Design Patterns for Handling Data 2

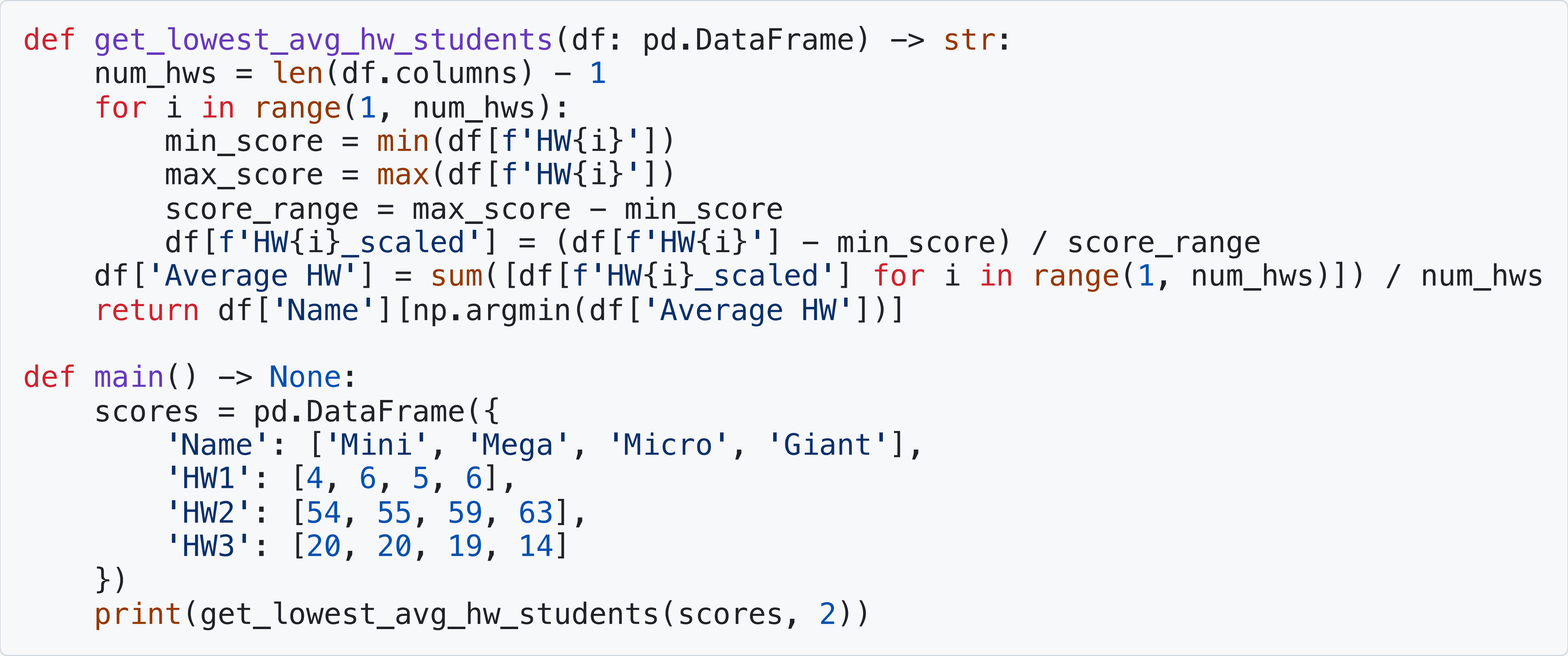
Welcome back to CS 2100!

Prof. Rasika Bhalerao

Refresher: Scaling and Argmax / Argmin

Different homework assignments are out of different numbers of points. Let's write a function that takes a dataframe of grades and returns the name of the student with the lowest average homework score.

