DOCUMENTACIÓN DE PRACTICA I DE ESTRUCTURA DE DATOS

Santiago Sanchez Carvajal

Clase: Nodo

En la clase nodo básicamente tenemos su constructor el cual contiene los valores pregunta el cual representa los **síntomas** y **enfermedad** los cuales se insertar con una función aparte.

En general todos los nodos cuentan con el valor pregunta menos las hojas que cuentan con el valor **enfermedad,** pero no el valor **pregunta.**

Aparte de eso contamos con los valores **si** y **no** los cuales cumplen la función de ser las ramas o los apuntadores, los cuales **apuntan** a los nodos del siguiente nivel.

```
3 usages
class Node:
    def __init__(self, question, disease):
        self.question = question
        self.disease = disease
        self.yes = None
        self.no = None
```

Función construir_Arbol:

Para construir ocupamos una lista con las preguntas sobre los síntomas dados en el exel.

```
question = ['¿Tiene fiebre?', '¿Tiene dolor de cabeza?', '¿Tiene Falta de aire?', '¿Tiene Mareo?',
            '¿Tiene Dolor muscular?', '¿Tiene Tos?']
```

Estas preguntas se llevan a la función **build_tree** al principio se crea la **raíz** con un nodo el cual va a tener como pregunta el primer elemento de la lista "**¿tiene fiebre?**

Se crea una lista donde se guarda los nodos del nivel anterior, para asi en el ciclo for poder modificar los valores de si y no de cada nodo de dicho nivel.

Esta lista va a actualizarse constantemente gracias a la creación de una nueva lista la cual ocupamos para guardar los nodos del **nivel actual**, los cuales se convertirán después en los nodos del anterior nivel.

```
def build_tree(question):
    root = Node(question[0], None)
    before_level = [root]
    for level in range(1, 6):
        current_level = []
        for node in before_level:
            node.yes = Node(question[level], None)
            node.no = Node(question[level], None)
            current_level.append(node.yes)
            current_level.append(node.no)
        before_level = current_level
    return root
```

Al final devolvemos la raíz la cual es el inicio del árbol.

Función añadir enfermedad:

Ahora con el árbol ya creado, solo falta añadir las enfermedades a su hoja correspondiente.

Para ello, lo primero que hay que saber es que se paso del exel a csv para poder leerlo dentro del programa, de tal manera que cada valor se separa con “**;**”

```
Si;Si;Si;Si;Si;Si;Influenza
Si;Si;Si;Si;Si;No;Dengue
Si;Si;Si;Si;No;Si;Gripe
Si;Si;Si;Si;No;No;Neumonía
Si;Si;Si;No;Si;Si;Meningitis
Si;Si;Si;No;Si;No;Chikungunya
Si;Si;Si;No;No;Si;Virus del Nilo Occidental
Si;Si;Si;No;No;No;Rubéola
Si;Si;No;Si;Si;Si;Sarampión
Si;Si;No;Si;Si;No;Varicela
Si;Si;No;Si;No;Si;Herpes zóster
Si;Si;No;Si;No;No;Fiebre maculosa de las Montañas Rocosas
Si;Si;No;No;Si;Si;Mononucleosis
Si;Si;No;No;Si;No;Granulomatosis de Wegener
Si;Si;No;No;No;Si;Linfoma
Si;Si;No;No;No;No;Virus del Zika
```

Este se lee con las siguientes líneas de código, sigue el mismo concepto utilizado para leer el JSON, todas las listar de síntomas con sus enfermedades, se agregan en una sola y se vuelve matriz, esta se utilizará para la función **add_diseases**

```
with open('enfermedades.csv', 'r', encoding="utf-8") as file:
    next(file, None)

    for line in file:
        line = line.rstrip()

        list = line.split(";")

        diseases.append(list)
```

En la función **add_diseases** se va viajando por el árbol, por medio de un nodo auxiliar el cual se igual a la **raíz** como se pudo ver en el **csv** hay valores **Si** y **No,** estos valores serán los utilizados para viajar, por ejemplo, si el valores es **Si** y el nodo siguiente no está vacío este se iguala al nodo del siguiente nivel, pero si el valor es **Si** y el valor del siguiente nivel es vacío, significa que se pueden insertar la enfermedad, la cual esta en la sexta posición de la lista.

