prototyp

March 13, 2024

Hinweise zum Umgang und Installation der notwendigen Pakete

[25]:
```
### Ziel: Installieren der notwendigen Pakete und Einrichtung der Umgebung. ###

#Empfehlung: Neue Conda-Umgebung einrichten im Terminal.
#conda create --name beng python=3.12.0
#conda activate beng

#Die bereits automatisch erstellten Dateien
    #formular_6_2_auto.xml
    #formular_6_2_auto.tex
    #formular_6_2_auto.pdf
#können gelöscht werden. Andernfalls werden sie einfach überschrieben.

#Alle Zellen ausführen.

#Es sind 2 grundlegende Eingaben notwendig.
    #Pumpe: P4712
    #Rohrleitungen: alle: Enter --> außer 47121: saug

#Automatische Erstellung von
    #formular_6_2_auto.xml
    #formular_6_2_auto.tex
#sollte erfolgen.

#Die formular_6_2_auto.pdf kann aus der formular_6_2_auto.tex compilliert werden.

%pip install requests
%pip install xml.etree.ElementTree
%pip install --upgrade --quiet langchain langchain-openai
%pip install python-dotenv
%pip install translate
%pip install reportlab
%pip install pdfkit
%pip install lxml
%pip install Jinja2
```

```
Requirement already satisfied: requests in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
```

```
(2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests) (2023.11.17)
Note: you may need to restart the kernel to use updated packages.
ERROR: Could not find a version that satisfies the requirement

xml.etree.ElementTree (from versions: none)

ERROR: No matching distribution found for

xml.etree.ElementTree

Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: python-dotenv in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(1.0.1)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: translate in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(3.6.1)
Requirement already satisfied: click in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from translate) (8.1.7)
Requirement already satisfied: lxml in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from translate) (5.1.0)
Requirement already satisfied: requests in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from translate) (2.31.0)
Requirement already satisfied: libretranslatepy==2.1.1 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from translate) (2.1.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests->translate) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests->translate) (2.10)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests->translate) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from requests->translate) (2023.11.17)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: reportlab in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(4.1.0)
Requirement already satisfied: pillow>=9.0.0 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from reportlab) (10.1.0)
Requirement already satisfied: chardet in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from reportlab) (3.0.4)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pdfkit in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(1.0.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: lxml in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(5.1.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: Jinja2 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(3.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/Users/millerfl/anaconda3/envs/schemaextraction/lib/python3.12/site-packages
(from Jinja2) (2.1.3)
Note: you may need to restart the kernel to use updated packages.
```

Zugriff DEXPI-R&I

[26]:
```
### Ziel: Zugriff auf das DEXPI-R&I im .xml-Format herstellen. ###
    # Quelle GitHub Repositry der Bachelorarbeit: https://github.com/FloT10/
 ↪MillerFlorinBA
    # Branch main: Hier wurde der Prototyp entwickelt
    # Branch Abgabe: Aufbereitung und Darbietung zur Abgabe

# Importieren des 'requests'-Moduls, um Daten von einer URL herunterzuladen, und␣
 ↪das 'xml.etree.ElementTree'-Modul, um XML-Daten zu verarbeiten.
import requests
import xml.etree.ElementTree as ET

# GitHub Raw-URLs für die DEXPI-Datei.
```

```
github_url_dexpi = 'https://raw.githubusercontent.com/FloT10/MillerFlorinBA/
 ↪Abgabe/C01V01-HEX.EX03.xml'

# Herunterladen der DEXPI-Datei von der GitHub-URL.
response = requests.get(github_url_dexpi)

# Speichern der heruntergeladenen DEXPI-Datei lokal im Textmodus ('w').
with open('/tmp/C01V01-HEX.EX03.xml', 'w') as file:
    file.write(response.text)

# Definieren eines Dateipfades zur lokal gespeicherten DEXPI-Datei zur weiteren␣
 ↪Verwendung.
dexpi_xml = "/tmp/C01V01-HEX.EX03.xml"

with open('/tmp/C01V01-HEX.EX03.xml', 'r') as file:
    dexpi_xml_test = file.read(1000)
    print(dexpi_xml_test)
```

```
<?xml version="1.0" encoding="utf-8"?>
<PlantModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../ProteusPIDSchema_4.0.1.xsd">
  <!--Created with INGR ISO15926 PostProc V0.0.7.1 and ConfigurationFileVersion
1.1 by PPM-VM-2014-R1\Intergraph on PPM-VM-2014-R1 at 4/12/2018 5:14:33 PM-->
  <PlantInformation SchemaVersion="4.0.1" OriginatingSystem="SPPID"
Date="2018-04-12" Time="15:13:38.0000000+02:00" Is3D="no" Units="Metre"
Discipline="PID">
    <UnitsOfMeasure Distance="Metre" />
  </PlantInformation>
  <PlantStructureItem ID="PBS945900F29F424D32A292F07BB90792E2"
ComponentClass="PlantSectionIso10209-2012" ComponentName="Github"
ComponentClassURI="http://sandbox.dexpi.org/rdl/PlantSectionIso10209-2012">
    <GenericAttributes Number="2" Set="DexpiAttributes">
      <GenericAttribute Name="PlantSectionIdentificationCodeAssignmentClass"
Value="Github" Format="string" AttributeURI="http://sandbox.dexpi.org/rdl/PlantS
ectionIdentificationCodeAssignment
```

Angaben zur Anlage

[27]:
```
### Ziel: Extrahieren der Anlagenbezeichnung ###
    # Die Zeichnungsnummer bzw. die RI-Bezeichnung wird als Anlagenbezeichnung␣
 ↪gewählt

# Funktion zum Laden und Extrahieren von Daten aus der DEXPI-Datei.
def extrahiere_anlagenbezeichnung(dexpi_xml):
    try:
        # Laden des XML-Dokuments.
        tree = ET.parse(dexpi_xml)
        root = tree.getroot()
```

```
        # Extrahieren der Anlagenbezeichnung aus dem XML-Baum.
        drawing_element = root.find('.//Drawing')
        if drawing_element is not None:
            anlagenbezeichnung = drawing_element.get('Name')
            return anlagenbezeichnung
        else:
            print("Die Anlagenbezeichnung wurde nicht gefunden.")
            return None

    except Exception as e:
        # Falls ein Fehler auftritt, Ausgabe einer Fehlermeldung und None als␣
 ↪Rückgabe.
        print(f"Fehler beim Laden und Extrahieren der Daten: {e}")
        return None

# Beispielaufruf der Funktion
anlagenbezeichnung = extrahiere_anlagenbezeichnung(dexpi_xml)

# Ausgabe der Variable.
if anlagenbezeichnung is not None:
    print(anlagenbezeichnung)
```

C01V01-HEX.EX03

[28]:
```
### Ziel: Extrahieren des Anlagenumfangs ###

# Funktion zum Laden und Extrahieren von Daten aus der DEXPI-Datei.
def extrahiere_anlagenumfang(dexpi_xml):
    try:
        # Laden des XML-Dokuments und Erhalten der Wurzel des Baums.
        tree = ET.parse(dexpi_xml)
        root = tree.getroot()

        # Extrahieren der ComponentClass aus allen Equipment- und␣
 ↪PipingNetworkSystem-Elementen.
        equipment_elements = root.findall('.//Equipment')
        piping_network_system_elements = root.findall('.//PipingNetworkSystem')

        component_classes = set()

        for equipment_element in equipment_elements:
            component_class = equipment_element.get('ComponentClass')

            # Überprüfe, ob die ComponentClass den erlaubten Werten entspricht.
            # Ausgerichtet am Beispiel-RI. Andere Anlagenbestandteile könnten␣
 ↪ergänzt werden.
```

```
            erlaubte_component_classes = ['Tank', 'ReciprocatingPump',␣
↪'CentrifugalPump', 'PlateAndShellHeatExchanger', 'ShellAndTubeHeatExchanger']
            if component_class in erlaubte_component_classes:
                component_classes.add(component_class)

        for piping_network_system_element in piping_network_system_elements:
            component_class = piping_network_system_element.get('ComponentClass')

            # Überprüfen, ob die ComponentClass den erlaubten Werten entspricht.
            erlaubte_component_classes = ['PipingNetworkSystem']
            if component_class in erlaubte_component_classes:
                component_classes.add(component_class)

        # Überprüfen, ob mindestens eine gültige ComponentClass gefunden wurde.
        if not component_classes:
            raise ValueError("Keine gültigen Anlagenbestandteile gefunden.")


        # Gib ein Dictionary mit den extrahierten Werten zurück.
        return {
            'Anlagenumfang (englisch)': list(component_classes),
            # Weitere extrahierte Werte könnten hier hinzugefügt werden.
        }

    except Exception as e:
        # Falls ein Fehler auftritt, gib eine Fehlermeldung aus und gibt None␣
↪zurück.
        print(f"Fehler beim Laden und Extrahieren der Daten: {e}")
        return None

# Beispielaufruf der Funktion und Ausgabe der extrahierten Daten.
anlagenumfang_englisch = extrahiere_anlagenumfang(dexpi_xml)
# print(anlagenumfang_englisch)

# Übersetzung des Amlagenumfangs von english zu deutsch
    # Ggf. muss Tabelle erweitert werden
def uebersetze_begriffe(begriffe, zielsprache='de'):
    manuelle_uebersetzungen = {
        'ReciprocatingPump': 'Kolbenpumpe',
        'PipingNetworkSystem': 'Rohrleitungssystem',
        'CentrifugalPump': 'Kreiselpumpe',
        'PlateAndShellHeatExchanger': 'Plattenwärmetauscher',
        'Tank': 'Behälter',
        'ShellAndTubeHeatExchanger': 'Rohrbündelwärmetauscher'
    }
```

```
    uebersetzte_begriffe = [manuelle_uebersetzungen.get(begriff, begriff) for␣
 ↪begriff in begriffe]
    return uebersetzte_begriffe

# Beispielaufruf
anlagenumfang_liste = uebersetze_begriffe(anlagenumfang_englisch['Anlagenumfang␣
 ↪(englisch)'])

# Verbinden der Elemente der Liste zu einem String mit Kommas
anlagenumfang = ", ".join(anlagenumfang_liste)


# Ausgabe der übersetzten Begriffe
print(anlagenumfang)
```

```
Kolbenpumpe, Rohrbündelwärmetauscher, Kreiselpumpe, Rohrleitungssystem,
Behälter, Plattenwärmetauscher
```

