[2-2. Excel 파일 읽고 쓰기]

1. openpyxl 라이브러리 소개

'openpyxl' 라이브러리는 파이썬에서 엑셀 파일을 다루는 데 사용되는 라이브러리입니다. 이를 사용하면 엑셀 파일을 열고, 수정하고, 저장할 수 있습니다.

1-1. 라이브러리 설치

먼저 'openpyxl' 라이브러리가 설치되어 있지 않다면, 이를 설치해야 합니다. 아나콘다를 설치하고, 아나콘다 환경의 파이썬을 사용하는 경우 해당 라이브러리가 이미 설치되어 있을 가능성이 높습니다.

1-1-1. 설치 확인

- 아나콘다 네비게이터 - Environments 에서 'openpyxl'을 조회해보면 설치 여부를 확인할 수 있습니다.
- 직접 import를 해보는 방식으로 설치 여부를 확인할 수도 있습니다. import 해보고 에러가 발생하면 설치가 필요합니다.

In [ ]:
```
import openpyxl
openpyxl
```

Out[ ]:
```
<module 'openpyxl' from '/Users/KP_Hong/anaconda3/lib/python3.11/site-packages/openpyxl/__init__.py'>
```

- 상기 실행 내용을 보시면 import를 하는 중에 에러가 발생하지 않고, 'openpyxl' 객체를 조회했을 때 해당 모듈의 내용이 조회되고 있습니다.
- 설치가 되어 있고, 사용이 가능한 상황입니다.

1-1-2. 설치 하기

- 가장 쉬운 방식은 아나콘다에서 라이브러리를 조회하고 설치하는 것입니다.
- 윈도우즈 명령 프롬프트 또는 파워쉘에서나 맥OS 터미널에서 아래와 같이 명령어를 실행하여 설치할 수도 있습니다.
  `pip install openpyxl`

1-2. Excel 파일 열고 쓰기

1-2-1. 엑셀 파일에 데이터 쓰기

In [ ]:
```python
from openpyxl import Workbook

# 새로운 워크북(엑셀 파일) 생성
workbook = Workbook()
```

- 'openpyxl' 라이브러리의 Workbook 클래스를 import 합니다.
- Workbook() 클래스를 사용하여 새로운 워크북(엑셀 파일)을 생성합니다.

In [ ]:
```python
# 활성 시트(첫 번째 시트) 선택
sheet = workbook.active

# 셀에 데이터 쓰기
sheet['A1'] = 'Name'
sheet['B1'] = 'Age'
sheet['C1'] = 'Gender'

# 여러 셀에 데이터 쓰기
data = [
    ('John', 25, 'male'),
    ('Emily', 30, 'female'),
    ('Michael', 35, 'male')
]
for row_data in data:
    sheet.append(row_data)
```

- 'workbook' 인스턴스의 'active' 속성을 사용하여 활성 시트(첫번째 시트)를 선택한 후 'sheet' 변수에 할당합니다.
- 셀의 주소를 인덱스로 사용하여 각 셀에 데이터를 씁니다.
- 'append()' 메서드를 사용하여 여러 셀에 한 번에 데이터를 쓸 수도 있습니다.

In [ ]:
```python
# 엑셀 파일 저장
workbook.save('excel_data.xlsx')
```

- 'workbook' 인스턴스의 'save()' 메서드를 사용하여 엑셀 파일을 저장합니다. 이때 저장할 파일의 이름을 인자로 전달합니다.

1-2-2. 엑셀 파일의 데이터 읽어오기

In [ ]:
```python
from openpyxl import load_workbook

# 엑셀 파일 열기
workbook = load_workbook('excel_data.xlsx')

# 활성 시트(첫 번째 시트) 선택
sheet = workbook.active

# 엑셀 파일의 각 행을 출력
for row in sheet.iter_rows(values_only=True):
    print(row)
```

```
('Name', 'Age', 'Gender')
('John', 25, 'male')
('Emily', 30, 'female')
('Michael', 35, 'male')
```

In [ ]:
```python
# 공 딕셔너리 생성
data_dict = {}

for col in sheet.iter_cols(values_only=True):
    # 각 컬럼 값들에 대해서 첫번째 값은 딕셔너리의 키로, 나머지 값들은 딕셔너리의 값으로 사용
    data_dict[col[0]] = list(col[1:])

print(data_dict)
```

```
{'Name': ['John', 'Emily', 'Michael'], 'Age': [25, 30, 35], 'Gender': ['male', 'female', 'male']}
```

In [ ]:
```python
data_dict['Age']
```

Out[ ]:
```
[25, 30, 35]
```

1-2-3. sheet 객체의 cell 객체에 접근하기

- sheet 객체에 셀의 주소를 지정해주는 방식으로 개별 셀 객체에 접근 가능합니다.

In [ ]:
```python
# sheet 객체의 cell 객체
sheet['A1']
```

Out[ ]:
```
<Cell 'Sheet'.A1>
```

In [ ]:
```python
sheet['A1'].value
```

Out[ ]:
```
'Name'
```

In [ ]:
```python
sheet['A2'].value
```

Out[ ]:
```
'John'
```

- sheet 객체는 cell 메서드를 가지고 있으며, cell 메서드에 행번호와 열번호를 지정해주는 방식으로 셀에 접근 가능합니다.

In [ ]:
```
sheet.cell(row=1, column=1)
```

Out[ ]:
```
<Cell 'Sheet'.A1>
```

In [ ]:
```
sheet.cell(row=1, column=1).value
```

Out[ ]:
```
'Name'
```

In [ ]:
```
sheet.cell(2,3).value
```

Out[ ]:
```
'male'
```

cell 객체는 value 속성 외에도 다양한 속성을 가지고 있습니다.