Matriz obtenida el csv:

```
[['Si', 'Si', 'Si', 'Si', 'Si', 'Si', 'Influenza'], ['Si', 'Si', 'Si', 'Si', 'Si', 'No', 'Dengue'], ['Si', 'Si', 'Si', 'Si', 'No', 'Si', 'Gripe'],
```

```
def add_diseases(node, diseases, level):
    node_aux = node
    for i in range(7):
        if diseases[level][i] == "Si" and node_aux.yes is not None:
            node_aux = node_aux.yes
        elif diseases[level][i] == "Si" and node_aux.yes is None:
            node_aux.yes = Node(None, diseases[level][6])
        if diseases[level][i] == "No" and node_aux.no is not None:
            node_aux = node_aux.no
        elif diseases[level][i] == "No" and node_aux.no is None:
            node_aux.no = Node(None, diseases[level][6])
```

```
for i in range(len(diseases)):
    add_diseases(root, diseases, i)

print(diseases)
```

La función través, nos permite viajar por el árbol ya terminado, según las respuestas del usuario, las cuales obtendremos, por medio de las casillas que el mismo llene en la interfaz gráfica. Esta recibe una lista de valores booleanos, los cuales determinan si va para el nodo.si o el nodo.no

```python
def travel(node, answers):
    node_aux = node
    for i in range(6):
        if not answers[i]:
            node_aux = node_aux.yes
        else:
            node_aux = node_aux.no
```

Al final, el programa te devuelve la enfermedad, según como haya sido el recorrido del árbol, y este te permitirá ver un video hablando sobre dicha enfermedad, esto se consigue por medio de una librería llamada **webbrowser.**

Este ocupa una url la cual se guarda dentro de un diccionario, el cual tiene como llave el mismo nombre de la enfermedad, de esta manera se accede a la url.

```python
videos = {
    "Influenza": "https://www.youtube.com/watch?v=hK5Oh3pWZyo",
    "Dengue": "https://www.youtube.com/watch?v=k2ohxoNUK9M",
    "Gripe": "https://www.youtube.com/watch?v=5uM7Z0xRBx4",
    "Neumonía": "https://www.youtube.com/watch?v=l1H7zDAOIU0",
    "Meningitis": "https://www.youtube.com/watch?v=w0tukS41qx4&pp=ygUKTWVuaW5naXRpcw%3D%3D",
    "Chikungunya": "https://www.youtube.com/watch?v=7wJddLVZrkY&pp=ygULQ2hpa3VuZ3VueWE%3D",
    "Virus del Nilo Occidental": "https://www.youtube.com/watch?v=G6V_uamtbcY&pp=ygUaRmllYnJlIGRlbCBOaWxvIE9iY2lkZW50YWw%3D",
    "Rubéola": "https://www.youtube.com/watch?v=7a0VAbzmRLM&pp=ygUIUnViw6lvbGE%3D",
    "Sarampión": "https://www.youtube.com/watch?v=ob4i_Zuz5gs&pp=ygUKU2FyYW1wacOzbg%3D%3D",
    "Varicela": "https://www.youtube.com/watch?v=-ZUXwuqYHRc&pp=ygUIVmFyaWNlbGE%3D",
```

```python
engine.say(f'Lamento informarle que usted sufre de {node_aux.disease
engine.say(
    f'¿Desea ver un video hablando sobre {node_aux.disease}? oprime
engine.runAndWait()
aux = input(f'¿Ver video? (S/N): ')
if aux.lower() == "s":
    engine.say(
        f'Desde potoncio industries (patente en proceso), le damos
    engine.runAndWait()
    webbrowser.open(videos[node_aux.disease])
```

Interfaz Gráfica:

Para la interfaz usamos la librería **tKinter**.

Creamos una ventana, la cual va a tener como dimensiones 300x250

Después creamos una lista la cual contendrá la respuesta que brinde el usuario a cada pregunta, esta lista es una lista de objetos **booleanos** esto para asi utilizarla en la función add_diseases.