[29]:
```
### Ziel: Extrahieren der Anlagenart ###
    # Vorgehensweise: es wird der größte Tank gesucht. Dann wird untersucht ob␣
 ↪es sich um einen Lagertank oder einen Prozesstank handelt.
    # Davon wird die Anlagenart abgeleitet. Dieses Vorgehen muss nicht immer zu␣
 ↪sinnvollen Ergebnissen führen.
    # Die Vorgehensweise kann noch ausgebaut werden.

def finde_groessten_tank(dexpi_xml):
    try:
        # Parsen der DEXPI-Datei
        tree = ET.parse(dexpi_xml)
        root = tree.getroot()
    except ET.ParseError as e:
        # Fehlerausgabe, wenn das Parsen fehlschlägt
        print("Fehler beim Parsen des XML-Dokuments:", e)
        return None, None

    # Initialisierung von Variablen für den größten Tank und dessen Kapazität
    groesster_tank = None
    max_capacity = 0

    # Iterieren über alle 'Equipment'-Elemente im XML-Dokument
    for tank in root.iter('Equipment'):
        if tank.attrib.get('ComponentClass') == 'Tank':
            # Suche nach dem Attribut 'NominalCapacity(Volume)' im aktuellen Tank
            capacity_attr = tank.find(".//
 ↪GenericAttribute[@Name='NominalCapacity(Volume)']")
```

```
            # Überprüfen, ob das Attribut gefunden wurde und einen␣
↪'Value'-Schlüssel hat
            if capacity_attr is not None and 'Value' in capacity_attr.attrib:
                # Extrahieren der Kapazität und vergleichen mit der bisherigen␣
↪maximalen Kapazität
                capacity = float(capacity_attr.attrib['Value'])
                if capacity > max_capacity:
                    max_capacity = capacity
                    groesster_tank = tank

    # Wenn ein größerer Tank gefunden wurde
    if groesster_tank is not None:
        # Extrahieren der Tag-Bezeichnung des größten Tanks
        tank_tag = groesster_tank.attrib['TagName']

        # Extrahieren der Beschreibungen der Kammern im größten Tank
        tank_beschreibungen = [chamber.find(".//
↪GenericAttribute[@Name='ChamberFunctionAssignmentClass']").attrib.get('Value',␣
↪'') for chamber in groesster_tank.iter('Equipment') if chamber.attrib.
↪get('ComponentClass') == 'Chamber']

        return groesster_tank, tank_tag, tank_beschreibungen
    else:
        # None, wenn kein größerer Tank gefunden wurde
        return None, None

# Aufruf mit XML-Dokument
groesster_tank, tank_tag, tank_beschreibungen = finde_groessten_tank(dexpi_xml)

# Ausgabe der Ergebnisse
#print("Größter Tank:", tank_tag)
#print("Beschreibungen der Kammern:", tank_beschreibungen)

def setze_anlagenart(tank_beschreibungen):
    # Überprüfen, ob 'Processing Chamber' in den Beschreibungen vorkommt
    if "Processing" in tank_beschreibungen:
        return "HBV-Anlage"
    # Überprüfen, ob 'Storage Chamber' in den Beschreibungen vorkommt
    elif "Storing" in tank_beschreibungen:
        return "Tanklager"
    else:
        # Fehlermeldung, wenn keine oder beide Beschreibungen vorhanden sind
        print("Fehler: Ungültige Anlagenkonfiguration. Anlagenart manuell␣
↪eintragen.")
        return None

# Beispielaufruf
```

```
anlagenart = setze_anlagenart(tank_beschreibungen)

# Ausgabe der Ergebnisse
print(anlagenart)
```

```
HBV-Anlage
```

Angaben zu den wassergefährdenden Stoffen in der Anlage

[30]:
```
### Ziel: Extrahieren der relevanten Stoffmengen ###
    # Relevant sind alle Mengen mit welchen in Behältern umgegangen wird
        # siehe: Extrahieren der relevanten Volumina von den Tanks und der Tank-Informationen
    # Relevant sind auch alle Mengen die im Fehlerfall zusätzlich von Pumpen in die Anlage gefördert werden können
        # siehe: Extrahieren der relevanten Volumina von den Pumpen
```

[31]:
```
# Stofftabelle mit Infos zu den Stoffen, da diese in Dexpi nicht hinterlegt sind
# Es handelt sich hierbei um Stoffcodes
stoff_tabelle = {
    "MNc": ("flüssig", "3"),
    "MNb": ("flüssig", "3"),
    "WKa": ("flüssig", "1"),
    "WKb": ("flüssig", "1")
    # Weitere Einträge, falls benötigt
}
```

[32]:
```
### Ziel: Extrahieren der relevanten Volumina von den Pumpen ###

def extrahiere_stoff_bezeichnung_pumpe(dexpi_xml, nozzle_id, stoff_tabelle):
    try:
        tree = ET.parse(dexpi_xml)
        root = tree.getroot()
    except ET.ParseError as e:
        print(f"Fehler beim Parsen des XML: {e}")
        return None, None, None  # Rückgabewerte

    for piping_network_segment in root.iter('PipingNetworkSegment'):
        for connection in piping_network_segment.iter('Connection'):
            to_id = connection.get("ToID")
            if to_id == nozzle_id:
                for generic_attributes in piping_network_segment.iter('GenericAttributes'):
                    if generic_attributes.get("Set") == "DexpiAttributes":
                        for generic_attribute in generic_attributes.iter('GenericAttribute'):
                            if generic_attribute.get("Name") == "FluidCodeAssignmentClass":
```

```
                                fluid_code_assignment_class = generic_attribute.
↪get("Value")
                                # Annahme: Aggregatzustand und WGK können aus␣
↪der Tabelle extrahiert werden.
                                # Stoffinformationen können nicht aus dem RI␣
↪kommen, sondern müssen an andere Stelle geführt werden.
                                aggregatzustand, wgk = stoff_tabelle.
↪get(fluid_code_assignment_class, ("", ""))
                                return fluid_code_assignment_class,␣
↪aggregatzustand, wgk

    print(f"Keine Fluid Code Assignment Class für die ToID {nozzle_id} gefunden.
↪")
    return None, None, None

def find_specific_pumps(dexpi_xml, stoff_tabelle, pump_tags):
    pumpen_data = []

    try:
        tree = ET.parse(dexpi_xml)
        root = tree.getroot()
    except ET.ParseError as e:
        print("Fehler beim Parsen des XML-Dokuments:", e)
        return

    for pump_tag in pump_tags:
        # Suche nach Pumpe mit dem angegebenen Tag
        pump = root.find(f".//Equipment[@TagName='{pump_tag}']")

        if pump is not None and 'Pump' in pump.attrib.get('ComponentClass', ''):
            volume_flow_rate_attr = pump.find(".//
↪GenericAttribute[@Name='DesignVolumeFlowRate']")

            if volume_flow_rate_attr is not None and 'Value' in␣
↪volume_flow_rate_attr.attrib:
                pumpe_stoff_volumen = round(float(volume_flow_rate_attr.
↪attrib['Value']) / 6) # teilen durch 6, relevant ist der Volumenstrom in 10min␣
↪nicht in 1h

                nozzle_id = pump.find(".//Nozzle[@TagName='N1']")
                if nozzle_id is not None:
                    nozzle_id = nozzle_id.attrib.get('ID', '')

                    # Extrahiere Stoffbezeichnung, Aggregatzustand und WGK
                    pumpe_stoff_bezeichnung, pumpe_aggregatzustand, pumpe_wgk =␣
↪extrahiere_stoff_bezeichnung_pumpe(dexpi_xml, nozzle_id, stoff_tabelle)
```

```
                    if pumpe_stoff_bezeichnung:
                        pumpen_data.append({
                            "pumpe_nummer": pump_tag,
                            "pumpe_volumenstrom": pumpe_stoff_volumen,
                            "pumpe_stoff": pumpe_stoff_bezeichnung,
                            "pumpe_aggregatzustand": pumpe_aggregatzustand,
                            "pumpe_wgk": pumpe_wgk
                        })

    return pumpen_data

# Benutzereingabe
pump_tags_to_search = input("Welche Pumpen sollen beim Anlagenvolumen␣
↪berücksichtigt werden? Geben Sie die Pumpen-Tags (z.B. P1234) ein␣
↪(Mehrfacheingabe getrennt durch Leerzeichen): ").split()

# Beispielaufruf
specific_pumps_data = find_specific_pumps(dexpi_xml, stoff_tabelle,␣
↪pump_tags_to_search)

for pumpe in specific_pumps_data:
    print(f'{pumpe["pumpe_nummer"]}, {pumpe["pumpe_stoff"]},␣
↪{pumpe["pumpe_aggregatzustand"]}, {pumpe["pumpe_volumenstrom"]},␣
↪{pumpe["pumpe_wgk"]}')
```

```
P4712, MNb, flüssig, 33, 3
```

