“You may also like”

Slice and Nested Loops

Question 1

Which of the four inputs below prints “Enjoy!”?

```
response = input()
    words = response.lower().strip("!").split(" ")
    if "coffee" in words:
        print("Enjoy!")
```

A. **I love *COFFEE* a lot!**

B. **I LOVE COFFEE!!**

C. **I!love!coffee!a!lot!**

D. **None of the above.**

Question 1b

Which of the four inputs below prints "Enjoy!"?

```
response = input()
words = response.lower().split(" ").strip("!")
if "coffee" in words:
    print("Enjoy!")
```

A. **I love *COFFEE* a lot!**

B. **I LOVE COFFEE!!**

C. **I!love!coffee!a!lot!**

D. **None of the above.**

Question 2

What does the following code output?

```
pets = "cats, dogs, birds"
petlist = pets.split(",")
print("cats" in petlist)
print("dogs" in petlist)
```

A. **True, True**
B. **True, False**
C. **False, True**
D. **False, False**

Question 3

Will this code run?

```
x = int(input("Enter a number: "))
if x > 1 and <= 3:
    print(x)
```

A. **Yes**

B. **No**

Question 4

What does this output?

```
foods = ["cherries", "tomatoes"]
print(foods[1][0].upper())
```

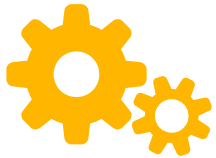
A. O

B. H

C. C

D. T

Sublists using slice



```
letters = ['a', 'b', 'c', 'd', 'e', 'f']
print(letters[1:3])
print(letters[:4])
print(letters[3:])
print(letters[:])
print(letters[-1])
print(letters[3:-1])
```

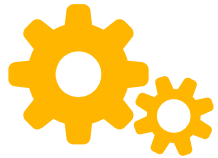
Note: Strings can also be denoted by single quotes. Stylistically, we mainly use ' ' around single characters. For longer strings, double quotes allows us to use single quotes inside it.

Includes element at index 1 up to but not including index 3

Up to but not including element 4

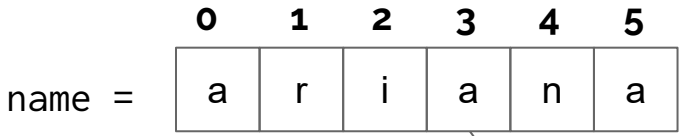
https://runestone.academy/runestone/books/published/thinkcspy/Lists/ListSlices.html

Accessing a character in a string



	Example

Accessing a specific character in a string	`name = "ariana"`
	`first_initial = name[0]`
	`print(first_initial)`



`print(name[3])` → **a**
`print(name[1:3])` → **ri**
`print(name[-2])` → **n**

http://interactivepython.org/runestone/static/CS152f17/Strings/IndexOperatorWorkingwiththeCharactersofaString.html

Substrings using slice

012345

```
mystring = "abcdef"
print(mystring[1:4])
```

Note: doesn't include the character at index 4

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
bcd
```

```
1 fruit = "banana"
2 print(fruit[:3])
3 print(fruit[3:])
4 print(fruit[3:-10])
5 print(fruit[3:99])
6
```

`fruit[3:-10]` prints nothing

`fruit[3:99]` **not** out of range

`fruit[-10]` would be out of range

https://runestone.academy/runestone/books/published/thinkcspy/Strings/IndexOperatorWorkingwiththeCharactersofaString.html

Advanced Recommendations

Recommendation algorithms

Basic What's the **Most** Popular?

One way to recommend something is simply to propose the most popular one, for example: most read news story, most visited cafe, most bought toaster.

Advanced People like you also liked...

Let's say Andrea likes apples, **bananas** and **cherries**. And let's say Bob likes durian, **bananas**, and **cherries**.

Maybe Andrea would like durian (?!) And maybe Bob would like apples.

Similarity

How similar are you to...?

Fork this: https://repl.it/repls/ClearAfraidCheckpoint#main.py

Getting Similarity Scores

```
# Comparing two people's favourite movies
# Author: Angelica Lim
# Date: December 1, 2017

# Description: Finds out how similar two people are by comparing
# their lists of favourite movies

# 1. Get the favourite movies for each person
angelica_favourite_movies = ["Big Hero 6", "Inside Out", "Wall-E"]
baymax_favourite_movies = ["Big Hero 6", "Star Wars", "Wall-E"]

