorna verdadero si está vacía
    - empty

Cola

Una cola es una lista de elementos en donde siempre se insertan nuevos elementos al final de la lista y se extraen elementos desde el inicio de la lista.

- También se conocen como listas FIFO (First In - First Out / el primero que entra es el primero que sale)
- Operaciones básicas
  - encolar: ingresa un elemento a la cola
  - sacar: saca el primer elemento insertado
  - vacia: retorna verdadero si está vacía



Cola

- En Python, el tipo lista provee los métodos necesarios para poder usar una lista como una cola
- También, podemos importar el módulo Queue que nos da una implementación de Cola

```
from queue import Queue
cola = Queue()
```

- Operaciones implementadas en el tipo:
  - encolar: ingresa un elemento a la pila
    - put
  - desencolar: saca el primer elemento insertado
    - get
  - vacia: retorna verdadero si está vacía
    - empty

Diccionario

Un diccionario es una estructura de datos que permite almacenar y organizar pares clave-valor.

- Las claves deben ser inmutables (como cadenas de texto, números, etc), mientras que los valores pueden ser de cualquier tipo de dato.
- La clave actúa como un identificador único para acceder a su valor correspondiente.
- Los diccionarios son mutables, lo que significa que se pueden modificar agregando, eliminando o actualizando elementos.
- No ordenados: Los elementos dentro de un diccionario no tienen un orden específico. No se garantiza que se mantenga el orden de inserción de los elementos.

diccionario = clave1:valor2, clave2:valor2, clave3:valor3

- Operaciones basicas de un diccionario:
  - Agregar un nuevo par Clave-Valor
  - Eliminar un elemento
  - Modificar el valor de un elemento
  - Verificar si existe una clave guardada
  - Obtener todas las claves
  - Obtener todas los elementos

Diccionario

Un diccionario es una estructura de datos que permite almacenar y organizar pares clave-valor.

- El valor puede ser cualquier tipo de dato, en particular podría ser otro diccionario

```
infoPaisFrancia = {'Capital':'París',
                   'Campeonatos de Mundo':2}

infoPaisArgentina = {'Capital':'Buenos Aires',
                     'Campeonatos de Mundo':3}

infoPaisChile = {'Capital':'Santiago',
                 'Campeonatos de Mundo':0}

infoPaises = {'Chile': infoPaisChile ,
              'Argentina': infoPaisArgentina,
              'Francia':infoPaisFrancia}
```

Manejo de Archivos

El manejo de archivos, también puede pensarse mediante la abstracción que nos brindan los TADs

- Necesitamos una operación que nos permita abrir un archivo
- Necesitamos una operación que nos permita leer sus lineas
- Necesitamos una operación que nos permita cerrar un archivo

```
# Abrir un archivo en modo lectura
archivo = open("archivo.txt", "r")

# Leer el contenido del archivo
contenido = archivo.read()
print(contenido)

# Cerrar el archivo
archivo.close()
```

Manejo de Archivos

archivo = open("PATH AL ARCHIVO", MODO, ENCODING)

- Algunos de los modos posibles son: escritura (w), lectura (r), texto (t - es el default)
- El encoding se refiere a como está codificado el archivo: UTF-8 o ASCII son los más frecuentes.

Operaciones básicas

- Lectura de contenido:
  - read(size): Lee y devuelve una cantidad específica de caracteres o bytes del archivo. Si no se especifica el tamaño, se lee el contenido completo.
  - readline(): Lee y devuelve la siguiente línea del archivo.
  - readlines(): Lee todas las líneas del archivo y las devuelve como una lista.
- Escritura de contenido:
  - write(texto): Escribe un texto en el archivo en la posición actual del puntero. Si el archivo ya contiene contenido, se sobrescribe.
  - writelines(lineas): Escribe una lista de líneas en el archivo. Cada línea debe terminar con un salto de línea explícito.

¿Podremos implementar este problema?

problema *invertirTexto*(in archivoOrigen: string, in archivoDestino: string) : {
  requiere: {El archivo nombreArchivo debe existir.}
  asegura: {Se crea un archivo llamado archivoDestino cuyo contenido será el resultado de hacer un reverse en cada una de sus filas}
  asegura: {Si el archivo archivoDestino existia, se borrará todo su contenido anterior}
}

Un paso más allá: ¿Qué es una API?

Un poquito fuera del alcance de la materia...

