Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2021.

PROYECTO 2 – PINES CON ELEMENTOS REACTIVOS

202112092 – BRANDON EDUARDO PABLO GARCIA

Resumen

Desarrollo de un programa a través de lenguaje Python donde fue utilizado POO (Programación Orientada a Objetos), listas con elementos TDA´s (Tipos de Datos Abstractos) y Graphviz que es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.

El programa se basa en poder leer archivos XML para generar graficas de organismos donde representa como seria el comportamiento de los mismo y poder estudiarlos de mejor manera.

La introducción de los organismos será como muestras diferenciales por colores para una mejor compresión de los usuarios.

Palabras clave

Listas anidadas: Una lista dentro de otra

Elemento: tipo de materia formada por átomos de la misma clase

XML: Es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium utilizado para almacenar datos en forma legible

Abstract

Development of a program through Python language where it was used OOP (Object Oriented Programming), lists with elements TDA's (Abstract Data Types) and Graphviz which is a set of software tools for the design of diagrams defined in the descriptive language DOT.

The program is based on being able to read XML files to generate graphs of organisms where it represents how would be the behavior of the same and to be able to study them in a better way.

The introduction of the organisms will be as differential samples by colors for a better understanding of the users.

Keywords

Nested lists: A list inside another one.

Element: type of matter made up of atoms of the same kind.

XML: A metalanguage for defining markup languages developed by the World Wide Web Consortium used to store data in a readable form.

Los estudios epidemiológicos ayudan a comprender los efectos de las enfermedades en el cuerpo humano y encontrar las soluciones más eficaces para la prevención y el tratamiento de enfermedades.

Los científicos utilizaron cuadrículas cuadradas de tejido enfermo para identificar patrones de comportamiento en las células infectadas y descubrieron que las condiciones específicas determinaban si la enfermedad era leve o grave.

Este documento aborda el tema de la investigación epidemiológica a través de un programa desarrollado en un entorno de Visual Studio Code con lenguaje Python y así comprender los modelos que utilizan los científicos para determinar la gravedad de la enfermedad con el uso de documentos denominados XML.

Las preguntas que se abordarán en este documento son: ¿Cómo los modelos respaldados por físicos previenen y tratan enfermedades? ¿Los softwares brindan una ayuda real a los científicos en el estudio de bacterias? ¿Cuáles son las implicaciones de estos resultados para futuros estudios epidemiológicos?

Desarrollo del tema

En la investigación epidemiológica es de suma importancia el desarrollar un software para determinar cómo sería la supervivencia de un organismo en un entorno real. El desarrollo de este software implica el uso de conocimientos en campos como la programación, la biología y la epidemiología.

Para ejecutar este programa, debe usar un archivo XML que contenga información básica del organismo tales como serian su nombre, código, descripción, y sobre su posición que será dada por unas etiquetas que ubicaran su fila y columna, para así otorgarle un tamaño de cuadrícula y una pantalla de cuadrícula con una plantilla básica. Esta información es procesada por el sistema para determinar el curso de los organismos.

Lectura de archivos XML:

En Python, se puede leer un archivo XML utilizando la librería estándar "xml.etree.ElementTree". Primero, se debe importar la librería; a continuación, se puede utilizar la función "parse" de la librería para abrir y parsear el archivo XML; una vez que se ha parseado el archivo XML, se puede acceder a los elementos del árbol XML utilizando la función "getroot". A partir de aquí, se puede navegar por el árbol XML utilizando los métodos y atributos del objeto "root" y sus hijos

Creación de TDA’s:

Para crear listas con ADT´s (tipos de datos abstractos) en programación, primero debe definir el TDA a usar y su estructura de datos. Luego, la lista se puede implementar utilizando la estructura de datos TDA.

Por ejemplo, si desea crear una lista con un TDA de pila (stack), puede definir el TDA mediante una estructura de datos que incluya una lista interna y métodos para agregar y eliminar elementos de la pila.

Se define el TDA de una pila con los métodos “push”, “pop” y” is_empty”. Luego, se utiliza esta estructura de datos en la clase “StackList” para implementar una lista que funciona como una pila. La clase “StackList” tiene los métodos “add_item” y “remove_item”, que se utilizan para agregar y eliminar elementos de la lista. La lista interna de la clase “StackList” es una pila que utiliza el TDA definido en la clase “Stack”.

