CMPT 120 LECTURE 5-1

Prof. Nick Vincent

Simon Fraser University

2023-10-04

QUICK NOTE FOR THOSE READING SLIDES

There’s only one set of slides for week 5 (Monday was a holiday).

5-1 AGENDA

- Quick set of review questions
- Quick review on accumulator pattern
- Completing our recommender system example
  - Putting together for loops, nested for loops, list slicing, string slicing

REVIEW QUESTIONS

QUESTION 1A

What does this snippet output?

```
response = "I LOVE RAIN!!"
words = response.lower().strip("!").split(" ")
if "rain" in words or "umbrella" in words:
    print("You're a fan of the rain?")
```

QUESTION 1A

You're a fan of the rain?

QUESTION 1B

```
response = "I LOVE RAIN!!"
try:
    words = response.lower().split(" ").strip("!")
    if "rain" in words or "umbrella" in words:
        print("You're a fan of the rain?")
except AttributeError:
    print("Something went wrong")
```

QUESTION 1B

```
Something went wrong
```

QUESTION 2A

What does this output?

```
1 fruits = "durian, rambutan, lychee"
2 fruit_list = fruits.split(",")
3 print("durian" in fruit_list)
4 print("rambutan" in fruit_list)
```

QUESTION 2A

True
False

QUESTION 2B

What does this output?

```
1 fruits = "durian,rambutan,lychee".split(",")
2 print(fruits[1][0].upper())
```

QUESTION 2B

R

QUESTION 3

Find three errors with this code:

```
1  z = int(input["Give me a number, any number: "])
2  if z > 5 and <= 10:
3      print(x)
```

QUESTION 3 – ANSWER

```
1  z = int(input("Give me a number, any number: "))
2  if z > 5 and z <= 10:
3      print(z)
```

QUESTION 4

What does this snippet output?

```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f']
2 print(len(letters))
3
4 print(letters[1:3])
5 print(letters[:4])
6 print(letters[3:])
7 print(letters[:])
8 print(letters[-1])
9 print(letters[3:-1])
```

QUESTION 4

```
6
['b', 'c']
['a', 'b', 'c', 'd']
['d', 'e', 'f']
['a', 'b', 'c', 'd', 'e', 'f']
f
['d', 'e']
```

TRICK TO REMEMBER SLICING

`letters[:4]` will give you 4 total items in your output! It must start at 0… hence you get 0,1,2, and 3.

ACCESSING CHARACTERS IN STRINGS

- Just like accessing items in a list
- If you think of strings as a list of characters, you're good to go
- (As long as you know how lists work)

ACCUMULATOR REVIEW

"Go through all the items in a pile and pick out the red ones"

```
items = ["red", "blue", "red", "green"]
items_we_grab = []
for item in items:
    if item == "red":
        items_we_grab += [item]
print(items_we_grab)
```

```
['red', 'red']
```

WEEK 5 LEARNING OBJECTIVES

OPEN + READ LINES FROM A FILE

```
1 with open('fake_data.csv', 'r') as file:
2     lines = file.readlines()
3     print(len(lines))
```

```
51
```

SPLIT A STRING INTO A LIST

```
1 text = "Python is fun"
2 words = text.split()  # ['Python', 'is', 'fun']
3 print(words)
```

```
['Python', 'is', 'fun']
```

ACCESS SPECIFIC ELEMENTS OF A LIST USING INDEXING/SLICING

```
numbers = [1, 2, 3, 4, 5]
first_element = numbers[0]    # 1
sliced_elements = numbers[1:4]  # [2, 3, 4]
```

ACCESS SPECIFIC CHARACTER IN A STRING USING INDEXING/SLICING

```
text = "Hello World"
first_char = text[0]  # 'H'
sliced_chars = text[6:11]  # 'World'
```

PERFORM COMPARISONS BETWEEN NUMBERS, TAKING INTO ACCOUNT ORDER OF OPERATORS (OPERATOR PRECEDENCE)

```
result = 2 + 3 * 4 > 10  # Evaluates to 2 + 12 > 10 => 14 > 10 => True
print(result)
```

```
True
```

PERFORM COMPARISONS (E.G. !=, <,>) WITH STRINGS

```
1 string1 = "apple"
2 string2 = "banana"
3 print(string1 != string2)  # True
4 print(string1 < string2)   # True, because 'a' comes before 'b' alphabetica
```

```
True
True
```

INTERPRET CODE WITH NESTED CONDITIONALS + COMPARISON OPERATORS (E.G. !=, <,>=)

```
x = 5
y = 10
if x != y:
    if x < y:
        print("x is less than y")
    else:
        print("x is greater than y")
else:
    print("x is equal to y")
```

```
x is less than y
```

FIND THE COMMON ELEMENTS BETWEEN 2 LISTS

```
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
common_elements = []
for x in list1:
    if x in list2:
        common_elements.append(x)
print(common_elements)  # [3, 4]
```

```
[3, 4]
```

UNDERSTAND + USE NESTED FOR LOOPS

```
1 for i in range(3):
2     for j in range(2):
3         print(i, j)
```

```
0 0
0 1
1 0
1 1
2 0
2 1
```

APPLY OPERATOR ORDER TO EVALUATE EXPRESSIONS

```
result = 3 + 4 * 2 / (1 - 5)  # Evaluates using operator precedence: multip
print(result)
```

```
1.0
```