[33]:
```
### Ziel: Extrahieren der relevanten Volumina von den Tanks und der␣
↪Tank-Informationen ###

def extrahiere_stoffinformationen(tank_stoff, stoff_tabelle):
    default_info = ("unbekannt", "unbekannt")  # Standardwerte, wenn der Stoff␣
↪nicht in der Tabelle gefunden wird
    return stoff_tabelle.get(tank_stoff, default_info)

def extrahiere_tanks(dexpi_xml):
    root = ET.parse(dexpi_xml)
    tanks = []

    for equipment in root.findall(".//Equipment[@ComponentClass='Tank']"):
        tag_name = equipment.get('TagName', '')

        if not tag_name.startswith('T'):
            continue

        nominal_capacity_element = equipment.find(".//
↪GenericAttribute[@Name='NominalCapacity(Volume)']")
```

```
        if nominal_capacity_element is not None:
            nominal_capacity_str = nominal_capacity_element.get('Value', '')

            try:
                nominal_capacity = round(float(nominal_capacity_str))
            except ValueError:
                print(f"Fehler: Nennvolumen für Tank mit Tanknummer '{tag_name}'␣
↪ist keine Zahl.")
                continue
        else:
            print(f"Fehler: Kein Nennvolumen für Tank mit Tanknummer␣
↪'{tag_name}' vorhanden.")
            continue

        fluid_code_element = equipment.find(".//
↪GenericAttribute[@Name='FluidCodeAssignmentClass']")

        if fluid_code_element is not None:
            fluid_name = fluid_code_element.get('Value', '')
        else:
            print(f"Fehler: Kein FluidCodeAssignmentClass für Tank mit␣
↪Tanknummer '{tag_name}' vorhanden.")
            continue

        chambers = equipment.findall(".//Equipment[@ComponentClass='Chamber']")

        chamber_function = "einwandig"
        chamber_materials = set()

        for chamber in chambers:
            chamber_function_value = chamber.find(".//
↪GenericAttribute[@Name='ChamberFunctionAssignmentClass']").get('Value', '')

            if chamber_function_value in ['Processing', 'Storing']:
                chamber_material_element = chamber.find(".//
↪GenericAttribute[@Name='MaterialOfConstructionCodeAssignmentClass']")
                if chamber_material_element is not None:
                    chamber_material = chamber_material_element.get('Value', '')
                    chamber_materials.add(chamber_material)
                else:
                    print(f"Fehler: Kein␣
↪MaterialOfConstructionCodeAssignmentClass für Kammer im Tank mit Tanknummer␣
↪'{tag_name}' vorhanden.")
                    continue
```

```
                if chamber_function_value == "leakage detection":
                    chamber_function = "doppelwandig"

        if len(chamber_materials) == 0:
            print(f"Warnung: Keine Kammern mit den Funktionen 'Processing' oder␣
↪'Storage' für Tank mit Tanknummer '{tag_name}' gefunden.")
        elif len(chamber_materials) > 1:
            print(f"Warnung: Mehrere Kammern mit den Funktionen 'Processing'␣
↪oder 'Storage' für Tank mit Tanknummer '{tag_name}' gefunden. Es wird nur das␣
↪Material der ersten Kammer verwendet.")

        engineering_standard_element = equipment.find(".//
↪GenericAttribute[@Name='EngineeringStandardClass']")
        engineering_standard = engineering_standard_element.get('Value') if␣
↪engineering_standard_element is not None else None

        # Extrahiere Stoffinformationen mit der neuen Funktion
        tank_aggregatzustand, tank_wgk =␣
↪extrahiere_stoffinformationen(fluid_name, stoff_tabelle)

        tank_data = {
            'tank_nummer': tag_name,
            'tank_volumen': nominal_capacity,
            'tank_stoff': fluid_name,
            'tank_aggregatzustand': tank_aggregatzustand,
            'tank_wgk': tank_wgk,
            'tank_wandig': chamber_function,
            'tank_material': chamber_materials.pop() if chamber_materials else␣
↪None,
            'tank_zulassung': engineering_standard,
        }
        tanks.append(tank_data)

    return tanks

# Beispielaufruf
tanks = extrahiere_tanks(dexpi_xml)

for tank in tanks:
    print(f'{tank["tank_nummer"]}, {tank["tank_stoff"]},␣
↪{tank["tank_aggregatzustand"]}, {tank["tank_wgk"]}, {tank["tank_wandig"]},␣
↪{tank["tank_volumen"]}, {tank["tank_material"]}, , {tank["tank_zulassung"]}')
```

```
T4750, MNb, flüssig, 3, einwandig, 22, 1.4306, , AD2000
```

Angaben zur Gefährdungsstufe

[34]:
```
### Ziel: Ermittlung des maßgebenden Anlagenvolumens ###

anlagen_volumen = pumpe["pumpe_volumenstrom"] + tank["tank_volumen"]

print('Maßgebendes Anlagenvolumen (m3):', anlagen_volumen)
```

```
Maßgebendes Anlagenvolumen (m3): 55
```

[35]:
```
### Ziel: Ermittlung der maßgebenden AnlagenWGK ###

stoffe_in_anlage = [
    {"name": pumpe["pumpe_stoff"], "wgk": pumpe["pumpe_wgk"], "volumen":␣
↪pumpe["pumpe_volumenstrom"]},
    {"name": tank["tank_stoff"], "wgk": tank["tank_wgk"], "volumen": tank.
↪get("tank_volumenstrom", 0)},
    # Weitere Stoffe
]

# Sortieren der Stoffe nach aufsteigender WGK
sortierte_stoffe = sorted(stoffe_in_anlage, key=lambda x: x["wgk"])

# Initialisieren der maßgebende WGK mit der niedrigsten WGK
anlagen_wgk = sortierte_stoffe[0]["wgk"]

# Überprüfen, ob die Mischungsregel angewendet werden muss
if any(stoff["volumen"] > 0 for stoff in stoffe_in_anlage):
    gesamtvolumen = sum(stoff["volumen"] for stoff in stoffe_in_anlage)

    for stoff in sortierte_stoffe:
        anteil_stoff = stoff["volumen"] / gesamtvolumen
        if anteil_stoff >= 0.03:
            anlagen_wgk = stoff["wgk"]
            print(f"Die maßgebende WGK ist {anlagen_wgk}, da der Anteil der␣
↪Stoffe mit {stoff['wgk']} größer/gleich 3% ist.")
            break

# Ausgabe der Ergebnisse
print(f'Maßgebende WGK: {anlagen_wgk}')
```

```
Die maßgebende WGK ist 3, da der Anteil der Stoffe mit 3 größer/gleich 3% ist.
Maßgebende WGK: 3
```

[36]:
```
### Ziel: Ermittlung der Gefährdungsstufe der Anlage ###

def ermittle_gefaehrdungsstufe(anlagen_wgk, anlagen_volumen):
    # Konvertierung zu Integer, falls die Werte als Strings vorliegen
    anlagen_wgk = int(anlagen_wgk)
    anlagen_volumen = int(anlagen_volumen)
```

```
    #print("In der Funktion - Wert von anlagen_wgk:", anlagen_wgk)
    #print("In der Funktion - Wert von anlagen_volumen:", anlagen_volumen)

    if anlagen_wgk == 1:
        if anlagen_volumen <= 0.22:
            return "keine Anlage"
        elif 0.22 < anlagen_volumen <= 1:
            return "A"
        elif 1 < anlagen_volumen <= 10:
            return "A"
        elif 10 < anlagen_volumen <= 100:
            return "A"
        elif 100 < anlagen_volumen <= 1000:
            return "B"
        else:
            return "C"
    elif anlagen_wgk == 2:
        if anlagen_volumen <= 0.22:
            return "keine Anlage"
        elif 0.22 < anlagen_volumen <= 1:
            return "A"
        elif 1 < anlagen_volumen <= 10:
            return "B"
        elif 10 < anlagen_volumen <= 100:
            return "C"
        elif 100 < anlagen_volumen <= 1000:
            return "D"
        else:
            return "D"
    elif anlagen_wgk == 3:
        if anlagen_volumen <= 0.22:
            return "keine Anlage"
        elif 0.22 < anlagen_volumen <= 1:
            return "B"
        elif 1 < anlagen_volumen <= 10:
            return "C"
        elif 10 < anlagen_volumen <= 100:
            return "D"
        elif 100 < anlagen_volumen <= 1000:
            return "D"
        else:
            return "D"
    else:
        return "Ungültige Wassergefährdungsklasse"
```

```
anlagen_gefaehrdungsstufe = ermittle_gefaehrdungsstufe(anlagen_wgk,␣
 ↪anlagen_volumen)
print("Die Gefährdungsstufe der Anlage ist:", anlagen_gefaehrdungsstufe)
```

```
Die Gefährdungsstufe der Anlage ist: D
```

Angaben zu Behältern

[37]:
```
### siehe: Extrahieren der relevanten Volumina von den Tanks und der␣
 ↪Tank-Informationen ###
```

Zusatz: KI-Anwendung Material

[38]:
```
# Importieren der benötigten Module
# import os

# Extrahieren des OpenAI-API-Schlüssels aus den Umgebungsvariablen oder direkt␣
 ↪eintragen zwischen ""
# openai_api_key = os.getenv("OPENAI_API_KEY", "")

# Jetzt ist der OpenAI-API-Schlüssel verfügbar und kann im Code verwendet␣
 ↪werden, der print unten nur zum Test
# print(f"OpenAI API Key: {openai_api_key}")

# Importieren der benötigten Module
# from langchain_openai import ChatOpenAI

# Initialisieren des ChatOpenAI-Objekts
# Mit der temperature kann man "herum experimentieren"
# chat = ChatOpenAI(model="gpt-3.5-turbo-1106", temperature=1,␣
 ↪openai_api_key=openai_api_key)
```

[39]:
```
# Definition der Werkstoffbezeichnung, hier kein Automatismus
# Es können unterschiedliche Werkstoffe getestet werden
#werkstoff_bezeichnung = "1.4571"

# Verwendung der Variable in der Chat-Anfrage
#from langchain_core.messages import HumanMessage

#chat.invoke(
#    [
#        HumanMessage(
```

```
#            content=f'Setze die Sprache auf Deutsch. Du bist ein Assistent. Du␣
 ↪hast eine Aufgabe. Ich gebe dir einen Werkstoff. Deine Aufgabe besteht darin,␣
 ↪zuzuordnen, ob es sich beim gegebenen Werkstoff um ein Kunststoff, ein Metall␣
 ↪oder einen anderes Material handelt. Versuche den gegebenen Werkstoff in eine␣
 ↪der drei Kategorien: "Metall", "Kunststoff", "anderes Material" einzustufen.␣
 ↪Antworte nur: "Metall", "Kunststoff", "anderes Material".  Der gesuchte␣
 ↪Werkstoff ist: {werkstoff_bezeichnung}. Falls es sich dabei gar nicht um einen␣
 ↪Werkstoff im engeren Sinn handelt sage "Error"'
#        )
#    ]
#)
```

