Cátedra: Estructuras de Datos

Carrera: Licenciatura en Sistemas

Año: 2022 – 2do Cuatrimestre

Trabajo Práctico N.º 3:

Facultad de Ciencias de la Administración
Universidad Nacional de Entre Ríos

Strings, Regex y Colecciones Lineales

Condiciones de Entrega

- El trabajo práctico deberá ser:
  - Realizado en forma individual o en grupos de no más de tres alumnos.
  - Publicado en un repositorio en GitHub creado por el equipo de trabajo cuya URL deberá especificarse en la Actividad del Campus Virtual o cargado en la sección del Campus Virtual correspondiente, junto con los archivos de código fuente que requieran las consignas. Cada uno de ellos en sus respectivos formatos pero comprimidos en un único archivo con formato zip, rar, tar.gz u otro.
    - En caso de realizar el Trabajo Práctico en grupo, deberá indicarse el apellido y nombre de los integrantes del mismo.
  - Ser entregado el **Lunes 10 de Octubre de 2022.**
- El práctico será evaluado con nota numérica y conceptual (Excelente, Muy Bueno, Bueno, Regular y Desaprobado), teniendo en cuenta la exactitud y claridad de las respuestas, los aportes adicionales fuera de los objetivos de los enunciados y la presentación estética.
- Las soluciones del grupo deben ser de autoría propia. Aquellas que se detecten como idénticas entre diferentes grupos serán clasificadas como **MAL** para todos los involucrados en esta situación que será comunicada en la devolución.
- Los ejercicios que exijan codificación se valorarán de acuerdo a su exactitud, prolijidad (identación y otras buenas prácticas).

Consideraciones

El equipo de cátedra considera que la corrección de los Trabajos Prácticos presentados por los alumnos es más ágil si no se programan interfaces de consola de comandos que exigen al usuario interactuar con la aplicación. Por tanto, se solicita no programar clases cliente que hagan uso de la función **`input()`** o similares**.**

Ejercicios

1. Utilizando una **único** patrón de expresión regular, verifique si un **`string`** se corresponde con un número telefónico con la característica de la ciudad de Concordia. Valores de prueba:

   (0345) 154123456 (válido).
   +5493454123456 (válido).
   3454123456 (válido).
   +54011123456 (no válido).
   34564123456 (no válido).

2. Programe la clase **`LinkedStackExt`** de forma que extienda la clase **`LinkedStack`** (vista en clase) y a su vez implemente los métodos definidos en la clase abstracta **`LinkedStackExtAbstract`**:

```python
from abc import ABC, abstractmethod
from typing import Any, List

class LinkedStackExtAbstract(ABC):
    """Representa un conjunto de métodos para extender la implementación original
    de LinkedStack.

    Args:
        ABC (_type_): _description_
    """
    @abstractmethod
    def multi_pop(self, num: int) -> List[Any]:
        """Realiza la cantidad de operaciones pop() indicada por num.

        Args:
            num (int): número de veces que se va a ejecutar pop().

        Raises:
            Exception: Arroja excepción si se invoca a pop() por cuando la estructura
            está vacía.

        Returns:
            List[Any]: lista formada por todos los topes que se quitaron de la pila.
        """
        pass

    @abstractmethod
    def replace_all(self, param1: Any, param2: Any) -> None:
        """Reemplaza todas las ocurrencias de param1 en la pila por param2.

        Args:
            param1 (Any): Valor a buscar/reemplazar.
            param2 (Any): Nuevo valor.
        """
        pass

    @abstractmethod
    def exchange(self) -> None:
        """Intercambia el elemento ubicado en el tope con el más antigüo o último.
        Raises:
            Exception: Arroja excepción si la estructura está vacía.
        """
            pass
```

3. Programe en un archivo diferente un script cliente con nombre **`linked_stack_ext_client`** donde se ponga a prueba la clase **`LinkedStackExt`** programada en el ejercicio anterior.

4. Un **Deque (Double Ended Queue) o Cola Doble** es una estructura de datos lineal que soporta inserción y eliminación por ambos extremos de la colección. Utilizando una representación por enlaces implemente la interfaz que se detalla a continuación:

```python
from typing import Any
from abc import ABC, abstractmethod

class DequeAbstract(ABC):
    """ Representa una estructura de datos lineal de acceso restringido
    que soporta inserción y eliminación por ambos extremos.

    Args:
        ABC (_type_):
    """

    @abstractmethod
    def __len__(self) -> int:
        """Devuelve la cantidad actual de elementos en la estructura.

        Returns:
            int: Devuelve la cantidad de elementos, cero la si estructura está vacía.
        """
        pass

    @abstractmethod
    def __str__(self) -> str:
        """ Concatena en un único string todos los elementos almacenados
        en los nodos de la estructura.

        Returns:
            str: convierte en str todos los elementos y los concatena en un string.
        """
        pass

    @abstractmethod
    def is_empty(self) -> bool:
        """Indica si la estructura está vacía.

        Returns:
            bool: True si la estructura está vacía, False en caso contrario.
        """
        pass

    @abstractmethod
    def first(self) -> Any:
        """Devuelve el elemento ubicado en el frente de la estructura.

        Raises:
            Exception: Arroja excepción si la estructura está vacía.

        Returns:
            Any: Devuelve el elemento dato correspondiente al frente de la
            estructura.
        """
        pass

    @abstractmethod
    def last(self) -> Any:
        """ Devuelve el elemento correspondiente al nodo ubicado al final de
        la estructura.
```

```
        Raises:
            Exception: Arroja excepción si la estructura está vacía.

        Returns:
            Any: Devuelve el elemento dato correspondiente al final de la estructura.
        """
        pass

    @abstractmethod
    def add_first(self, element : Any) -> None:
        """_summary_

        Args:
            element (Any): elemento que va a ser agregado al principio de la
            estructura.
        """
        pass

    @abstractmethod
    def add_last(self, element : Any) -> None:
        """Agrega un elemento al final de la estructura.

        Args:
            element (Any): elemento que va a ser agregado al final de la estructura.
        """
        pass

    @abstractmethod
    def delete_first(self) -> None:
        """Quita el elemento ubicado en el frente de la estructura.

        Raises:
            Exception: Arroja excepción si la estructura está vacía.
        """
        pass

    @abstractmethod
    def delete_last(self) -> None:
        """Quita el elemento ubicado al final de la estructura.

        Raises:
            Exception: Arroja excepción si la estructura está vacía.
        """
            pass
```