Esta estructura de datos se puede usar para implementar una lista que actúa como una pila.

Pero; ¿Qué definimos como una clase? Su definición es la siguiente: “Las clases son los modelos sobre los cuáles se construirán nuestros objetos. En Python, una clase se define con la instrucción class seguida de un nombre genérico para el objeto.” Y los métodos de la misma manera son “Los métodos son *funciones*, solo que técnicamente se denominan métodos, y representan acciones propias que puede realizar el objeto (y no otro)”. Para comprender de mejor manera se realizó el siguiente diagrama.

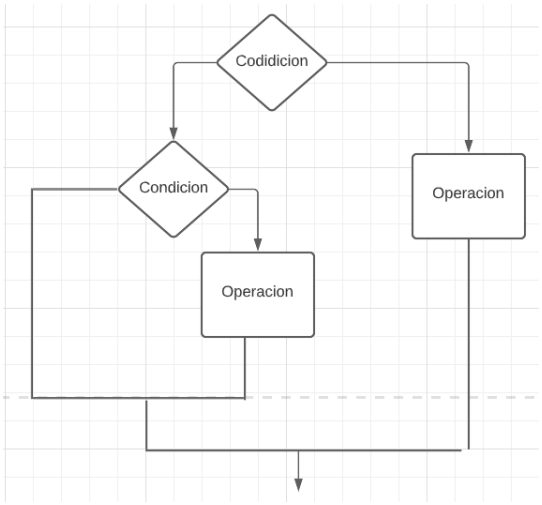

Figura 1. Diagrama de una lista anidada

Fuente: elaboración propia

Explicado de forma general eso incluirá nuestro código solo que para eso fueron creadas clases con nombres muy específicos; alguno de esos seria “class organismos” que nos ayudó guardar los elementos leídos del archivo XML guardados en funciones para después guárdalos en listas con objetos, objetos como serían los nombres, los códigos y sobre todo su posición en la tabla que está determinada por una fila y columna.

Existen muchas más clases como los serian “class CeldaViva”, como su nombre lo indica se utilizo para guardar los organimos vivos, otra clase más utilizada fue “class FilayColumna” el más impórtate que se uso para colocar los organismos en la gráfica que dará la posiciones del mismo y poder lograr así ver todas los organismos y tomar las siguientes consideraciones: Que la muestra sea distintiva, que se logre ver el código del organismo, y validar cosas como lo serian si el organismo sobrevira, si podrá lograr su reproducción entre otras cosas.

Como podemos apreciar podemos separar los organismos por clases y cada clase tiene sus funciones. Existen sobre todo las clases Nodos que son una base fundamental de las listas empleadas
Ciertamente la definición correcta de un nodo seria: "Un nodo, en informática, es un componente que forma parte de una red. En otras palabras, tanto si se trata de Internet como de Intranet (utilizada en ámbitos cerrados, con acceso limitado a los usuarios autorizados), cada servidor u ordenador constituye un nodo y se encuentra conectado a otro u otros nodos."

Una lista enlazada es una estructura de datos que puede utilizarse para la implementación de nuevas estructuras (tales como las colas, las pilas y sus derivados) y está formada por una serie de nodos que almacenan, además de la información deseada, un enlace, un puntero o una referencia al nodo que lo precede, al posterior, o bien uno a cada uno. La ventaja fundamental de una lista enlazada en comparación con un vector convencional es que sus elementos no presentan un orden rígido ni relacionado con el que tuvieron al momento de ser almacenados, sino que éste depende del enlace que posee cada nodo, y puede ser modificado cuando así se desee.

Los principales a usar son el self.cabeza y self.final que su importancia va más a ya de indicar el inicio y el final se la lista para recorrer dicha lista se self.siguiente. Self.siguiente es una implementación que nos ayuda a tener el siguiente elemento de una lista como lo fue mencionado anteriormente.

Para completar de manera correcta el código y su implementación se usó Graphviz. Graphviz es “Es un conjunto de herramientas open-source realizado inicialmente en los laboratorios de investigación de

AT&T para el dibujo de gráficos especificados en lenguaje de scripts DOT. Provee librerías para ser usadas por otras aplicaciones. Graphviz es software libre licenciado bajo CPL (Common Public License).”