# 2. Initialize a common interests counter
common_interests_counter = 0

# 3. Go through all the favourite movies of the first person
for movie in angelica_favourite_movies:

  # 3a. If that movie is also in the 2nd person's list
  if movie in baymax_favourite_movies:

    # Add to the common interests counter
    common_interests_counter += 1

#4. Print the common interests counter to get a similarity score
print(common_interests_counter)
```

Test and test

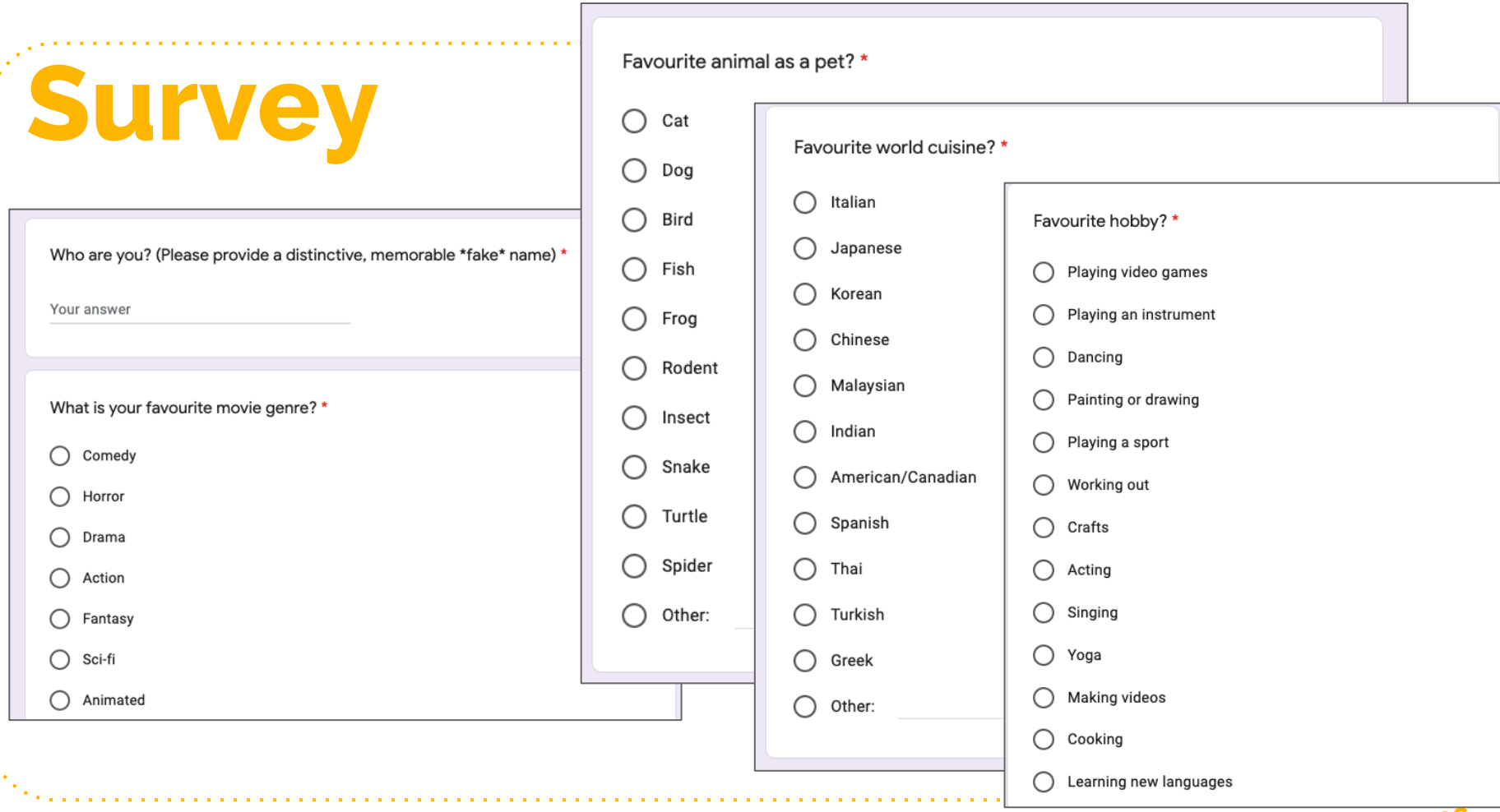
```
# 1. Get the favourite movies for each person
angelica_favourite_movies = ["Big Hero 6", "Inside Out", "Wall-E"]
baymax_favourite_movies = ["Big Hero 6", "Star Wars", "Wall-E"]
```