5. Ponga a prueba la clase **Deque** y sus métodos programados en un archivo con nombre **deque_client**.

6. Programe la estructura de datos **DoubleLinkedList (Lista Doblemente Enlazada)** utilizando representación por enlaces. En este tipo de estructura, los nodos de la misma cuentan con enlaces que referencian tanto a su sucesor como a su antecesor lo que permite que se puedan realizar recorridos en ambos sentidos, sin la necesidad de variables auxiliares. Programe la clase **DoubleLinkedList** como derivada de **DoubleLinkedListAbstract**:

```
from abc import ABC, abstractmethod
from typing import Any, Iterator

class DoubleLinkedListAbstract(ABC):
```

```
    """Representa una lista doblemente enlazada en la cada nodo tienen referencia a su
antecesor y sucesor."""

    @abstractmethod
    def __len__(self) -> int:
        """Devuelve la cantidad de elementos que tiene actualmente la lista.

        Returns:
            int: entero que indica a longitud de la lista. 0 si está vacía.
        """
        pass

    @abstractmethod
    def __getitem__(self, key: int) -> Any:
        """Devuelve el elemento ubicado en la posición indicada por key.

        Args:
            key (int): posición de la que se va a retornar el elemento de la lista.

        Raises:
            Exception: Arroja excepción si el índice está fuera de rango.

        Returns:
            Any: Devuelve el valor ubicado en la posición indicada por key.
        """
        pass

    @abstractmethod
    def __setitem__(self, key: int, value: Any) -> None:
        """En la posición indicada por key se va a colocar value.

        Args:
            key (int): posición que se va a actualizar.
            value (Any): nuevo valor que se va a colocar.

        Raises:
            Exception: Arroja excepción si el índice está fuera de rango.
        """
        pass

    @abstractmethod
    def __delitem__(self, key: int) -> None:
        """Elimina de la estructura el elemento ubicado en la posición key.

        Args:
            key (int): posición cuyo nodo se va a quitar de la estructura.

        Raises:
            Exception: Arroja excepción si el índice está fuera de rango.
        """
        pass

    @abstractmethod
    def __iter__(self) ->  Iterator[Any]:
        """ Visita desde el principio hacia el final todos los nodos de la lista y
        retorna sus elementos.

        Yields:
            Iterator[Any]: Cada uno de los elementos de los nodos de lista.
        """
        pass

    @abstractmethod
    def __str__(self) -> str:
        """Concatena en un único string todos los elementos de la lista.

        Returns:
```

```
            str: string con todos los elementos de la lista convertidos en str.
        """
        pass

    @abstractmethod
    def is_empty(self) -> bool:
        """ Indica si la estructura está vacía.

        Returns:
            bool: True si la lista está vacía, caso contrario, False.
        """
        pass

    @abstractmethod
    def append(self, elem: Any) -> None:
        """ Agrega el elemento pasado por parámetro al final de la lista.

        Args:
            elem (Any): elemento que va a quedar ubicado en la última posición
        """
        pass
```

7. Ponga a prueba las clases y métodos programados en el punto anterior en una archivo con nombre **`double_linked_list_client`**.

Proyectos de programación (opcional)

1. Procesar utilizando expresiones regulares el código fuente de la página Web que muestra la altura de los ríos de la Prefectura Naval Argentina. Para ello deberá:
   a) Abrir un navegador Web y en la barra de direcciones ingresar el texto: "`view-source:`https://contenidosweb.prefecturanaval.gob.ar/alturas/index.php"
   b) Guardar el código de la página Web en el sistema de archivos local. (Generalmente botón derecho y luego "Guardar como...")
   c) Construir un nuevo proyecto Java en donde se realicen las siguientes tareas:
      i. Abrir el archivo de la página Web (**utilizar conjunto de caracteres `re.UNICODE`**) y procesar una línea a la vez.
      ii. Utilizando una **expresiones regulares** procesar cada string obteniendo los campos: Puerto, Río, Último Registro, Fecha Hora, y Estado:
      iii. Guardar en una lista de objetos de tipo **`RegistroAlturaRio`** los datos.
   d) Programe los siguientes métodos para filtrar resultados:
      i. **`filtrar_por_puerto(`**`nombrePuerto: str`**`) → List[RegistroAlturaRio]`**.
      ii. **`filtrar_por_rio(`**`rio: Rio`**`) → List[RegistroAlturaRio]`**.
      iii. **`filtrar_por_estado(`**`estado: EstadoEnum`**`) → List[RegistroAlturaRio]`**.