Angaben zu den Rohrleitungen

[40]:
```
### Ziel: Alle unterschiedlichen Rohrklassen aus dem DEXPI-RI extrahieren ###
import xml.etree.ElementTree as ET

def extract_unique_piping_class_code_assignment_class(dexpi_xml):
    tree = ET.parse(dexpi_xml)
    root = tree.getroot()

    piping_class_code_values = set()

    # Durchsuche alle PipingNetworkSystem-Tags
    for piping_network_system in root.findall('.//PipingNetworkSystem'):
        # Durchsuche alle GenericAttribute-Tags unter GenericAttributes
        for generic_attribute in piping_network_system.findall('.//
 ↪GenericAttributes/GenericAttribute'):
            # Überprüfe, ob das Attribut den gewünschten Namen hat
            if generic_attribute.get('Name') == 'PipingClassCodeAssignmentClass':
                # Extrahiere den Wert des Attributs und füge ihn zum Set hinzu
                value = generic_attribute.get('Value')
                piping_class_code_values.add(value)

    return piping_class_code_values

# Beispielaufruf

unique_piping_class_code_values =␣
 ↪extract_unique_piping_class_code_assignment_class(dexpi_xml)

# Ausgabe der eindeutigen Werte
print("Eindeutige PipingClassCodeAssignmentClass Werte:")
for value in unique_piping_class_code_values:
    print(value)
```

```
Eindeutige PipingClassCodeAssignmentClass Werte:
```

```
75HG12
73HG12
73KH12
75HB13
```

[41]:
```
### Ziel: Alle vorhandenen Rohrklassen mit Eigenschaften definieren. Im DEXPI-RI␣
↪so nicht vorhanden. ###
    # Hier werden willkürliche Werte definiert

class Rohrklasse:
    def __init__(self, kennnummer, verlegungsart, material, fertigungsnorm):
        self.kennnummer = kennnummer
        self.verlegungsart = verlegungsart
        self.material = material
        self.fertigungsnorm = fertigungsnorm

# Beispiel-Rohrklassen
rk73KH12 = Rohrklasse("73KH12", "oberirdisch", "1.4404", "AD2000")
rk75HG12 = Rohrklasse("75HG12", "oberirdisch", "1.4435", "AD2000")
rk73HG12 = Rohrklasse("73HG12", "oberirdisch", "1.4571", "AD2000")
rk75HB13 = Rohrklasse("75HB13", "oberirdisch", "1.4303", "AD2000")

# Liste der Rohrklassen
rohrklassen_liste = [rk73KH12, rk75HG12, rk73HG12, rk75HB13]


# Beispiel-Zugriffe auf Eigenschaften
#print(rk73KH12.kennnummer)  # Ausgabe: 73KH12
#print(rk75HG12.material)    # Ausgabe: 1.4435
#print(rk73HG12.fertigungsnorm)  # Ausgabe: ISO 4427
```

[42]:
```
### Ziel: Beurteilung der relevanten Rohrleitungen ###
    # Eine PipingNetworkSystem soll als eine Rohrleitung gewertet werden
    # Im DEXPI-Viewer wurden 11 Rohrleitungen identifiziert: https://
↪dexpi-viewer.model-broker.com/
    # Durch zählen auf PDF R&I werden auch 11 Rohrleitungen identifiziert

# Parsen der XML-Daten
tree = ET.parse(dexpi_xml)
root = tree.getroot()

# Zählen der Piping Network Systems
piping_network_system_count_all = len(root.findall(".//PipingNetworkSystem"))

# Ausgabe der Gesamtanzahl an Piping Network Systems aus der Quelldatei
print(f"Gesamtanzahl Piping Network Systems: {piping_network_system_count_all}")
```

```
# Iterieren über alle Piping Network Systems und Extrahieren der IDs und TagNames
for piping_system in root.iter('PipingNetworkSystem'):
    # Extrahieren des TagNames
    piping_system_tagname = piping_system.attrib.get('TagName', '')

    # Prüfen, ob der TagName aus genau 5 Ziffern besteht
    if piping_system_tagname.isdigit() and len(piping_system_tagname) == 5:
        # Extrahieren der ID
        piping_system_id = piping_system.attrib.get('ID', '')

        # Ausgabe der ID und des TagNames
        print(f"Piping Network System ID: {piping_system_id}, TagName:␣
 ↪{piping_system_tagname}")
    else:
        # Ausgabe für ungültigen Tag
        print(f"Piping Network System ID: {piping_system.attrib.get('ID', '')},␣
 ↪Kein gültiger Tag")
```

```
Gesamtanzahl Piping Network Systems: 14
Piping Network System ID: SY5D814AA7FC0A47F5921830DCF5A4A8EB, TagName: 47131
Piping Network System ID: SYF0E638DCE25148A0ADE8BE802F8AFAE3, TagName: 47130
Piping Network System ID: SY11933F04339546E2BB3FAE0C30B7D6AA, TagName: 47122
Piping Network System ID: SY043C45C1F8ED42C092CDCC5BF3456CB4, TagName: 47121
Piping Network System ID: SY708551978F43471FA98F8244FC0A913F, TagName: 47140
Piping Network System ID: SYAEBFFC9AB2C94F5993EA8C4D89EC1125, TagName: 47126
Piping Network System ID: SY1F37391ED2314597A4A128606FDD1B01, TagName: 47125
Piping Network System ID: SY8B12D6816E9F432695222500F08A648F, TagName: 47124
Piping Network System ID: SY81C7023B2F9846A4B5AB695A2DDB2140, Kein gültiger Tag
Piping Network System ID: SYE2E35A8ADC204BBEA377E7EBFF661EB6, Kein gültiger Tag
Piping Network System ID: SYAC2C1AB5FF9F4AE59D88F023714689A1, Kein gültiger Tag
Piping Network System ID: SY1D329D8EFB14468CB0F4807D9312F472, TagName: 47123
Piping Network System ID: SY31A53D930F7F45F487B9F03340ECF678, TagName: 47127
Piping Network System ID: SY823ED83CA9884DDCB4FC998AF641D8A2, TagName: 47141
```

[43]:
```
### Ziel: Kategorisierung und Aufsummierung der relevanten Rohrleitungen ###
### Alternativ: Bearbeitung XML und andere Klassen bestimmen. Z.B.: SUCTION␣
 ↪PIPING SYSTEM (http://data.posccaesar.org/rdl/RDS426402071)

# Parsen der XML-Daten
tree = ET.parse(dexpi_xml)
root = tree.getroot()

# Piping Network Systems, die ausgeschlossen werden sollen, da Werte nicht␣
 ↪sinnvoll erscheinen (kein gültiger Tag usw.)
# Manueller Eingriff aufgrund mangelnder Datenqualität in Quelldatei
excluded_system_ids = ['SYAC2C1AB5FF9F4AE59D88F023714689A1',␣
 ↪'SYE2E35A8ADC204BBEA377E7EBFF661EB6', 'SY81C7023B2F9846A4B5AB695A2DDB2140']
```

```
# Zählen der verbleibenden  Piping Network Systems (mit Ausschluss)
piping_network_systems = [piping_system for piping_system in root.
 ↪iter('PipingNetworkSystem') if piping_system.attrib.get('ID') not in␣
 ↪excluded_system_ids]
piping_network_system_count = len(piping_network_systems)

# Initialisierung von Zählern für die manuell zugewiesenen Kategorien
rl_2_anzahl = 0
rl_3_anzahl = 0
rl_4_anzahl = 0
rl_1_anzahl = 0

# Initialisierung von Listen für die Tags pro Kategorie
einwandig_standard_tags = []
einwandig_saugleitung_tags = []
einwandig_schutzrohr_tags = []
doppelwandig_tags = []

def kategorisiere_rohrleitungen(piping_system):
    # Sucht nach dem GenericAttribute-Element mit dem Attribut 'Name' gleich␣
 ↪'JacketedPipeSpecialization'
    jacketed_pipe_specialization = piping_system.find(".//
 ↪GenericAttribute[@Name='JacketedPipeSpecialization']")

    # Überprüft, ob das Element gefunden wurde
    if jacketed_pipe_specialization is not None:
        # Extrahiert den Wert des 'Value'-Attributs des gefundenen Elements
        specialization_value = jacketed_pipe_specialization.attrib.get('Value',␣
 ↪'')

        # Überprüft die spezialisierte Eigenschaft der Rohrleitung
        if specialization_value.lower() == 'unjacketedpipe':
            # Fordert den Benutzer auf, eine Kategorie für die Rohrleitung␣
 ↪einzugeben
            category_input = input(f"Bitte Kategorie für Rohrleitung␣
 ↪{piping_system.attrib.get('TagName')} (Enter für Einwandig Standard, 'saug'␣
 ↪für Einwandig Saugleitung, 'schutz' für Einwandig Schutzrohr): ").strip().
 ↪lower()

            # Überprüft die Benutzereingabe und gibt die entsprechende Kategorie␣
 ↪zurück
            if category_input == '':
                return 'Einwandig Standard'
            elif category_input == 'saug':
                return 'Einwandig Saugleitung'
```

```
            elif category_input == 'schutz':
                return 'Einwandig Schutzrohr'
            else:
                # Falls die Eingabe ungültig ist, wird eine Standardkategorie␣
↪ausgewählt und eine Meldung ausgegeben
                print(f"Ungültige Eingabe: {category_input}. Einwandig Standard␣
↪wird ausgewählt.")
                return 'Einwandig Standard'
        elif specialization_value.lower() == 'jacketedpipe':
            # Gibt die Kategorie 'Doppelwandig' zurück, wenn die spezialisierte␣
↪Eigenschaft 'jacketedpipe' ist
            return 'Doppelwandig'
        else:
            # Falls die spezialisierte Eigenschaft ungültig ist, wird eine␣
↪entsprechende Meldung ausgegeben
            return "Ungültige Piping Network System-Kategorie gefunden."
    else:
        # Falls kein unjacketed Piping Network System gefunden wird, wird eine␣
↪entsprechende Meldung ausgegeben
        return "Kein unjacketed Piping Network System gefunden."


# Verwendung der Funktion in Ihrer Schleife
for piping_system in piping_network_systems:
    result = kategorisiere_rohrleitungen(piping_system)

    # Zählvariablen und Listen aktualisieren
    if result == 'Einwandig Standard':
        rl_2_anzahl += 1
        einwandig_standard_tags.append(piping_system.attrib.get('TagName'))
    elif result == 'Einwandig Saugleitung':
        rl_3_anzahl += 1
        einwandig_saugleitung_tags.append(piping_system.attrib.get('TagName'))
    elif result == 'Einwandig Schutzrohr':
        rl_4_anzahl += 1
        einwandig_schutzrohr_tags.append(piping_system.attrib.get('TagName'))
    elif result == 'Doppelwandig':
        rl_1_anzahl += 1
        doppelwandig_tags.append(piping_system.attrib.get('TagName'))

    # Ergebnis ausgeben
    print(f"Kategorie für Rohrleitung {piping_system.attrib.get('TagName')}:␣
↪{result}")

# Ausgabe der Ergebnisse
print("\n# Ausgabe der Anzahl der Rohrleitungen")
```

```
print(f"Anzahl Gesamt: {piping_network_system_count}")
print(f"Anzahl Doppelwandig: {rl_1_anzahl}")
print(f"Anzahl Einwandig Standard: {rl_2_anzahl}")
print(f"Anzahl Einwandig Saugleitung: {rl_3_anzahl}")
print(f"Anzahl Einwandig Schutzrohr: {rl_4_anzahl}")

# Ausgabe der Tags pro Kategorie
print("\n# Ausgabe der Tag-Zuordnung")
print(f"Tags für Doppelwandig: {doppelwandig_tags}")
print(f"Tags für Einwandig Standard: {einwandig_standard_tags}")
print(f"Tags für Einwandig Saugleitung: {einwandig_saugleitung_tags}")
print(f"Tags für Einwandig Schutzrohr: {einwandig_schutzrohr_tags}")
```

```
Kategorie für Rohrleitung 47131: Einwandig Standard
Kategorie für Rohrleitung 47130: Einwandig Standard
Kategorie für Rohrleitung 47122: Einwandig Standard
Kategorie für Rohrleitung 47121: Einwandig Saugleitung
Kategorie für Rohrleitung 47140: Einwandig Standard
Kategorie für Rohrleitung 47126: Einwandig Standard
Kategorie für Rohrleitung 47125: Einwandig Standard
Kategorie für Rohrleitung 47124: Einwandig Standard
Kategorie für Rohrleitung 47123: Einwandig Standard
Kategorie für Rohrleitung 47127: Einwandig Standard
Kategorie für Rohrleitung 47141: Einwandig Standard

# Ausgabe der Anzahl der Rohrleitungen
Anzahl Gesamt: 11
Anzahl Doppelwandig: 0
Anzahl Einwandig Standard: 10
Anzahl Einwandig Saugleitung: 1
Anzahl Einwandig Schutzrohr: 0

# Ausgabe der Tag-Zuordnung
Tags für Doppelwandig: []
Tags für Einwandig Standard: ['47131', '47130', '47122', '47140', '47126',
'47125', '47124', '47123', '47127', '47141']
Tags für Einwandig Saugleitung: ['47121']
Tags für Einwandig Schutzrohr: []
```

