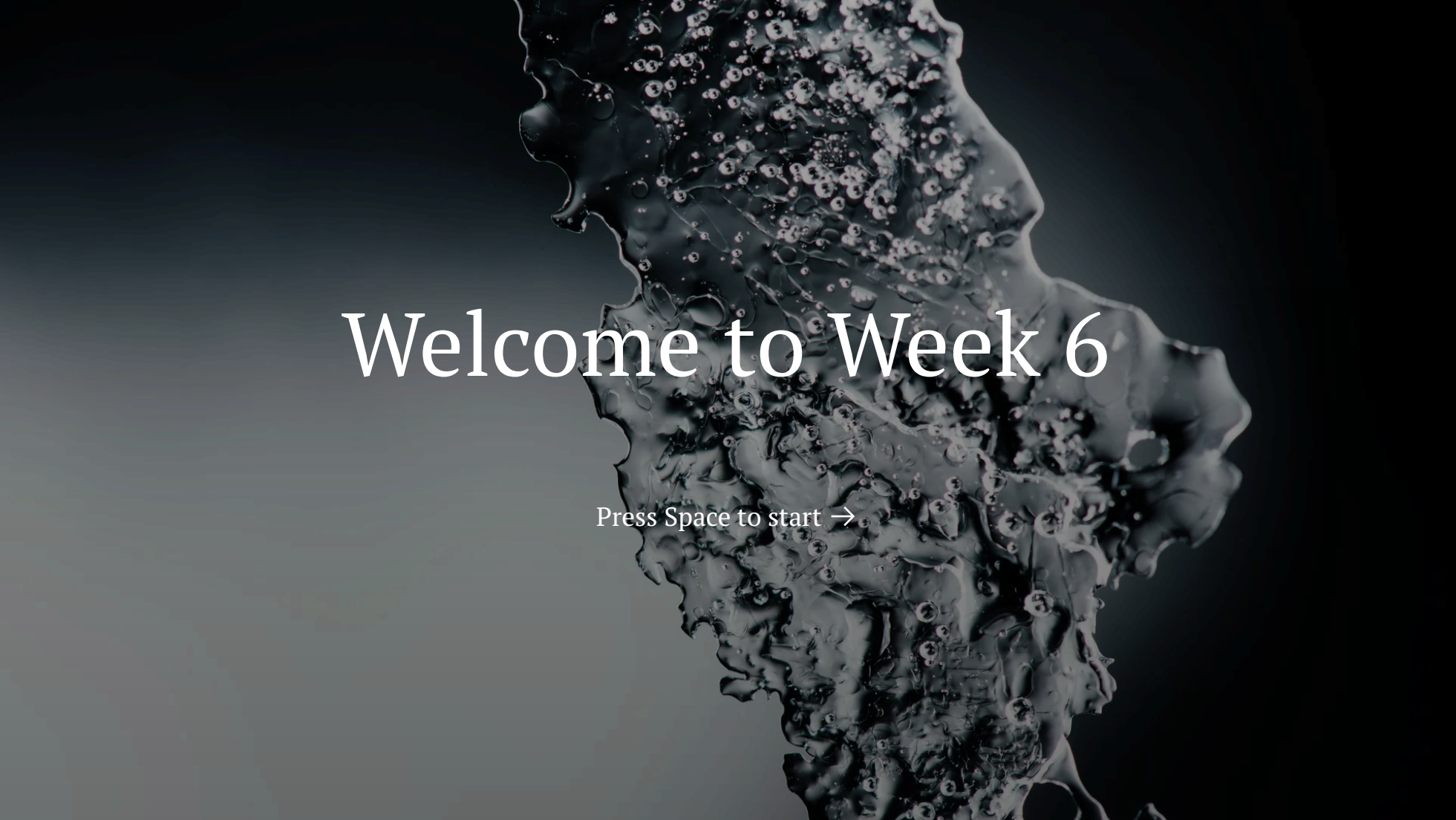
Welcome to Week 6
Press Space to start →

Thursday Announcements, etc.