- value 속성 : 셀의 값
- row 속성 : 셀의 행 인덱스(1부터 시작)
- column 속성 : 셀의 열 인덱스(A부터 시작)
- coordinate : 셀의 좌표(예: A1, B2 등)
- font : 셀의 글꼴 설정 정보
- fill : 셀의 배경색 설정 정보
- number_format : 셀의 숫자 형식

1-2-4. sheet 객체의 셀 범위에 접근하기

- sheet 객체에 셀의 주소 범위를 지정해주는 방식으로 특정 범위의 셀에 접근 가능합니다.

In [ ]:
```
# sheet 객체의 CellRange 객체
sheet['A1:C1']
```

Out[ ]:
```
((<Cell 'Sheet'.A1>, <Cell 'Sheet'.B1>, <Cell 'Sheet'.C1>),)
```

In [ ]:
```
sheet['A2:C4']
```

Out[ ]:
```
((<Cell 'Sheet'.A2>, <Cell 'Sheet'.B2>, <Cell 'Sheet'.C2>),
 (<Cell 'Sheet'.A3>, <Cell 'Sheet'.B3>, <Cell 'Sheet'.C3>),
 (<Cell 'Sheet'.A4>, <Cell 'Sheet'.B4>, <Cell 'Sheet'.C4>))
```

In [ ]:
```
data_list = []
for row in sheet['A2:C4']:
    row_list = []
    for val in row:
        row_list.append(val.value)
    data_list.append(row_list)

print(data_list)
```

```
[['John', 25, 'male'], ['Emily', 30, 'female'], ['Michael', 35, 'male']]
```

In [ ]:
```
values = [[cell.value for cell in row] for row in sheet['A2:C4']]
print(values)
```

```
[['John', 25, 'male'], ['Emily', 30, 'female'], ['Michael', 35, 'male']]
```

1-2-5 데이터 추가 입력하기

- sheet 객체의 append 메서드를 사용하면 행 전체를 입력할 수 있습니다.

In [ ]:
```
sheet.append(['David', 27, 'male'])
workbook.save('excel_data.xlsx')
```

- cell 객체의 value 속성을 이용하여 특정 행과 열에 직접 데이터를 입력하는 것도 가능합니다.

In [ ]:
```
row_data = ['Celine', 24, 'female']
for col, data in enumerate(row_data, start=1):
    sheet.cell(row=5, column=col).value = data
workbook.save('excel_data.xlsx')
```

1-3. Worksheet 객체(sheet)의 주요 속성 및 메서드

1-3-1. 속성

- title : 시트의 이름을 가져오거나 설정합니다.
- max_row : 시트의 최대 행 번호를 반환합니다.
- max_column : 시트의 최대 열 번호를 반환합니다.
- dimensions : 시트의 데이터가 있는 범위를 문자열로 반환합니다.(예: 'A1:C10')
- rows : 시트의 모든 행을 순회할 수 있는 객체를 반환합니다. 각 행은 셀 객체의 튜플입니다.
- columns : 시트의 모든 열을 순회할 수 있는 객체를 반환합니다. 각 열은 셀 객체의 튜플입니다.

1-3-2. 메서드

- cell(row, column, value=None) : 주어진 행과 열에 해당하는 셀 객체를 반환하거나, 값을 설정합니다.
- append(iterable) : 리스트, 튜플, 또는 딕셔너리 형태의 데이터를 받아 시트의 마지막 행 다음에 새로운 행으로 추가합니다.
- iter_rows(min_row=None, max_row=None, min_col=None, max_col=None, values_only=False) : 지정된 범위의 행을 순회할 수 있는 객체를 반환합니다.
- iter_cols(min_row=None, max_row=None, min_col=None, max_col=None, values_only=False) : 지정된 범위의 열을 순회할 수 있는 객체를 반환합니다.
- delete_rows(start=None, end=None) : 지정된 범위의 행을 삭제합니다.
- delete_cols(start=None, end=None) : 지정된 범위의 열을 삭제합니다.

1-4. Workbook 객체의 주요 속성 및 메서드

1-4-1. 속성

- active : 현재 활성화된 시트를 가져오거나 설정합니다. 일반적으로 마지막으로 열린 시트가 활성 시트가 됩니다.
- sheetnames : 워크북에 있는 모든 시트의 이름을 리스트로 반환합니다.

1-4-2. 메서드

- create_sheet(title=None, index=None) : 새로운 시트를 생성하고 추가합니다. 'title'로 시트 이름을, 'index'로 추가 위치를 지정할 수 있습니다. 생성된 시트 객체를 반환합니다.
- remove(sheet) : 주어진 시트를 워크북에서 제거합니다. 'sheet'는 시트 객체 또는 시트 이름일 수 있습니다.
- save(filename) : 워크북을 주어진 파일 이름으로 저장합니다. 파일 확장자에 따라 저장 포맷이 결정됩니다. (일반적으로 '.xlsx')

1-4-3. Worksheet 지정하기

- Workbook['Sheet_Name'] : 시트 이름으로 Worksheet 객체를 지정합니다.

In [ ]:
```
sheet = workbook['Sheet']
```

In [ ]:
```
sheet
```

Out[ ]:
```
<Worksheet "Sheet">
```

In [ ]:
```
sheet.dimensions
```

Out[ ]:
```
'A1:C5'
```

In [ ]:
```

```