- El término API es muy usado actualmente y está relacionado con poder usar desde un programa funcionalidades de otro programa.
- API significa *Application Programming Interface* (Interfaz de Programación de Aplicaciones, en español). Una API define cómo las distintas partes de un software deben interactuar, especificando los métodos y formatos de datos que se utilizan para el intercambio de información.
- En el contexto de desarrollo de software, una API puede ser considerada como un contrato entre dos aplicaciones.
- Una API encapsula el comportamiento de otro programa y en muchos casos, su utilización es similar al uso de un TAD. Detrás de este encapsulamiento se esconden un gran números de problemas a resolver como ser: conexiones de red, uso de protocolos, manejo de errores, transformaciones de datos, etc (y son muchos etcs).

Veamos una API cualquiera: Google Translate API

Un paso más allá: ¿Qué es una API?

- Instalamos el módulo: pip install googletrans googletrans==3.1.0a0
- Y veamos que nos ofrece su contrato:

translator = Translator()

- El método translate(texto, idioma origen, idioma destino) devuelve la siguiente estructura :
  - src: idioma original dest: idioma destino
  - origin: texto en idioma original
  - text: texto traducido
  - pronunciation: pronunciación del texto tranducido

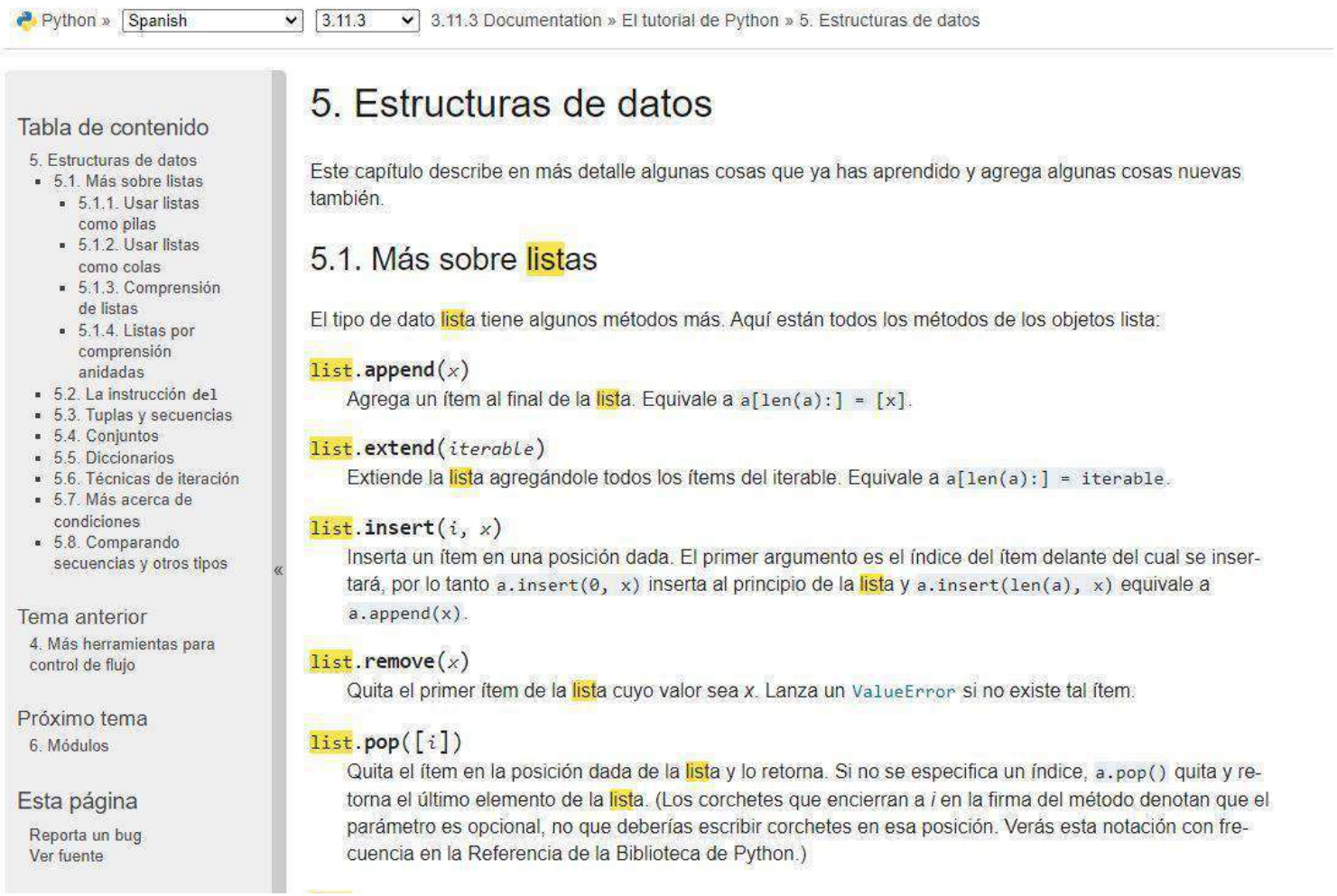
¿Podremos implementar este problema?