En las aplicaciones de Graphviz también tenemos:

- Estructuras de datos.
- Estructuras de árbol.
- Representación de análisis social de redes.
- Diagramas entidad relación.
- Diagramas de redes.
- Diagramas de flujo.
- Diagramas de procesos.

Graphiz es muy útil para esta parte del código porque nos ayuda a poder graficar cada organismo y sus posiciones. Esta parte ya mencionada es esencial para entender su comportamiento y el usuario intérprete de mejor manera el...
Para dar un ejemplo se implementó una tabla donde se mostrará el código del organismo, su fila y columna

Tabla I.

Lista de Elementos

Elemento	SIMBOLO	NOMBRE
1	H	Helio
2	He	Helio
3	Li	Litio
4	Be	Berilio
5	B	Boro

Fuente: elaboración propia

Lista de Maquinas

Elemento
H
He
Li
Be
B

Fuente: elaboración propia

Conclusiones

En conclusión, desarrollar un software de aplicación para determinar la gravedad o mortalidad de un organismo a partir de la información un archivo XML es una tarea compleja que requiere la integración de conocimientos de diferentes disciplinas. La visualización gráfica de la evolución de la enfermedad en la red, la identificación de patrones y la generación de datos en bruto en formato XML son los aspectos principales del desarrollo de esta aplicación.

Referencias bibliográficas

Anonimo. (04 de febrero de 2023). *uniwebsidad*. Obtenido de uniwebsidad: https://uniwebsidad.com/libros/python/capitulo-5/programacion-orientada-a-objetos

Pérez Porto, J. G. (04 de 03 de 2023). *Definicion.de*. Obtenido de Definicion.de: https://definicion.de/nodo/

Souza, I. d. (08 de 04 de 2023). *reckcontent*. Obtenido de reckcontent: https://rockcontent.com/es/blog/que-es-xml/

Figura 1. Lectura de archivo XML

```python
class Pin:
    def __init__(self):
        self.elemento = None
        self.posicion = 0

    def avanzar(self):
        self.posicion += 1

    def retroceder(self):
        self.posicion -= 1

    def esperar(self):
        pass

    def fusionar(self, elemento):
        if self.elemento is None:
            self.elemento = elemento
        else:
            print("Este pin ya contiene un elemento químico")
```

Figura 2. Función de maquinas

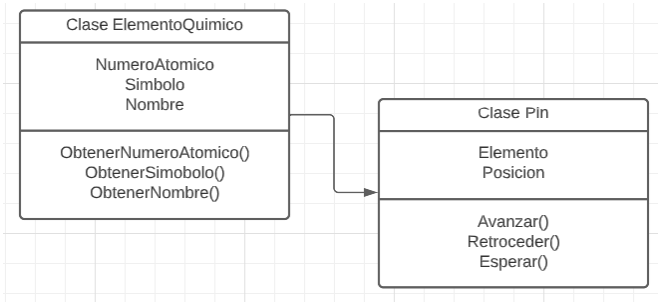

Figura 3. Diagrama de maquinas

```python
class Maquina:
    def __init__(self, nombre, numPines):
        self.nombre = nombre
        self.pines = [Pin() for i in range(numPines)]
        self.elementos = set()

    def fusionar(self, index):
        pin_actual = self.pines[index]
        if pin_actual.elemento is not None:
            if pin_actual.posicion == len(self.pines) - 1:
                compuesto = self._crear_compuesto()
                print("Compuesto creado:", compuesto)
                self.elementos.clear()
                for pin in self.pines:
                    pin.elemento = None
                    pin.posicion = 0
            else:
                pin_siguiente = self.pines[index + 1]
                if pin_siguiente.elemento is None:
                    pin_siguiente.elemento = pin_actual.elemento
                    pin_actual.elemento = None
                    pin_actual.avanzar()
                    pin_siguiente.avanzar()
                    self.elementos.add(pin_siguiente.elemento)
                else:
                    print("El pin siguiente ya contiene un elemento químico")
        else:
            print("Este pin no contiene ningún elemento químico")

    def _crear_compuesto(self):
        elementos = sorted(list(self.elementos), key=lambda x: x.numeroAtomico)
        nombre_compuesto = ""
        for elemento in elementos:
            nombre_compuesto += elemento.simbolo
        return nombre_compuesto
```

Figura 4. Implementación de clases maquinas