[44]:
```
# Initialisierung der Listen für die Piping Class Codes
rl_1_rk = []    #doppelwandig
rl_2_rk = []    #eiwandig
rl_3_rk = []    #einwandig Saugleitung
rl_4_rk = []    #einwandig Schutzrohr


def sammle_rohrklassen(tag_list, piping_network_systems):
```

```
    result_list = []

    for piping_system in piping_network_systems:
        tag_name = piping_system.attrib.get('TagName')

        # Nur für Systeme in den ausgewählten Tags
        if tag_name in tag_list:
            # Prüfen der PipingClassCodeAssignmentClass
            class_assignment = piping_system.find(".//
↪GenericAttribute[@Name='PipingClassCodeAssignmentClass']")

            if class_assignment is not None:
                class_code = class_assignment.attrib.get('Value', '').strip()

                # Piping Class Code zur entsprechenden Liste hinzufügen
                result_list.append(class_code)

    return result_list

# Verwendung der Funktion
rl_1_rk = sammle_rohrklassen(doppelwandig_tags, piping_network_systems)
rl_2_rk = sammle_rohrklassen(einwandig_standard_tags, piping_network_systems)
rl_3_rk = sammle_rohrklassen(einwandig_saugleitung_tags, piping_network_systems)
rl_4_rk = sammle_rohrklassen(einwandig_schutzrohr_tags, piping_network_systems)

# Eindeutige Rohrklassen erhalten
rl_1_rk = list(set(rl_1_rk))
rl_2_rk = list(set(rl_2_rk))
rl_3_rk = list(set(rl_3_rk))
rl_4_rk = list(set(rl_4_rk))

# Ergebnisse ausgeben
print("Rohrklassen Doppelwandig:", rl_1_rk)
print("Rohrklassen Einwandig Standard:", rl_2_rk)
print("Rohrklassen Einwandig Saugleitung:", rl_3_rk)
print("Rohrklassen Einwandig Schutzrohr:", rl_4_rk)
```

```
Rohrklassen Doppelwandig: []
Rohrklassen Einwandig Standard: ['75HG12', '75HB13']
Rohrklassen Einwandig Saugleitung: ['75HB13']
Rohrklassen Einwandig Schutzrohr: []
```

[45]:
```
# Definition der Variablen
rl_1_oberirdisch = ""
rl_1_unterirdisch = ""
rl_1_material = []
rl_1_zulassung = []
```

```
rl_2_oberirdisch = ""
rl_2_unterirdisch = ""
rl_2_material = []
rl_2_zulassung = []

rl_3_oberirdisch = ""
rl_3_unterirdisch = ""
rl_3_material = []
rl_3_zulassung = []

rl_4_oberirdisch = ""
rl_4_unterirdisch = ""
rl_4_material = []
rl_4_zulassung = []
```

[46]:
```
# Initialisierung der Variablen für rl_1 bis rl_4
rl_oberirdisch = ["", "", "", ""]  # Liste für die Kennzeichnung, ob die␣
 ↪Verlegungsart oberirdisch ist
rl_unterirdisch = ["", "", "", ""]  # Liste für die Kennzeichnung, ob die␣
 ↪Verlegungsart unterirdisch ist
rl_material = [[], [], [], []]  # Liste für die Extraktion und Speicherung von␣
 ↪Materialinformationen
rl_zulassung = [[], [], [], []]  # Liste für die Extraktion und Speicherung von␣
 ↪Zulassungsinformationen

# Durchsuchen von rl_1 bis rl_4_rk und Ausgabe von Verlegungsart_oberirdisch,␣
 ↪Verlegungsart_unterirdisch, Material und Zulassung
for index, tag_list in enumerate([doppelwandig_tags, einwandig_standard_tags,␣
 ↪einwandig_saugleitung_tags, einwandig_schutzrohr_tags]):
    rl_rk = []  # Liste für die Rohrklassen des aktuellen Durchlaufs

    # Je nach Index den entsprechenden rl_rk zuweisen
    if index == 0:
        rl_rk = rl_1_rk
    elif index == 1:
        rl_rk = rl_2_rk
    elif index == 2:
        rl_rk = rl_3_rk
    elif index == 3:
        rl_rk = rl_4_rk

    # Durchsuchen von rl_rk und Ausgabe von Verlegungsart_oberirdisch,␣
 ↪Verlegungsart_unterirdisch, Material und Zulassung
    for code in rl_rk:
```

```
        passende_rohrklasse = next((rohrklasse for rohrklasse in␣
↪rohrklassen_liste if rohrklasse.kennnummer == code), None)

        if passende_rohrklasse:
            verlegungsart_wert = passende_rohrklasse.verlegungsart
            material_wert = passende_rohrklasse.material
            zulassung_wert = passende_rohrklasse.fertigungsnorm

            # Überprüfen Sie den Wert der Verlegungsart und setzen Sie die␣
↪Variablen entsprechend
            if verlegungsart_wert == "oberirdisch":
                rl_oberirdisch[index] = "X"  # Kennzeichnung für oberirdische␣
↪Verlegungsart
            elif verlegungsart_wert == "unterirdisch":
                rl_unterirdisch[index] = "X"  # Kennzeichnung für unterirdische␣
↪Verlegungsart

            # Speichern Sie das Material und die Zulassung in den entsprechenden␣
↪Listen
            rl_material[index].append(material_wert)
            rl_zulassung[index].append(zulassung_wert)
        else:
            print("Keine passende Rohrklasse gefunden für", code)

# Beispiel: Ausgabe der Ergebnisse für rl_1 bis rl_4
for i in range(4):
    print(f"rl_{i+1}_oberirdisch:", rl_oberirdisch[i])
    print(f"rl_{i+1}_unterirdisch:", rl_unterirdisch[i])
    print(f"rl_{i+1}_material:", rl_material[i])  # Liste mit allen Materialien
    print(f"rl_{i+1}_zulassung:", rl_zulassung[i])  # Liste mit allen Zulassungen

print(rl_material[1])
```

```
rl_1_oberirdisch:
rl_1_unterirdisch:
rl_1_material: []
rl_1_zulassung: []
rl_2_oberirdisch: X
rl_2_unterirdisch:
rl_2_material: ['1.4435', '1.4303']
rl_2_zulassung: ['AD2000', 'AD2000']
rl_3_oberirdisch: X
rl_3_unterirdisch:
rl_3_material: ['1.4303']
rl_3_zulassung: ['AD2000']
rl_4_oberirdisch:
rl_4_unterirdisch:
```

```
rl_4_material: []
rl_4_zulassung: []
['1.4435', '1.4303']
```