Assume the assignments are named `HW1`, `HW2`, ...



```python
def get_lowest_avg_hw_students(df: pd.DataFrame) -> str:
    num_hws = len(df.columns) - 1
    for i in range(1, num_hws):
        min_score = min(df[f'HW{i}'])
        max_score = max(df[f'HW{i}'])
        score_range = max_score - min_score
        df[f'HW{i}_scaled'] = (df[f'HW{i}'] - min_score) / score_range
    df['Average HW'] = sum([df[f'HW{i}_scaled'] for i in range(1, num_hws)]) / num_hws
    return df['Name'][np.argmin(df['Average HW'])]

def main() -> None:
    scores = pd.DataFrame({
        'Name': ['Mini', 'Mega', 'Micro', 'Giant'],
        'HW1': [4, 6, 5, 6],
        'HW2': [54, 55, 59, 63],
        'HW3': [20, 20, 19, 14]
    })
    print(get_lowest_avg_hw_students(scores, 2))
```

Motivating example

Grades come from two dataframes:

Pawtograder:

Student ID	HW1 score
001	40
002	39

PollEv:

Student ID	Number of lectures attended
001	10
002	11

How can we combine them into a single gradebook?

Combining dataframes

We previously discussed concatenating dataframes to add one to the "end" of the other.

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat'],
    'Age': [13, 10]})

new_rows = pd.DataFrame({
    'Person': ['Dog', 'Giraffe'],
    'Age': [3, 6]})

df = pd.concat([df, new_rows], ignore_index=True)
```

```
     Person  Age
0  Elephant   13
1       Cat   10
2       Dog    3
3   Giraffe    6
```

But that wouldn't work for our case (combining grade tables)

```python
import pandas as pd

df_hw = pd.DataFrame({
    'Student ID': ['001', '002'],
    'HW1 score': [40, 39]
})

df_polls = pd.DataFrame({
    'Student ID': ['001', '002'],
    'N lec att': [10, 11]
})

df = pd.concat(
    [df_hw, df_polls],
    ignore_index=True)
print(df)
```

```
  Student ID  HW1 score  N lec att
0        001       40.0        NaN
1        002       39.0        NaN
2        001        NaN       10.0
3        002        NaN       11.0
```

Merge: combining dataframes side-by-side instead of at the end

Default: `pandas.merge()` combines dataframes using any columns with the same name:

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Dog', 'Giraffe'],
    'Age': [13, 10, 3, 6]})

other_df = pd.DataFrame(
    {'Person': ['Dog', 'Giraffe', 'Elephant', 'Cat'],
     'BFF': ['Cat', 'Elephant', 'Giraffe', 'Cat']}
)

print(pd.merge(df, other_df))
```

```
     Person  Age       BFF
0  Elephant   13   Giraffe
1       Cat   10       Cat
2       Dog    3       Cat
3   Giraffe    6  Elephant
```

`merge()` works for our grade dataframe

```
df = pd.merge(df_hw, df_polls)
```

```
  Student ID  HW1 score  Number of lectures attended
0        001         40                           10
1        002         39                           11
```

Next complication: What if both tables have Student ID and Name?

Pandas can't assume that we do something silly like this:

Pawtograder:

Student ID	Name	HW1 score
001	Mini	40
002	Micro	39

PollEv:

Student ID	Name	Number lectures attended
001	Giant	10
002	Micro	11

Which identifier will it use for merging?

Merge: combining dataframes side-by-side

Multiple columns with the same name -> specify which one to use with the `on` arg

If the columns with the same name don't "agree" across the two dataframes (e.g., in one dataset, the cat's age is 10, and in the other, the cat's age is 6), it will include both of the disagreeing columns, suffixed by the original dataframe (`x` or `y`):

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Dog', 'Giraffe'],
    'Age': [13, 10, 3, 6]})

other_df = pd.DataFrame(
    {'Person': ['Dog', 'Giraffe', 'Elephant', 'Cat'],
     'Age': [13, 10, 3, 6],  # these ages are different from the ones in df
     'BFF': ['Cat', 'Elephant', 'Giraffe', 'Cat']}
)

print(pd.merge(df, other_df, on='Person'))
```

Merge: combining dataframes side-by-side

If the columns with the same name don't "agree" across the two dataframes (e.g., in one dataset, the cat's age is 10, and in the other, the cat's age is 6), it will include both of the disagreeing columns, suffixed by the original dataframe (`x` or `y`):

```
     Person  Age_x  Age_y       BFF
0  Elephant     13      3   Giraffe
1       Cat     10      6       Cat
2       Dog      3     13       Cat
3   Giraffe      6     10  Elephant
```

Merge

Values that only appear in one dataset and not the other -> it will omit those rows entirely:

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Dog', 'Giraffe'],
    'Age': [13, 10, 3, 6]})

df_age_year = pd.DataFrame({
    'Age': [3, 6, 9, 10],
    'Year': [2022, 2019, 2016, 2015]
})

print(pd.merge(df, df_age_year))
```

```
    Person  Age  Year
0      Cat   10  2015
1      Dog    3  2022
2  Giraffe    6  2019
```