problema *traducirTexto*(in nombreArchivo: string, in idiomaOrigen: string, in idiomaDestino: string) : {
  requiere: {El archivo nombreArchivo debe existir.}
  asegura: {Se crea un archivo llamado *idiomaDestino* – *nombreArchivo* cuyo contenido será el resultado de traducir cada una de sus filas}
  asegura: {Si el archivo archivoDestino existia, se borrará todo su contenido anterior}
}

Documentación de Python

Python (como muchos de los otros lenguajes) tiene documentación pública con la descripción del comportamiento de sus distintos tipos de datos.

- https://docs.python.org/es/3/tutorial/datastructures.html?highlight=list



Python » Spanish 3.11.3 3.11.3 Documentation » El tutorial de Python » 5. Estructuras de datos

Tabla de contenido

Tema anterior

4. Más herramientas para control de flujo

Próximo tema

6. Módulos

Esta página

Reporta un bug
Ver fuente

5. Estructuras de datos

Este capítulo describe en más detalle algunas cosas que ya has aprendido y agrega algunas cosas nuevas también.

5.1. Más sobre listas

El tipo de dato lista tiene algunos métodos más. Aquí están todos los métodos de los objetos lista:

list.append(*x*)
Agrega un ítem al final de la lista. Equivale a `a[len(a):] = [x]`.

list.extend(*iterable*)
Extiende la lista agregándole todos los ítems del iterable. Equivale a `a[len(a):] = iterable`.

list.insert(*i*, *x*)
Inserta un ítem en una posición dada. El primer argumento es el índice del ítem delante del cual se insertará, por lo tanto `a.insert(0, x)` inserta al principio de la lista y `a.insert(len(a), x)` equivale a `a.append(x)`.

list.remove(*x*)
Quita el primer ítem de la lista cuyo valor sea *x*. Lanza un `ValueError` si no existe tal ítem.

list.pop([*i*])
Quita el ítem en la posición dada de la lista y lo retorna. Si no se especifica un índice, `a.pop()` quita y retorna el último elemento de la lista. (Los corchetes que encierran a *i* en la firma del método denotan que el parámetro es opcional, no que deberías escribir corchetes en esa posición. Verás esta notación con frecuencia en la Referencia de la Biblioteca de Python.)

Documentación de API Google Translate

La API de Google Translate tiene su propia página de especificación

- https://py-googletrans.readthedocs.io/en/latest/

API Guide

googletrans.Translator

class googletrans.**Translator**(*service_urls=None, user_agent='Mozilla/5.0 (Windows NT 10.0; Win64; x64)', raise_exception=False, proxies: Dict[str, httpcore._sync.base.SyncHTTPTransport] = None, timeout: httpx._config.Timeout = None, http2=True*) ¶

Google Translate ajax API implementation class

You have to create an instance of Translator to use this API

Parameters:
- **service_urls** (*a sequence of strings*) – google translate url list. URLs will be used randomly. For example `['translate.google.com', 'translate.google.co.kr']`
- **user_agent** (`str`) – the User-Agent header to send when making requests.
- **proxies** (*dictionary*) – proxies configuration. Dictionary mapping protocol or protocol and host to the URL of the proxy For example `{'http': 'foo.bar:3128', 'http://host.name': 'foo.bar:4012'}`
- **timeout** (*number or a double of numbers*) – Definition of timeout for httpx library. Will be used for every request.
- **proxies** – proxies configuration. Dictionary mapping protocol or protocol and host to the URL of the proxy For example `{'http': 'foo.bar:3128', 'http://host.name': 'foo.bar:4012'}`
- **raise_exception** (*boolean*) – if `True` then raise exception if smth will go wrong

translate(*text, dest='en', src='auto', **kwargs*)

Translate text from source language to destination language

Parameters:
- **text** (UTF-8 `str`; `unicode`; string sequence (list, tuple, iterator, generator)) – The source text(s) to be translated. Batch translation is supported via sequence input.
- **dest** – The language to translate the source text into. The value should be one of the language codes listed in `googletrans.LANGUAGES` or one of the language names listed in `googletrans.LANGCODES`.