[47]:
```python
# Initialisierung der Variablen für rl_1 bis rl_4
rl_oberirdisch = ["", "", "", ""]  # Liste für die Kennzeichnung, ob die␣
 ↪Verlegungsart oberirdisch ist
rl_unterirdisch = ["", "", "", ""]  # Liste für die Kennzeichnung, ob die␣
 ↪Verlegungsart unterirdisch ist
rl_material = [[], [], [], []]  # Liste für die Extraktion und Speicherung von␣
 ↪Materialinformationen
rl_zulassung = [[], [], [], []]  # Liste für die Extraktion und Speicherung von␣
 ↪Zulassungsinformationen

# Durchsuchen von rl_1 bis rl_4_rk und Ausgabe von Verlegungsart_oberirdisch,␣
 ↪Verlegungsart_unterirdisch, Material und Zulassung
for index, tag_list in enumerate([doppelwandig_tags, einwandig_standard_tags,␣
 ↪einwandig_saugleitung_tags, einwandig_schutzrohr_tags]):
    rl_rk = []  # Liste für die Rohrklassen des aktuellen Durchlaufs

    # Je nach Index den entsprechenden rl_rk zuweisen
    if index == 0:
        rl_rk = rl_1_rk
    elif index == 1:
        rl_rk = rl_2_rk
    elif index == 2:
        rl_rk = rl_3_rk
    elif index == 3:
        rl_rk = rl_4_rk

    # Durchsuchen von rl_rk und Ausgabe von Verlegungsart_oberirdisch,␣
 ↪Verlegungsart_unterirdisch, Material und Zulassung
    for code in rl_rk:
        passende_rohrklasse = next((rohrklasse for rohrklasse in␣
 ↪rohrklassen_liste if rohrklasse.kennnummer == code), None)

        if passende_rohrklasse:
            verlegungsart_wert = passende_rohrklasse.verlegungsart
            material_wert = passende_rohrklasse.material
            zulassung_wert = passende_rohrklasse.fertigungsnorm

            # Überprüfen Sie den Wert der Verlegungsart und setzen Sie die␣
 ↪Variablen entsprechend
            if verlegungsart_wert == "oberirdisch":
                rl_oberirdisch[index] = "X"  # Kennzeichnung für oberirdische␣
 ↪Verlegungsart
            elif verlegungsart_wert == "unterirdisch":
```

```
                rl_unterirdisch[index] = "X"  # Kennzeichnung für unterirdische␣
↪Verlegungsart

            # Speichern Sie das Material und die Zulassung in den entsprechenden␣
↪Listen
            rl_material[index].append(material_wert)
            rl_zulassung[index].append(zulassung_wert)
        else:
            print("Keine passende Rohrklasse gefunden für", code)

# Beispiel: Ausgabe der Ergebnisse für rl_1 bis rl_4
for i in range(4):
    print(f"rl_{i+1}_oberirdisch:", rl_oberirdisch[i])
    print(f"rl_{i+1}_unterirdisch:", rl_unterirdisch[i])
    print(f"rl_{i+1}_material:", ', '.join(list(set(map(str, rl_material[i])))))␣
↪ # Eindeutige Materialien ohne Klammern und Anführungszeichen
    print(f"rl_{i+1}_zulassung:", ', '.join(list(set(map(str,␣
↪rl_zulassung[i])))))  # Eindeutige Zulassungen ohne Klammern und␣
↪Anführungszeichen
```

```
rl_1_oberirdisch:
rl_1_unterirdisch:
rl_1_material:
rl_1_zulassung:
rl_2_oberirdisch: X
rl_2_unterirdisch:
rl_2_material: 1.4435, 1.4303
rl_2_zulassung: AD2000
rl_3_oberirdisch: X
rl_3_unterirdisch:
rl_3_material: 1.4303
rl_3_zulassung: AD2000
rl_4_oberirdisch:
rl_4_unterirdisch:
rl_4_material:
rl_4_zulassung:
```

Formular XML

[48]:
```
### Ziel: Erstellen eines Formblatts 6.2 im .xml-Format ###
# Die anlagenspezifischen Werte welche ins Formblatt eingetragen werden stammen␣
↪aus den oben ermittelten Variablen.
import os
from xml.dom import minidom

# XML-Deklaration für die Ausgabe
xml_declaration_str = '<?xml version="1.0" encoding="UTF-8"?>'
```

```python
# Erstellen des Wurzelelements formblatt_6_2
formblatt_6_2 = ET.Element('formblatt_6_2')

def create_table(parent, table_name, heading, column_names, data):
    table = ET.SubElement(parent, f'tabelle_{table_name}')
    table_heading = ET.SubElement(table, 'ueberschrift')
    table_heading.text = heading

    columns = ET.SubElement(table, 'spalten')
    for column_name in column_names:
        column = ET.SubElement(columns, 'spalte')
        column.text = column_name

    table_data = ET.SubElement(table, 'daten')
    data_row = ET.SubElement(table_data, 'zeile')
    for value in data:
        value_element = ET.SubElement(data_row, 'wert')
        value_element.text = value

# Hier können die oben bestimmten Variablen eingesetzt werden

# Kopftabelle
kopf_data = {
    'Antragsunterlage': 'Für immissionsschutzrechtliches Genehmigungsverfahren',
    'Anlage 1 / Formblatt 6.2': 'Detailangaben / Wassergefährdende Stoffe',
}

# Angaben zur Anlage
angaben_anlage_data = {
    'Anlagenart': anlagenart,
    'Anlagenbezeichnung': anlagenbezeichnung,
    'Anlagenumfang': anlagenumfang
}

# Angaben zu den wassergefährdenden Stoffen in der Anlage
angaben_stoffe_data = {
    "Stoffbezeichnung_Pumpe": pumpe.get("pumpe_stoff", ""),
    "Aggregatzustand_Pumpe": pumpe.get("pumpe_aggregatzustand", ""),
    "WGK_Pumpe": str(pumpe.get("pumpe_wgk", "")),
    "Volumen_Pumpe": str(pumpe.get("pumpe_volumen", "")),
    "Stoffbezeichnung_Tank": tank["tank_stoff"],
    "Aggregatzustand_Tank": tank["tank_aggregatzustand"],
    "WGK_Tank": str(tank["tank_wgk"]),
    "Volumen_Tank": str(tank["tank_volumen"]),
}
```

```
# Angaben zur Gefährdungsstufe
angaben_gefaehrdungsstufe_data = {
    'Maßgebendes Volumen': str(anlagen_volumen),
    'Maßgebende WGK': str(anlagen_wgk),
    'Gefährdungsstufe': anlagen_gefaehrdungsstufe
}

# Angaben zu den Behältern
angaben_behaelter_data = {
    'Tanknummer': tank["tank_nummer"],
    'enthaltener Stoff': tank["tank_stoff"],
    'einwandig / doppelwandig': tank["tank_wandig"],
    'Nennvolumen (m3)': str(tank["tank_volumen"]),
    'Material': str(tank["tank_material"]),
    'Zulassung': tank["tank_zulassung"]
}

# Angaben zu den doppelwandigen Rohrleitungen
angaben_rohrleitungen1_data = {
    'Bauart': 'Doppelwandig mit Leckanzeige',
    'oberirdisch / unterirdisch': rl_oberirdisch[0],
    'Anzahl': str(rl_1_anzahl),
    'Material': ', '.join(list(set(map(str, rl_material[0])))),
    'Zulassung': ', '.join(list(set(map(str, rl_zulassung[0])))),
}

# Angaben zu den einwandigen Rohrleitungen
angaben_rohrleitungen2_data = {
    'Bauart': 'Einwandig',
    'oberirdisch / unterirdisch': rl_oberirdisch[1],
    'Anzahl': str(rl_2_anzahl),
    'Material': ', '.join(list(set(map(str, rl_material[1])))),
    'Zulassung': ', '.join(list(set(map(str, rl_zulassung[1])))),
}

# Angaben zu den einwandigen Rohrleitungen als Saugleitung
angaben_rohrleitungen3_data = {
    'Bauart': 'Einwandig als Saugleitung',
    'oberirdisch / unterirdisch': rl_oberirdisch[2],
    'Anzahl': str(rl_3_anzahl),
    'Material': ', '.join(list(set(map(str, rl_material[2])))),
    'Zulassung': ', '.join(list(set(map(str, rl_zulassung[2])))),
}

# Angaben zu den einwandigen Rohrleitungen im Schutzrohr
angaben_rohrleitungen4_data = {
    'Bauart': 'Einwandig im Schutzrohr',
```

```
    'oberirdisch / unterirdisch': rl_oberirdisch[3],
    'Anzahl': str(rl_4_anzahl),
    'Material': ', '.join(list(set(map(str, rl_material[3])))),
    'Zulassung': ', '.join(list(set(map(str, rl_zulassung[3])))),
}


create_table(formblatt_6_2, 'kopf', 'Angaben zum Formblatt', kopf_data.keys(),␣
 ↪kopf_data.values())
create_table(formblatt_6_2, 'angaben_anlage', 'Angaben zur Anlage',␣
 ↪angaben_anlage_data.keys(), angaben_anlage_data.values())
create_table(formblatt_6_2, 'angaben_stoffe', 'Angaben zu den wassergefährdenden␣
 ↪Stoffen', angaben_stoffe_data.keys(), angaben_stoffe_data.values())
create_table(formblatt_6_2, 'angaben_gefaaehrdungsstufe', 'Ermittlung der␣
 ↪Gefährdungsstufe der Anlage nach §39 AwSV', angaben_gefaehrdungsstufe_data.
 ↪keys(), angaben_gefaehrdungsstufe_data.values())
create_table(formblatt_6_2, 'angaben_behaelter', 'Angaben zu den Behältern',␣
 ↪angaben_behaelter_data.keys(), angaben_behaelter_data.values())