Merge

Don't like that strategy for values that only appear in one dataset? Use the `how` arg:

- `'left'` -> resulting dataframe will include all rows from the left dataframe, filling in `NaN` ("Not a Number") values for pieces missing on the right

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Dog', 'Giraffe'], 'Age': [13, 10, 3, 6]})

df_age_year = pd.DataFrame({
    'Age': [3, 6, 9, 10], 'Year': [2022, 2019, 2016, 2015]})

print(pd.merge(df, df_age_year, on='left'))
```

```
     Person  Age    Year
0  Elephant   13     NaN
1       Cat   10  2015.0
2       Dog    3  2022.0
3   Giraffe    6  2019.0
```

Merge

- `'left'` -> resulting dataframe will include all rows from the left dataframe, filling in `NaN` ("Not a Number") values for pieces missing on the right
- `'right'` -> result will include all rows from the right dataframe, filling in `NaN` for the pieces missing on the left

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Dog', 'Giraffe'], 'Age': [13, 10, 3, 6]})
df_age_year = pd.DataFrame({
    'Age': [3, 6, 9, 10], 'Year': [2022, 2019, 2016, 2015]})

print(pd.merge(df, df_age_year, on='right'))
```

```
    Person  Age  Year
0      Dog    3  2022
1  Giraffe    6  2019
2      NaN    9  2016
3      Cat   10  2015
```

Options for `merge()`'s `how` arg

- `'left'` -> resulting dataframe will include all rows from the left dataframe, filling in `NaN` ("Not a Number") values for pieces missing on the right
- `'right'` -> result will include all rows from the right dataframe, filling in `NaN` for the pieces missing on the left
- `'outer'` -> include all rows from both
- `'inner'` -> only include rows that were in both

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Dog', 'Giraffe'],
    'Age': [13, 10, 3, 6]})

df_age_year = pd.DataFrame({
    'Age': [3, 6, 9, 10],
    'Year': [2022, 2019, 2016, 2015]
})

print(pd.merge(df, df_age_year, on='outer'))
```

```
     Person  Age    Year
0  Elephant   13     NaN
1       Cat   10  2015.0
2       Dog    3  2022.0
3   Giraffe    6  2019.0
4       NaN    9  2016.0
```

Poll: What is output?

```python
df_all_students = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Cat', 'Dog', 'Elephant', 'Giraffe']
})

df_swimming_class_grades = pd.DataFrame({
    'ID': [1, 3, 5],
    'Grade': ['A', 'B', 'C']
})

print(pd.merge(df_all_students, df_swimming_class_grades, on='ID', how='left'))
```

1. A dataframe with 3 rows
2. A dataframe with 4 rows
3. A dataframe with 5 rows
4. Error -- cannot merge dataframes with different lengths

Exercise

Assume we have two dataframes:

Prices:

Item	Cost
Blue Shirt	10
Black Shirt	12
White Shirt	12
Pink Shirt	10

Mini's purchases:

Item
White Shirt
White Shirt
Pink Shirt

How can we find the total cost of all of Mini's purchases?

MapReduce

MapReduce is a framework for processing data using the `map` and `reduce` concepts, independent of programming language:

1. **Map phase**: data is broken into pieces, and each piece is "mapped" or transformed
2. **Reduce phase**: mapped data is "reduced" or combined into a result

It's a popular way to process large datasets because it can be split across multiple computers (the separate chunks don't rely on each other). The results will be reduced, or combined, together at the end.

MapReduce Example

We are counting the frequency of each word in a piece of text.

1. The map phase will involve splitting the text into words, and mapping each word to a key-value pair such as `(word, 1)`, where `word` is the word being mapped.
2. The reduce phase will involve grouping the key-value pairs by word, and then reducing, or adding up, the numbers in each group.

Let's walk through the Python code for this example (`map_reduce.py`)

Poll:

1. What is your main takeaway from today?

2. What would you like to revisit next time?