CONCATENATE LISTS

```
1 list1 = [1, 2, 3]
2 list2 = [4, 5, 6]
3 concatenated_list = list1 + list2
4 print(concatenated_list)  # [1, 2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

APPLY ACCUMULATION PATTERN FOR STRINGS AND LISTS (PREVIOUSLY WAS NUMBERS)

```
# For strings
words = ["Hello", "World"]
sentence = ""
for word in words:
    sentence += word + " "
print(sentence.strip())  # Hello World

# For lists
matrix = [[1, 2], [3, 4]]
flat_list = []
for sublist in matrix:
    for item in sublist:
        flat_list.append(item)
print(flat_list)  # [1, 2, 3, 4]
```

```
Hello World
[1, 2, 3, 4]
```

CALCULATE THE MAXIMUM AMONG SEVERAL VALUES

```
values = [3, 5, 2, 8, 6]
maximum = values[0]
for value in values:
    if value > maximum:
        maximum = value
print(maximum)  # 8
```

```
8
```

OK, BACK TO CONTENT

- We want to extend our recommender systems example

LET'S RECAP...

```
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 Joh
         "Mickey Mouse", "Hermione Granger", "Naruto", "Serena Williams", "
         "Leonardo DiCaprio", "Mario", "Taylor Swift", "Darth Vader", "Jame
         "Usain Bolt", "Tony Stark", "Bugs Bunny", "The Rock", "Walter Whit
         "Sheldon Cooper", "Mulan", "Elsa", "Sherlock Holmes", "Jack Sparro
         "Luffy", "Luke Skywalker", "Marilyn Monroe", "Freddie Mercury", "L
         "Superman", "Wonder Woman", "Neo", "Beyonce", "Ash Ketchum",
         "Lara Croft", "Elvis Presley", "Kermit the Frog", "Spider-Man", "H
         "Indiana Jones", "Simba", "Rihanna", "Dexter Morgan", "Morpheus",
         "Wolverine", "Cinderella", "Captain America", "Thor", "Katniss Eve

movies_by_genre = {
    "Drama": ["The Shawshank Redemption", "Forrest Gump", "The Godfather",
    "Action": ["The Dark Knight", "Avengers: Endgame", "Pulp Fiction", "Jur
```

```
name, favorite_movie, second_favorite_movie, preferred_political_party,
ideal_diet
Arya Stark, Lord of the Rings: The Fellowship of the Ring, Lord of the Rings:
```

GET YOUR OWN FAKE_DATA.CSV

- Download csv file from Canvas -> GitHub
- Download python file from Canvas -> GitHub and run it
- Bonus question: what will be different?

CHECK THAT YOU CAN READ IT

```
1 with open("fake_data.csv", "r") as file:
2     header = file.readline()
3     print(header)
4     line = file.readline()
5     print(line)
```

```
name,favorite_movie,second_favorite_movie,preferred_political_party,ideal_diet

Arya Stark,Lord of the Rings: The Fellowship of the Ring,Lord of the Rings:
The Fellowship of the Ring,left wing,balanced
```

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 distance, 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

- 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

FOR EACH PERSON OTHER THAN OUR CHOSEN ONE

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 use
    # 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 use
    # 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 = ""

# load data
with open("fake_data.csv", "r") as file:
    # Skip header
    header = file.readline()
    print(header)
    # use a loop to go through all other record lines (one record = one use
    # 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
```

```
name,favorite_movie,second_favorite_movie,preferred_political_party,ideal_diet
```

FIRST FOR 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:
    # Skip header
    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 use
    for record in records:
        if record == user_record:
```

```
['Arya Stark', 'Lord of the Rings: The Fellowship of the Ring', 'Lord of the
Rings: The Fellowship of the Ring', 'left wing', 'balanced']
['Superman', 'Inception', 'The Matrix', 'right wing', 'plant-heavy']
['Spider-Man', 'Pulp Fiction', 'The Dark Knight', 'very left wing',
```

```
'balanced']
['Rihanna', 'Citizen Kane', 'Citizen Kane', 'very right wing', 'balanced']
['Luffy', 'Finding Nemo', 'The Lion King', 'left wing', 'dairy-heavy']
['Ash Ketchum', 'Shrek', 'Shrek', 'very right wing', 'plant-heavy']
['Wonder Woman', 'Avengers: Endgame', 'Avengers: Endgame', 'very left wing',
'fruit-heavy']
['Lara Croft', 'Star Wars: Episode IV', 'The Matrix', 'very right wing',
'plant-heavy']
['Walter White', 'Shrek', 'The Lion King', 'very right wing', 'meat-heavy']
['Walter White', 'Citizen Kane', 'Gone with the Wind', 'very left wing',
'balanced']
```

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

NESTED LOOP

```
Skipping  Wonder Woman,Avengers: Endgame,Avengers: Endgame,very left
wing,fruit-heavy

Skipping  Wonder Woman,Shrek,The Lion King,right wing,fruit-heavy

Skipping  Wonder Woman,The Godfather,Forrest Gump,left wing,plant-heavy

You are ['Wonder Woman', 'The Godfather', 'Forrest Gump', 'left wing', 'plant-
heavy']
After careful consideration, we have found that the most similar user is
Wolverine,Shrek,Shrek,left wing,plant-heavy

You have a similarity score of 2
We recommend you watch Shrek
```