create_table(formblatt_6_2, 'angaben_rohrleitungen1', 'Angaben zu den␣
 ↪doppelwandigen Rohrleitungen', angaben_rohrleitungen1_data.keys(),␣
 ↪angaben_rohrleitungen1_data.values())
create_table(formblatt_6_2, 'angaben_rohrleitungen2', 'Angaben zu den␣
 ↪einwandigen Rohrleitungen', angaben_rohrleitungen2_data.keys(),␣
 ↪angaben_rohrleitungen2_data.values())
create_table(formblatt_6_2, 'angaben_rohrleitungen3', 'Angaben zu den␣
 ↪einwandigen Rohrleitungen als Saugleitung', angaben_rohrleitungen3_data.
 ↪keys(), angaben_rohrleitungen3_data.values())
create_table(formblatt_6_2, 'angaben_rohrleitungen4', 'Angaben zu den␣
 ↪einwandigen Rohrleitungen im Schutzrohr', angaben_rohrleitungen4_data.keys(),␣
 ↪angaben_rohrleitungen4_data.values())

# Erstellen des ElementTree mit dem Wurzelelement
tree = ET.ElementTree(formblatt_6_2)

# Hinzufügen der XML-Deklaration und Korrektur der Ausgabe
xml_str = ET.tostring(formblatt_6_2, encoding='utf-8', xml_declaration=True).
 ↪decode()

# Drucken des XML-str mit Einzügen
xml_str = ET.tostring(formblatt_6_2, encoding='utf-8').decode()
xml_str = minidom.parseString(xml_str).toprettyxml(indent="   ")
print(xml_str)

# Pfad auf aktuelles Verzeichnis
current_directory_path_xml = os.path.join(os.getcwd(), "formular_6_2_auto.xml")
```

```python
# Speichern des XML-Codes in einer .xml-Datei in dem aktuellen Verzeichnis
with open(current_directory_path_xml, "w") as xml_file:
    xml_file.write(xml_str)

print(f"XML-Formular wurde in '{current_directory_path_xml}' gespeichert.")
```

```
<?xml version="1.0" ?>
<formblatt_6_2>
   <tabelle_kopf>
      <ueberschrift>Angaben zum Formblatt</ueberschrift>
      <spalten>
         <spalte>Antragsunterlage</spalte>
         <spalte>Anlage 1 / Formblatt 6.2</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>Für immissionsschutzrechtliches Genehmigungsverfahren</wert>
            <wert>Detailangaben / Wassergefährdende Stoffe</wert>
         </zeile>
      </daten>
   </tabelle_kopf>
   <tabelle_angaben_anlage>
      <ueberschrift>Angaben zur Anlage</ueberschrift>
      <spalten>
         <spalte>Anlagenart</spalte>
         <spalte>Anlagenbezeichnung</spalte>
         <spalte>Anlagenumfang</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>HBV-Anlage</wert>
            <wert>C01V01-HEX.EX03</wert>
            <wert>Kolbenpumpe, Rohrbündelwärmetauscher, Kreiselpumpe,
Rohrleitungssystem, Behälter, Plattenwärmetauscher</wert>
         </zeile>
      </daten>
   </tabelle_angaben_anlage>
   <tabelle_angaben_stoffe>
      <ueberschrift>Angaben zu den wassergefährdenden Stoffen</ueberschrift>
      <spalten>
         <spalte>Stoffbezeichnung_Pumpe</spalte>
         <spalte>Aggregatzustand_Pumpe</spalte>
         <spalte>WGK_Pumpe</spalte>
         <spalte>Volumen_Pumpe</spalte>
         <spalte>Stoffbezeichnung_Tank</spalte>
         <spalte>Aggregatzustand_Tank</spalte>
         <spalte>WGK_Tank</spalte>
```

```
         <spalte>Volumen_Tank</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>MNb</wert>
            <wert>flüssig</wert>
            <wert>3</wert>
            <wert/>
            <wert>MNb</wert>
            <wert>flüssig</wert>
            <wert>3</wert>
            <wert>22</wert>
         </zeile>
      </daten>
   </tabelle_angaben_stoffe>
   <tabelle_angaben_gefaaehrdungsstufe>
      <ueberschrift>Ermittlung der Gefährdungsstufe der Anlage nach §39
AwSV</ueberschrift>
      <spalten>
         <spalte>Maßgebendes Volumen</spalte>
         <spalte>Maßgebende WGK</spalte>
         <spalte>Gefährdungsstufe</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>55</wert>
            <wert>3</wert>
            <wert>D</wert>
         </zeile>
      </daten>
   </tabelle_angaben_gefaaehrdungsstufe>
   <tabelle_angaben_behaelter>
      <ueberschrift>Angaben zu den Behältern</ueberschrift>
      <spalten>
         <spalte>Tanknummer</spalte>
         <spalte>enthaltener Stoff</spalte>
         <spalte>einwandig / doppelwandig</spalte>
         <spalte>Nennvolumen (m3)</spalte>
         <spalte>Material</spalte>
         <spalte>Zulassung</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>T4750</wert>
            <wert>MNb</wert>
            <wert>einwandig</wert>
            <wert>22</wert>
            <wert>1.4306</wert>
```

```
            <wert>AD2000</wert>
         </zeile>
      </daten>
   </tabelle_angaben_behaelter>
   <tabelle_angaben_rohrleitungen1>
      <ueberschrift>Angaben zu den doppelwandigen Rohrleitungen</ueberschrift>
      <spalten>
         <spalte>Bauart</spalte>
         <spalte>oberirdisch / unterirdisch</spalte>
         <spalte>Anzahl</spalte>
         <spalte>Material</spalte>
         <spalte>Zulassung</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>Doppelwandig mit Leckanzeige</wert>
            <wert/>
            <wert>0</wert>
            <wert/>
            <wert/>
         </zeile>
      </daten>
   </tabelle_angaben_rohrleitungen1>
   <tabelle_angaben_rohrleitungen2>
      <ueberschrift>Angaben zu den einwandigen Rohrleitungen</ueberschrift>
      <spalten>
         <spalte>Bauart</spalte>
         <spalte>oberirdisch / unterirdisch</spalte>
         <spalte>Anzahl</spalte>
         <spalte>Material</spalte>
         <spalte>Zulassung</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>Einwandig</wert>
            <wert>X</wert>
            <wert>10</wert>
            <wert>1.4435, 1.4303</wert>
            <wert>AD2000</wert>
         </zeile>
      </daten>
   </tabelle_angaben_rohrleitungen2>
   <tabelle_angaben_rohrleitungen3>
      <ueberschrift>Angaben zu den einwandigen Rohrleitungen als
Saugleitung</ueberschrift>
      <spalten>
         <spalte>Bauart</spalte>
         <spalte>oberirdisch / unterirdisch</spalte>
```

```
         <spalte>Anzahl</spalte>
         <spalte>Material</spalte>
         <spalte>Zulassung</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>Einwandig als Saugleitung</wert>
            <wert>X</wert>
            <wert>1</wert>
            <wert>1.4303</wert>
            <wert>AD2000</wert>
         </zeile>
      </daten>
   </tabelle_angaben_rohrleitungen3>
   <tabelle_angaben_rohrleitungen4>
      <ueberschrift>Angaben zu den einwandigen Rohrleitungen im
Schutzrohr</ueberschrift>
      <spalten>
         <spalte>Bauart</spalte>
         <spalte>oberirdisch / unterirdisch</spalte>
         <spalte>Anzahl</spalte>
         <spalte>Material</spalte>
         <spalte>Zulassung</spalte>
      </spalten>
      <daten>
         <zeile>
            <wert>Einwandig im Schutzrohr</wert>
            <wert/>
            <wert>0</wert>
            <wert/>
            <wert/>
         </zeile>
      </daten>
   </tabelle_angaben_rohrleitungen4>
</formblatt_6_2>
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[48], line 136
    132 current_directory_path_xml = os.path.join(os.getcwd(), "formular_6_2_auto.
 ↪xml")
    135 # Speichern des XML-Codes in einer .xml-Datei in dem aktuellen Verzeichnis
--> 136 with open(current_directory_path_xml, "w") as xml_file:
    137     xml_file.write(xml_str)
    139 print(f"XML-Formular wurde in '{current_directory_path_xml}' gespeichert.")
```

```
File ~/anaconda3/envs/schemaextraction/lib/python3.12/site-packages/IPython/core/
 ↪interactiveshell.py:310, in _modified_open(file, *args, **kwargs)
    303 if file in {0, 1, 2}:
    304     raise ValueError(
    305         f"IPython won't let you open fd={file} by default "
    306         "as it is likely to crash IPython. If you know what you are doing,␣
 ↪"
    307         "you can use builtins' open."
    308     )
--> 310 return io_open(file, *args, **kwargs)

OSError: [Errno 30] Read-only file system: '/formular_6_2_auto.xml'
```

