Abschlussarbeit_Giordano

August 22, 2024

[ ]:
```
import pandas as pd

# Excel-Datei lesen
df = pd.read_excel("systematik.xlsx")
df
```

[ ]:
```
#Klasse namens Node ist definiert, die einen Knoten in einem Baum repräsentiert.
#Jeder Knoten hat einen name, eine optionale url und eine Liste von children,␣
 ↪die untergeordnete Knoten enthalten.
#Die Methode add_child ermöglicht es, einen untergeordneten Knoten (Child) zum␣
 ↪aktuellen Knoten hinzuzufügen.

class Node:
    def __init__(self, name, url=None):
        self.name = name
        self.url = url
        self.children = []

    def add_child(self, child_node):
        self.children.append(child_node)
```

[ ]:
```
#Diese Funktion build_tree_from_df erstellt einen Baum (oder Wald) aus dem␣
 ↪DataFrame df.
#Für jede Zeile im DataFrame wird geprüft, welche Ebenen (Levels) ausgefüllt␣
 ↪sind
#und diese Ebenen werden als Knoten im Baum dargestellt.
#Wenn die letzte Ebene eine URL ist, wird sie dem entsprechenden Knoten␣
 ↪zugewiesen.
#Schließlich wird ein Wurzelknoten (root) erstellt, und jeder Knoten wird,␣
 ↪falls notwendig, mit untergeordneten Knoten verknüpft.
#Die Funktion gibt eine Liste von Wurzelknoten zurück, die den gesamten Baum␣
 ↪darstellen.


def build_tree_from_df(df):
    roots = {}
    for _, row in df.iterrows():
```

```
        # Identifiziere die ausgefüllten Ebenen (nicht null)
        levels = [row[f'Level{i}'] for i in range(1, df.shape[1] + 1) if pd.
 ↪notna(row[f'Level{i}'])]

        # Betrachte die letzte Ebene als URL, falls zutreffend
        url = levels.pop() if levels and isinstance(levels[-1], str) and␣
 ↪levels[-1].startswith(('http://', 'https://')) else None

        current_node = None
        for i, level in enumerate(levels):
            if i == 0:
                if level not in roots:
                    roots[level] = Node(level)
                current_node = roots[level]
            else:
                child_node = next((child for child in current_node.children if␣
 ↪child.name == level), None)
                if not child_node:
                    child_node = Node(level)
                    current_node.add_child(child_node)
                current_node = child_node

        # Weise die URL dem letzten Knoten der Zeile zu, falls vorhanden
        if url:
            current_node.url = url

    return list(roots.values())
```

[ ]:
```
#Diese folgenden beiden Funktionen generieren den HTML-Code, der die␣
 ↪Baumstruktur darstellt.
#Die Funktion generate_html nimmt einen Knoten (Node) und erstellt ein␣
 ↪Listenelement (<li>) im HTML-Format.
#Wenn der Knoten eine URL hat, wird sein Name als anklickbarer Link dargestellt.
#Falls der Knoten untergeordnete Knoten hat, wird eine verschachtelte Liste␣
 ↪(<ul>) generiert,
#die nur sichtbar wird, wenn auf den Knoten geklickt wird.
#Die Funktion generate_forest_html kombiniert den HTML-Code aller Wurzelknoten␣
 ↪zu einem vollständigen HTML-Dokument,
#das die Baumstruktur als interaktive Liste darstellt.
#Zusätzlich wird ein JavaScript-Code eingebunden, der es ermöglicht,
#die Listenpunkte durch Klicken auf die Knoten zu expandieren oder zu␣
 ↪minimieren.

def generate_html(node):
    # Definiere die Farbe und den Stil des Links, falls eine URL vorhanden ist
    if node.url:
```

```
        name_html = f'<a href="{node.url}" target="_blank" style="color:␣
↪#007c6c; text-decoration: underline;">{node.name}</a>'
    else:
        name_html = node.name

    if not node.children:
        return f'<li>{name_html}</li>'

    children_html = ''.join(generate_html(child) for child in node.children)
    return f'''
    <li>
        <span class="caret">{name_html}</span>
        <ul class="nested">
            {children_html}
        </ul>
    </li>
    '''

def generate_forest_html(forest):
    forest_html = ''.join(generate_html(root) for root in forest)
    return f'''

<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="shortcut icon" href="logo.jpg" type="image/x.icon">
    <title>MPILs Systematik</title>
    <style>
        ul {{
            list-style-type: none; /* Entferne Aufzählungszeichen */
        }}
        li {{
            color: black; /* Standard-Textfarbe ist schwarz */
        }}
        li:hover {{
            color: #007c6c; /* Minerva Grün bei Hover */
        }}
        a {{
            color: black; /* Erbe die Textfarbe */
            text-decoration: none; /* Entferne Unterstreichung standardmäßig */
        }}
        a:hover {{
            color: #007c6c; /* Minerva Grün bei Hover */
            text-decoration: underline; /* Unterstreichung bei Hover */
        }}
```

```
        .nested {{
            display: none;
        }}
        .active {{
            display: block;
        }}
        .caret {{
            cursor: pointer;
            user-select: none;
            font-weight: bold; /* Hebe Caret-Items hervor */
        }}

    </style>
</head>
<body>
    <h2>MPILs Systematik</h2>
    <ul id="myMenu">
        {forest_html}
    </ul>
    <script>
        document.addEventListener('DOMContentLoaded', function() {{
            var toggler = document.getElementsByClassName("caret");
            for (var i = 0; i < toggler.length; i++) {{
                toggler[i].addEventListener("click", function() {{
                    this.parentElement.querySelector(".nested").classList.
 ↪toggle("active");
                    this.classList.toggle("caret-down");
                }});
            }}
        }});
    </script>
</body>
</html>
    '''

# Excel-Datei lesen
df = pd.read_excel('systematik.xlsx')

# Baum aus dem DataFrame erstellen
forest = build_tree_from_df(df)

# HTML-Inhalt generieren
html_content = generate_forest_html(forest)

# HTML-Inhalt in einer Datei speichern
with open('giordano.html', 'w') as file:
    file.write(html_content)
```

```
print("HTML-Datei 'giordano.html' wurde generiert.")
```

[ ]: