1. 리스트 컴프리헨션

- [표현식 for 항목 in 반복가능객체 if 조건문]

In [ ]:
```
# 0부터 9까지의 숫자 중에 대해서 제곱을 구하여 새로운 리스트 생성
squares = [x**2 for x in range(10)]
print(squares)  # 출력: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [ ]:
```
# 0부터 9까지의 숫자 중에서 짝수의 제곱을 구하여 새로운 리스트 생성
squares = [x**2 for x in range(10) if x % 2 == 0]
print(squares)  # 출력: [0, 4, 16, 36, 64]
```

```
[0, 4, 16, 36, 64]
```

In [ ]:
```
# 두 리스트의 모든 조합을 포함하는 새로운 리스트 생성
pairs = [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
print(pairs)  # 출력: [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3,
```

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

2. csv 라이브러리로 csv 파일 읽고, 쓰기

In [ ]:
```
#### csv 파일 쓰기 ####

import csv

with open('csv_data.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    # 데이터 쓰기
    writer.writerow(['Name', 'Age', 'Grade'])
    writer.writerow(['John', '25', 'A'])
    writer.writerow(['Emily', '22', 'B'])
```

In [ ]:
```
#### csv 파일 읽기 ####

import csv

with open('csv_data.csv', 'r') as file:
    reader = csv.reader(file)
    # 각 행에 대해 반복하며 데이터 읽기
    for row in reader:
        print(row)
```

```
['Name', 'Age', 'Grade']
['John', '25', 'A']
['Emily', '22', 'B']
```

3. openpyxl 라이브러리로 excel 파일 읽고, 쓰기

In [ ]:
```python
#### 엑셀 파일에 데이터 쓰기 ####

from openpyxl import Workbook

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

# 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)

# 엑셀 파일 저장
workbook.save('excel_data.xlsx')
```

In [ ]:
```python
#### 엑셀 파일 데이터 읽어오기 ####

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']}
```

4. Pandas 주요 메서드

4-1. merge 메서드

- 두 DataFrame을 병합하는 기능을 제공하며, SQL의 JOIN 연산과 유사하게 작동

In [ ]:
```python
pandas.merge(
    left, right,
    how='inner', on=None, left_on=None, right_on=None,
    left_index=False, right_index=False, sort=True
)
```

4-2. join 메서드

- 한 DataFrame을 다른 DataFrame의 인덱스에 따라 병합할 때 사용
- 인덱스를 기준으로 merge 작업을 할 때 편리함

In [ ]:
```python
DataFrame.join(other, on=None, how='left', sort=False)
```

4-3. concat 메서드

- pandas에서 여러 객체를 축(axis)을 따라 연결

In [ ]:
```python
pandas.concat(objs, axis=0, join='outer', ignore_index=False)
```

4-4. pivot_table

- 복잡한 데이터를 집계하고, 데이터를 재구성하는데 사용
- 데이터를 특정 키에 따라 그룹화하고, 각 그룹에 대해 다양한 집계 함수를 적용할 수 있음

In [ ]:
```python
import numpy as np
import pandas as pd
from pandas import DataFrame, Series

df = pd.DataFrame({
    'A': ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
    'B': ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
    'C': np.random.randn(8),
    'D': np.random.randn(8)
})

# pivot_table 생성
pivot_table = pd.pivot_table(
    df,
    values='D',
    index=['A'],
    columns=['B'],
    aggfunc='sum'
)

print(pivot_table)
```

```
B         one     three       two
A
bar  0.688553  1.306816  0.069684
foo -1.050675  1.294156  0.628948
```

4-5. groupby

- SQL의 'GROUP BY' 명령어와 유사한 방식으로 작동하며, 하나 이상의 컬럼을 기준으로 데이터를 그룹화하고, 각 그룹에 대해 집계 함수를 적용함
- groupby의 작동 원리
  - 분할(Split): 데이터를 특정 기준에 따라 여러 그룹으로 분할
  - 적용(Apply): 각 그룹에 대해 집계, 변환, 필터링 등의 연산을 적용
  - 결합(Combine): 연산의 결과를 하나의 데이터 구조로 결합

In [ ]:
```python
df.groupby(
    by=None, axis=0, level=None, as_index=True, sort=True,
    group_keys=True, dropna=True
)
```

[주요 매개변수]

- by : 그룹화할 기준을 설정합니다. 컬럼 이름이나, 컬럼을 선택하는 함수, 컬럼 이름의 리스트 등을 사용할 수 있습니다.
- axis : 0은 행을 기준으로 그룹화하고, 1은 열을 기준으로 그룹화합니다.
- level : 멀티인덱스인 경우, 인덱스의 라벨을 기준으로 그룹화합니다.
- as_index : True일 경우, 그룹 라벨을 인덱스로 사용합니다. False일 경우, 그룹 라벨이 인덱스로 사용되지 않고, 데이터 컬럼 중 하나로 남습니다.
- sort : 그룹 키에 따라 정렬할지 여부를 결정합니다.
- group_keys : 기본값은 True입니다. 이를 False로 설정하면, groupby에 의해 반환된 객체에서 그룹 키가 인덱스에 추가되지 않습니다. 그룹화 작업 후, 결과 DataFrame에서 그룹화를 위해 사용된 컬럼이 멀티인덱스의 일부로 포함되지 않게 합니다.
- dropna : 기본값은 True입니다. False로 설정하면, 그룹화하는 과정에서 NA(null) 값을 하나의 그룹으로 간주합니다. 이는 데이터 분석 시 NA 값을 따로 분류하고 싶을 때 유용할 수 있습니다.

4-5. pandas display 설정

In [ ]:
```
import pandas as pd
pd.set_option('display.max_rows', 30)
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_colwidth', 20)
pd.set_option('display.width', 300)

# DataFrame의 출력을 확장하여 한 줄로 계속 출력되도록 설정
pd.set_option('display.expand_frame_repr', True)
```

5. 데이터베이스 Connector

In [ ]:
```python
import mysql.connector
import pandas as pd

class Connector:
    def __enter__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            port="3306",
            password="password",
            database="database_name"
        )
        self.cursor = self.connection.cursor(dictionary=True)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cursor.close()
        self.connection.close()

    def insert_data(self, query, data):
        self.cursor.executemany(query, data)
        self.connection.commit()

    def fetch_data(self, query):
        self.cursor.execute(query)
        result = self.cursor.fetchall()
        return pd.DataFrame(result)
```

In [ ]:
```python

```