Formular PDF

[49]:
```
import os
from jinja2 import Template

# LaTeX-Template als String in einer Zelle
latex_template = r"""
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{enumitem}
\usepackage{fancyhdr}
\usepackage{tabularx}
\usepackage{float}
\usepackage{xcolor}

% Seitenränder einstellen
\usepackage[left=2cm, right=2cm, top=2cm, bottom=1cm, headheight=1.5cm,␣
 ↪includehead]{geometry}

% Fancyhdr-Stil definieren
\pagestyle{fancy}
\fancyhf{}  % Kopf- und Fußzeilen leeren

% Befüllung der Kopfzeile mit einer Tabelle und manuellen Breiten und Höhen
\fancyhead[L]{\begin{tabular}{|p{0.75\textwidth}|p{0.25\textwidth}|}
    \hline
    \textbf{\Large{Antragsunterlage}} \newline \Large{für␣
 ↪immissionsschutzrechtliche Genehmigungsverfahren} & Anlage 1 / Formblatt 6.2␣
 ↪\newline Detailangaben wassergefährdende Stoffe \\[1cm]
    \hline
\end{tabular}}

\pagecolor{yellow!10} % 20% Gelb, anpassbar
\color{black}
```

```
\begin{document}

\subsection*{Angaben zur Anlage}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|l|X|}
        \hline
        \textbf{Anlagenart} & {{ anlagenart }} \\
        \hline
        \textbf{Anlagenbezeichnung} & {{ anlagenbezeichnung }} \\
        \hline
        \textbf{Anlagenumfang} & {{ anlagenumfang }} \\
        \hline
    \end{tabularx}
\end{table}

\subsection*{Angaben zu den wassergefährdenden Stoffen}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|X|}
        \hline
        \textbf{Stoffbezeichnung} & \textbf{Aggregatzustand} & \textbf{WGK} &␣
↪\textbf{Volumen ($m^3$)/ \newline Masse (t)} \\
        \hline
        {{ pumpe_stoff_bezeichnung }} & {{ pumpe_stoff_aggregatzustand }} & {{␣
↪pumpe_stoff_wgk }} & {{ pumpe_stoff_volumen_masse }} \\
        \hline
        {{ tank_stoff_bezeichnung }} & {{ tank_stoff_aggregatzustand }} & {{␣
↪tank_stoff_wgk }} & {{ tank_stoff_volumen_masse }} \\
        \hline
    \end{tabularx}
\end{table}

\subsection*{Ermittlung der Gefährdungsstufe der Anlage nach §39 AwSV}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|}
        \hline
        \textbf{Maßgebendes Volumen ($m^3$)} & \textbf{Maßgebende WGK} &␣
↪\textbf{Gefährdungsstufe} \\
        \hline
```

```
        {{ anlage_volumen }} & {{ anlage_wgk }} & {{ anlage_gefaehrdungsstufe }}␣
↪\\
        \hline
    \end{tabularx}
\end{table}

\subsection*{Angaben zu den Behältern}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|X|X|X|}
        \hline
        \textbf{Tanknummer} & \textbf{Stoff} & \textbf{einwandig/ \newline␣
↪doppelwandig} & \textbf{Nennvolumen \newline ($m^3$)} & \textbf{Material} &␣
↪\textbf{Nachweis} \\
        \hline
        {{ tank_nummer }} & {{ tank_stoff }} & {{ tank_wandig }} & {{␣
↪tank_volumen }} & {{ tank_material }} & {{ tank_zulassung }} \\
        \hline
    \end{tabularx}
\end{table}



\subsection*{Angaben zu den Rohrleitungen}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|X|X|X|}
        \hline
        \textbf{Bauart} & \textbf{oberirdisch} & \textbf{unterirdisch} &␣
↪\textbf{Anzahl} & \textbf{Material} & \textbf{Nachweis} \\
        \hline
        \textbf{Doppelwandig \newline mit Leckanzeige} & {{ rl_1_oberirdisch }}␣
↪& {{ rl_1_unterirdisch }} & {{ rl_1_anzahl }} & {{ rl_1_material }} & {{␣
↪rl_1_zulassung }} \\
        \hline
        \textbf{Einwandig \newline} & {{ rl_2_oberirdisch }} & {{␣
↪rl_2_unterirdisch }} & {{ rl_2_anzahl }} & {{ rl_2_material }} & {{␣
↪rl_2_zulassung }} \\
        \hline
        \textbf{Einwandig \newline als Saugleitung} & {{ rl_3_oberirdisch }} &␣
↪{{ rl_3_unterirdisch }}  & {{ rl_3_anzahl }} & {{ rl_3_material }} & {{␣
↪rl_3_zulassung }} \\
        \hline
```

```
        \textbf{Einwandig \newline im Schutzrohr} & {{ rl_4_oberirdisch }} & {{␣
↪rl_4_unterirdisch }}  & {{ rl_4_anzahl }} & {{ rl_4_material }} & {{␣
↪rl_4_zulassung }} \\
        \hline
    \end{tabularx}
\end{table}



\end{document}

"""


# Erstellen einer Jinja2-Vorlage
template = Template(latex_template)

# Variablen setzen (diese sollten zuvor definiert worden sein)
variables = {
    "anlagenart": anlagenart,
    "anlagenbezeichnung": anlagenbezeichnung,
    "anlagenumfang": anlagenumfang,
    "pumpe_stoff_bezeichnung": pumpe["pumpe_stoff"],
    "pumpe_stoff_aggregatzustand": pumpe["pumpe_aggregatzustand"],
    "pumpe_stoff_wgk": pumpe["pumpe_wgk"],
    "pumpe_stoff_volumen_masse": pumpe["pumpe_volumenstrom"],
    "tank_stoff_bezeichnung": tank["tank_stoff"],
    "tank_stoff_aggregatzustand": tank["tank_aggregatzustand"],
    "tank_stoff_wgk": tank["tank_wgk"],
    "tank_stoff_volumen_masse": tank["tank_volumen"],
    "anlage_volumen": anlagen_volumen,
    "anlage_wgk": anlagen_wgk,
    "anlage_gefaehrdungsstufe": anlagen_gefaehrdungsstufe,
    "tank_nummer": tank["tank_nummer"],
    "tank_stoff": tank["tank_stoff"],
    "tank_wandig": tank["tank_wandig"],
    "tank_volumen": tank["tank_volumen"],
    "tank_material": tank["tank_material"],
    "tank_zulassung": tank["tank_zulassung"],
    "rl_1_oberirdisch": rl_oberirdisch[0],
    "rl_1_unterirdisch": rl_unterirdisch[0],
    "rl_1_anzahl": rl_1_anzahl,
    "rl_1_material": ', '.join(list(set(map(str, rl_material[0])))),
    "rl_1_zulassung": ', '.join(list(set(map(str, rl_zulassung[0])))),
    "rl_2_oberirdisch": rl_oberirdisch[1],
    "rl_2_unterirdisch": rl_unterirdisch[1],
    "rl_2_anzahl": rl_2_anzahl,
```

```
    "rl_2_material": ', '.join(list(set(map(str, rl_material[1])))),
    "rl_2_zulassung": ', '.join(list(set(map(str, rl_zulassung[1])))),
    "rl_3_oberirdisch": rl_oberirdisch[2],
    "rl_3_unterirdisch": rl_unterirdisch[2],
    "rl_3_anzahl": rl_3_anzahl,
    "rl_3_material": ', '.join(list(set(map(str, rl_material[2])))),
    "rl_3_zulassung": ', '.join(list(set(map(str, rl_zulassung[2])))),
    "rl_4_oberirdisch": rl_oberirdisch[3],
    "rl_4_unterirdisch": rl_unterirdisch[3],
    "rl_4_anzahl": rl_4_anzahl,
    "rl_4_material": ', '.join(list(set(map(str, rl_material[3])))),
    "rl_4_zulassung": ', '.join(list(set(map(str, rl_zulassung[3])))),
}

# Rendern der Vorlage mit den Variablen
latex_code = template.render(variables)

# Printen des LaTeX-Codes
print(latex_code)

# Pfad auf das aktuelles Verzeichnis
current_directory_path_tex = os.path.join(os.getcwd(), "formular_6_2_auto.tex")

# Speichern des LaTeX-Codes in einer .tex-Datei in dem aktuellen Verzeichnis
with open(current_directory_path_tex, "w") as tex_file:
    tex_file.write(latex_code)

print(f"LaTeX-Formular wurde in '{current_directory_path_tex}' gespeichert.")
```

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{enumitem}
\usepackage{fancyhdr}
\usepackage{tabularx}
\usepackage{float}
\usepackage{xcolor}

% Seitenränder einstellen
\usepackage[left=2cm, right=2cm, top=2cm, bottom=1cm, headheight=1.5cm,
includehead]{geometry}

% Fancyhdr-Stil definieren
\pagestyle{fancy}
\fancyhf{}  % Kopf- und Fußzeilen leeren

% Befüllung der Kopfzeile mit einer Tabelle und manuellen Breiten und Höhen
\fancyhead[L]{\begin{tabular}{|p{0.75\textwidth}|p{0.25\textwidth}|}
```

```
    \hline
    \textbf{\Large{Antragsunterlage}} \newline \Large{für
immissionsschutzrechtliche Genehmigungsverfahren} & Anlage 1 / Formblatt 6.2
\newline Detailangaben wassergefährdende Stoffe \\[1cm]
    \hline
\end{tabular}}

\pagecolor{yellow!10} % 20% Gelb, anpassbar
\color{black}


\begin{document}

\subsection*{Angaben zur Anlage}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|l|X|}
        \hline
        \textbf{Anlagenart} & HBV-Anlage \\
        \hline
        \textbf{Anlagenbezeichnung} & CO1VO1-HEX.EXO3 \\
        \hline
        \textbf{Anlagenumfang} & Kolbenpumpe, Rohrbündelwärmetauscher,
Kreiselpumpe, Rohrleitungssystem, Behälter, Plattenwärmetauscher \\
        \hline
    \end{tabularx}
\end{table}

\subsection*{Angaben zu den wassergefährdenden Stoffen}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|X|}
        \hline
        \textbf{Stoffbezeichnung} & \textbf{Aggregatzustand} & \textbf{WGK} &
\textbf{Volumen ($m^3$)/ \newline Masse (t)} \\
        \hline
        MNb & flüssig & 3 & 33 \\
        \hline
        MNb & flüssig & 3 & 22 \\
        \hline
    \end{tabularx}
\end{table}

\subsection*{Ermittlung der Gefährdungsstufe der Anlage nach §39 AwSV}

\begin{table}[H]
```

```
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|}
        \hline
        \textbf{Maßgebendes Volumen ($m^3$)} & \textbf{Maßgebende WGK} &
\textbf{Gefährdungsstufe} \\
        \hline
        55 & 3 & D \\
        \hline
    \end{tabularx}
\end{table}

\subsection*{Angaben zu den Behältern}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|X|X|X|}
        \hline
        \textbf{Tanknummer} & \textbf{Stoff} & \textbf{einwandig/ \newline
doppelwandig} & \textbf{Nennvolumen \newline ($m^3$)} & \textbf{Material} &
\textbf{Nachweis} \\
        \hline
        T4750 & MNb & einwandig & 22 & 1.4306 & AD2000 \\
        \hline
    \end{tabularx}
\end{table}



\subsection*{Angaben zu den Rohrleitungen}

\begin{table}[H]
    \centering
    \begin{tabularx}{\textwidth}{|X|X|X|X|X|X|}
        \hline
        \textbf{Bauart} & \textbf{oberirdisch} & \textbf{unterirdisch} &
\textbf{Anzahl} & \textbf{Material} & \textbf{Nachweis} \\
        \hline
        \textbf{Doppelwandig \newline mit Leckanzeige} &  &  & 0 &  &  \\
        \hline
        \textbf{Einwandig \newline} & X &  & 10 & 1.4435, 1.4303 & AD2000 \\
        \hline
        \textbf{Einwandig \newline als Saugleitung} & X &  & 1 & 1.4303 &
AD2000 \\
        \hline
        \textbf{Einwandig \newline im Schutzrohr} &  &  & 0 &  &  \\
        \hline
    \end{tabularx}
\end{table}
```

```
\end{document}
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[49], line 175
    172 current_directory_path_tex = os.path.join(os.getcwd(), "formular_6_2_auto.
 ↪tex")
    174 # Speichern des LaTeX-Codes in einer .tex-Datei in dem aktuellen␣
 ↪Verzeichnis
--> 175 with open(current_directory_path_tex, "w") as tex_file:
    176     tex_file.write(latex_code)
    178 print(f"LaTeX-Formular wurde in '{current_directory_path_tex}' gespeicher.
 ↪")

File ~/anaconda3/envs/schemaextraction/lib/python3.12/site-packages/IPython/core/
 ↪interactiveshell.py:310, in _modified_open(file, *args, **kwargs)
    303 if file in {0, 1, 2}:
    304     raise ValueError(
    305         f"IPython won't let you open fd={file} by default "
    306         "as it is likely to crash IPython. If you know what you are doing,␣
 ↪"
    307         "you can use builtins' open."
    308     )
--> 310 return io_open(file, *args, **kwargs)

OSError: [Errno 30] Read-only file system: '/formular_6_2_auto.tex'
```

Arbeitsverzeichnis prüfen

[ ]:
```
# Aktuelles Arbeitsverzeichnis anzeigen
current_directory = os.getcwd()
print("Aktuelles Arbeitsverzeichnis:", current_directory)
```

```
Aktuelles Arbeitsverzeichnis: /
```