Usamos **labels** para poner las preguntas y el título de inicio, también ocupamos **grid** como la manera de organizar los elementos de la interfaz (Por filas y columnas)

```
window = tk.Tk()
window.title("diagnosticación de enfermedades")
window.geometry("300x250")

answers = [tk.BooleanVar() for i in range(len(question))]

tk.Label(window, text="diagnosticación de enfermedades").grid(row=0, column=0)
tk.Label(window, text="").grid(row=1, column=0)
```

Después se crea los respectivos labels (Preguntas), botones y recuadros, para cada fila por medio de un ciclo for el cual va cambiando la pregunta y la fila en que se ingresa dicha pregunta.

No hemos hablado de ella por ahora, pero el programa cuenta con mi primera versión de Gio, un asistente virtual, por ahora solo puede decir lo que se le ordene. En este caso usamos a Gio para que cada vez que se oprima el botón que contiene el altavoz está repita la pregunta

```
for i in range(len(question)):
    tk.Label(window, text=question[i]).grid(row=i + 2, column=0)
    tk.Checkbutton(window, variable=tk.BooleanVar(value=False),
                   command=lambda i=i: answers.__setitem__(i, not answers[i])).grid(row=i + 2, column=1)
    tk.Button(window, text="🔊", command=lambda q=question[i]: voice(q)).grid(row=i + 2, column=2)

tk.Label(window, text="").grid(row=8, column=0)
tk.Button(window, text="Confirmar", command=end_window).grid(row=9, column=0)
tk.Button(window, text="Instrucciones", command=lambda
```

Función para que Gio hable en la interfaz

```
def voice(texto):
    engine.say(texto)
    engine.runAndWait()
```

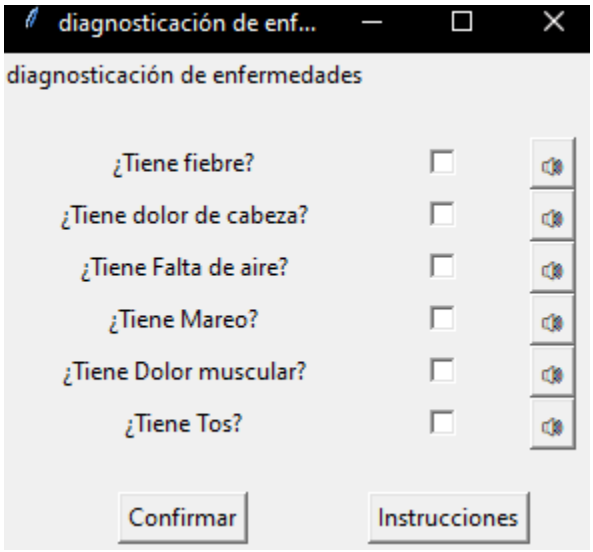
Al final tenemos dos últimos botones, una para **confirmar** la respuesta, este botón a su vez llama a la función **end_window** que simplemente cierra la interfaz

```
1 usage
def end_window():
    window.destroy()
```

El otro botón es el botón de instrucciones el cual simplemente hace que Gio mencione las instrucciones básicas de lo que hay que hacer en la interfaz.

```
tk.Label(window, text="").grid(row=8, column=0)
tk.Button(window, text="Confirmar", command=end_window).grid(row=9, column=0)
tk.Button(window, text="Instrucciones", command=lambda
    g='A continuación se te darán algunas preguntas sobre diferentes síntomas qu
    g)).grid(row=9, column=1)
tk.Label(window, text="").grid(row=10, column=0)

window.mainloop()
```



Gio: Gio es un asistente virtual, la cual espero ir actualizando mediante nuevos conocimientos obtenga de mi carrera, por ahora solo dicta oraciones las cuales se ingresan previamente.

```
engine.say(f'¿Desea escuchar la introducción al programa? oprime la letra s para si, de lo contrario oprime la letra n')
engine.runAndWait()
```

Configuraciones de Gio:

```
#_----------------------------------------
engine = pyttsx3.init()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[2].id)
#_----------------------------------------
```