A few things to discuss before jumping back into slides

- Announcement about quiz and midterm
- Clarifying exact end of "midterm content"
- Upcoming Academic Enhancement Program
  - well-being and "learning about learning
- Exam schedule

Table of contents

Ok, back to content

- We want to extend our recommender systems example

```
with open("example.csv", "r") as file:
    # Skip header
    header = file.readline()

    # Process and print each record
    for line in file:
        columns = line.strip().split(",")
        nice_output = ""
        for column in columns:
          nice_output += column + " | "
        print(nice_output)
```

Generating data (advanced)

```
import csv
import random

# Lists of names and diets
names = ["Michael Jordan", "Tom Hanks", "Arya Stark", "Goku", "Scarlett Johansson",
         "Mickey Mouse", "Hermione Granger", "Naruto", "Serena Williams", "Oprah Winfrey",
         "Leonardo DiCaprio", "Mario", "Taylor Swift", "Darth Vader", "James Bond",
         "Usain Bolt", "Tony Stark", "Bugs Bunny", "The Rock", "Walter White",
         "Sheldon Cooper", "Mulan", "Elsa", "Sherlock Holmes", "Jack Sparrow",
         "Luffy", "Luke Skywalker", "Marilyn Monroe", "Freddie Mercury", "Lady Gaga",
         "Superman", "Wonder Woman", "Neo", "Beyonce", "Ash Ketchum",
         "Lara Croft", "Elvis Presley", "Kermit the Frog", "Spider-Man", "Hannibal Lecter",
         "Indiana Jones", "Simba", "Rihanna", "Dexter Morgan", "Morpheus",
         "Wolverine", "Cinderella", "Captain America", "Thor", "Katniss Everdeen"]
```

```
movies_by_genre = {
    "Drama": ["The Shawshank Redemption", "Forrest Gump", "The Godfather", "Titanic", "Fight Club"],
    "Action": ["The Dark Knight", "Avengers: Endgame", "Pulp Fiction", "Jurassic Park", "Lord of the Rings: The Fellowsh
    "Sci-Fi": ["Inception", "The Matrix", "Back to the Future", "Star Wars: Episode IV", "Blade Runner"],
    "Animation": ["The Lion King", "Toy Story", "Frozen", "Finding Nemo", "Shrek"],
    "Classic": ["Casablanca", "Jaws", "Gone with the Wind", "Psycho", "Citizen Kane"]
}
```

```
parties = ["very left wing", "left wing", "right wing", "very right wing"]

diets = ["plant-heavy", "meat-heavy", "fruit-heavy", "balanced", "dairy-heavy"]

#Generate 50 records
records = [["name", "favorite_movie", "second_favorite_movie", "preferred_political_party", "ideal_diet"]]
for _ in range(50):
    name = random.choice(names)
    genre = random.choice(list(movies_by_genre.keys()))
    movie = random.choice(movies_by_genre[genre])
    movie2 = random.choice(movies_by_genre[genre])
    party = random.choice(parties)
    diet = random.choice(diets)
    records.append([name, movie, movie2, party, diet])

# Write to a CSV file
with open("fake_data.csv", "w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows(records)

# Display the first few rows
for row in records[:3]:
    print(", ".join(row))
```

(Switch over to VS Code and show what this looks like as a single file)

Check that you can read it

```
with open("fake_data.csv", "r") as file:
    header = file.readline()
    print(header)
    line = file.readline()
    print(line)
```

Similarity

Our initial definition: "common interests counter"

- If we have the same favorite movie, that’s one common interest
- If we have the same preferred political party, that’s one common interest.
- If we also have the same preferred diet, that’s one common interest.
- Let’s see who is the most similar!

Many definitions of similar

In CS and math, there are many ways we can see how "similar" or "close" two items are

- Cosine similarity, manhattan distance, etc.
- So we need to specify a definition and algorithm

Our movie/political party/diet recommender system

To recommend a movie to someone, we take the following approach:

- We’ll look at favorite movie, preferred political party, and ideal diet
- Each match counts as one similarity score. So score between any two people ranges from 0 to 3.
- When we find the "highest" score, suggest our user watch that person’s second favorite movie!

Our algorithm

Initalize

- Set variables to hold all the things we need (i.e., "get our buckets" ready)

Loop

- Run through every record and process it

Initialize, more specifically…

- Pick the user we'll give a recommendation to
- Define a variable to hold similarity score (starts at zero)
- Define a variable to keep track of which person is most similar
- Define a variable to keep track of what what we'll recommend

Loop, more specifically…

For each person other than our chosen one

- Compare the two people
- Check if favorite movie is the same. if so, add one similarity point
- Check if preferred political party is the same. if so, add one similarity point
- Check if ideal diet is the same. if so, add one similarity point
- If similarity score is higher than the current top score, update top score/recommendation

Note: we'll have to use our list/string indexing carefully!

Our pseudo code

```
# select a user (who will receive recommendation)
# init variables to hold top similarity score, top person, recommendation

# load data
    # use a loop to go through all other record lines (one record = one user)
    # make sure to skip chosen user
    # split each record line into a list of items
        # use a nested loop to go through each item
        # make sure we compare columns correctly
    # check if similarity score is higher
        # update if so

# print top score, record, and recommendation
```

Initialize

```
# select a user + init variables to keep track of scores
index_of_user = 17
top_score = 0
top_record = ""
recommendation = ""

# load data
    # use a loop to go through all other record lines (one record = one user)
    # make sure to skip chosen user
    # split each record line into a list of items
        # use a nested loop to go through each item
        # make sure we compare columns correctly
    # check if similarity score is higher
        # update if so

# print top score, record, and recommendation
```

Load data

```
# select a user + init variables to keep track of scores
index_of_user = 17
top_score = 0
top_record = ""
recommendation = ""
with open("fake_data.csv", "r") as file:
    header = file.readline()
    print(header)
    # use a loop to go through all other record lines (one record = one user)
    # make sure to skip chosen user
    # split each record line into a list of items
        # use a nested loop to go through each item
        # make sure we compare columns correctly
    # check if similarity score is higher
        # update if so

# print top score, record, and recommendation
```

First for loop

```
index_of_user = 17
top_score = 0
top_record = ""
recommendation = ""
with open("fake_data.csv", "r") as file:
    all_lines = file.readlines()
    header = all_lines[0]
    records = all_lines[1:]
    user_record = records[index_of_user]
    # use a loop to go through all other record lines (one record = one user)
    for record in records:
        if record == user_record:
            continue
        # split each record line into a list of items
        columns = record.strip().split(",")
        print(columns)
        # use a nested loop to go through each item
```

Note on printing as we go

As we write this code, at each step let’s make our code print something out so we know we’re making progress!

When you’re feeling stuck, trying printing something for every new line of code you write.

Nested loop

```
# select a user + init variables to keep track of scores
index_of_user = 17
top_score = 0
top_record = ""
recommendation = ""

# load data
with open("fake_data.csv", "r") as file:
    all_lines = file.readlines()
    header = all_lines[0]
    records = all_lines[1:]

    user_record = records[index_of_user].strip().split(",")

    # use a loop to go through all other record lines (one record = one user)
```

```
    for record in records:
        columns = record.strip().split(",")

        if user_record[0] == columns[0]:
            print("Skipping ", record)
            continue

        score_for_current_record = 0
        # split each record line into a list of items

        # use a nested loop to go through each item
        # use manually defined [1,3,4,5] to get only
        # favorite_movie, party, diet (skip)
        for column_index in [1,3,4]:
            if columns[column_index] == user_record[column_index]:
                score_for_current_record += 1

        if score_for_current_record > top_score:
            top_score = score_for_current_record
            top_record = record
            recommendation = columns[2]
```

```
print("You are", user_record)

print("After careful consideration, we have found that the most similar user is")
print(top_record)
print(f"You have a similarity score of {top_score}")

print("We recommend you watch", recommendation)
```

(Switch over to VS Code and show what this looks like as a single file)