```
# 1. Get the favourite movies for each person
angelica_favourite_movies = ["Big Hero 6", "Inside Out", "Wall-E"]
baymax_favourite_movies = ["Big Hero 6", "Inside Out", "Wall-E"]
```

Most Similar

Let the computer find the most similar!

Survey


Who are you? (Please provide a distinctive, memorable *fake* name) *
Your answer
What is your favourite movie genre? *
Comedy
Horror
Drama
Action
Fantasy
Sci-fi
Animated
Favourite animal as a pet? *
Cat
Dog
Bird
Fish
Frog
Rodent
Insect
Snake
Turtle
Spider
Other:
Favourite world cuisine? *
Italian
Japanese
Korean
Chinese
Malaysian
Indian
American/Canadian
Spanish
Thai
Turkish
Greek
Other:
Favourite hobby? *
Playing video games
Playing an instrument
Dancing
Painting or drawing
Playing a sport
Working out
Crafts
Acting
Singing
Yoga
Making videos
Cooking
Learning new languages

Survey results

Download the .csv file from Canvas.

Recommendation algorithms

Basic What's the **Most** Popular?

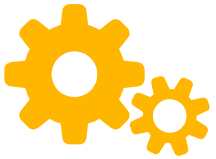
One way to recommend something is simply to propose the most popular one, for example: most read news story, most visited cafe, most bought toaster.

Advanced People like you also liked...

Let's say Andrea likes apples, **bananas** and **cherries**. And let's say Bob likes durian, **bananas**, and **cherries**.

Maybe Andrea would like durian (?!) And maybe Bob would like apples.

Most similar



To make an Advanced Recommendation System, we need to find people who are **most similar**.

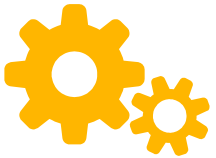
- Last week, we learned how keep track of scores.
- We now know how to calculate a **similarity** score between two people

Now you can find who has **highest similarity score** to you :)

Nested Loops

A loop inside a loop

An algorithm



Initialize:

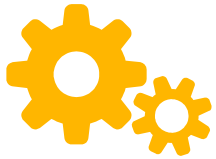
- A **top score** in terms of similarity (e.g., 0 at first)
- The **top name** of the person with that similarity top score (e.g. “” at first)

For each person other than you:

- **Compare** yourself with that one person
- **Calculate** the **similarity score** between you and that person
- If that person has a *higher* similarity score than your current top score, update the **top name** and **top score**

At the end, you'll know who the most similar person is!

A Most Similar algorithm



Let's assume that "my" data is in the first data row. If not, you can cut and paste your data into Row 2. Make sure not to leave an empty line!

favourites-survey.csv

```
1  "Timestamp","Who are you? (Please provide a distinctive,
   memorable *fake* name)","What is your favourite movie genre?",
   "Favourite animal as a pet?","Favourite world cuisine?",
   "Favourite hobby?","What career did you think you'd have as a kid?
   ","What time of the day do you prefer to study?"
2  "2021/02/03 11:06:01 AM PST","Jayrad","Comedy","Dog","Italian",
   "Learning new languages","Teacher","Morning"
3  "2021/02/03 11:21:17 AM PST","Harry Potter","Comedy","Cat",
   "Chinese","Reading","Pilot","Late night"
4  "2021/02/03 11:22:51 AM PST","Kevin Hart","Comedy","Dog",
   "Chinese","Playing a sport","Doctor","Afternoon"
5  "2021/02/03 11:22:55 AM PST","Steve Rogers","Action","Fish",
   "Japanese","Working out","Teacher","Afternoon"
6  "2021/02/03 11:22:55 AM PST","Unkowner","Comedy","Dog","Greek",
   "Playing video games","Detective","Afternoon"
7  "2021/02/03 11:23:03 AM PST","W","Comedy","Cat","Chinese",
   "Cooking","Teacher","Late night"
```

A Most Similar algorithm

```
1   # Similar People Finder
2   # Author: Angelica Lim
3   # Oct. 7, 2020
4
5   # Open the file
6   # Remove/process header
7
8   # Read the first line to get a list of my preferences
9
10  # Initialize variables for top friend and top score
11
12  # Go through each line of the file
13
14      # Get their favourites
15
16      # Get similarity score
17      # Initialize a common interest counter
18      # For each of my favourite things, check if it's also in theirs
19
20      # Check if their score is higher than the current top score
21          # If so, set the top friend name to them
22          # And set the top score to their similarity score
23
24  # Print the top friend
25
```

Assume that "my" data is in the first data row

This is the loop code from earlier today!

In Python

```python
# Find the person in the data who is most similar to you.
# (Put your data on the 2nd line of the .csv file)
import pathlib


# 1. Open the data file
#   Wave have to tell Python where to find the file, and the file's name.
#   a) Get name of folder where this code is saved
folder_of_code = pathlib.Path(__file__).parent.resolve()
#   b) Build the full name of the `favourites.csv` file in that folder
full_file_name = f"{folder_of_code}/favourites.csv"
#   c) Open the file
file = open(full_file_name)
junk_header = file.readline()

# 2. Read the first line and get my list of favourite things
# (Skip the first 2 columns: date, and name)
my_favourites = file.readline().strip().split(",")[2:]

# 3. Initialize variables
top_friend = ""
top_score = 0
top_friends = []

# Go through all remaining people in the file
for line in file:

    # Get their name and list of favourites
    person_data = line.strip().split(",")
    their_name = person_data[1]
    their_favourites = person_data[2:]

    # Calculate similarity score
    common_fav_tally = 0
    for favourite in my_favourites:
        if favourite in their_favourites:
            common_fav_tally += 1

    # Are they more similar than previous people?
    if common_fav_tally > top_score:
        top_score = common_fav_tally
        top_friend = their_name

    # If their score was similar (best or not), add to friends
    if common_fav_tally > 2:
        top_friends.append(their_name)

# Print results
print(f"Top friend: {top_friend} with score {top_score}")
print(f"List of friends: {top_friends}")
```

Remove timestamp and my name

Nested loop

```
"2021/02/03 11:06:01 AM PST","Jayrad","Comedy","Dog","Italian","Learning new languages","Teacher","Morning"
```

```
['"Comedy"', '"Dog"', '"Italian"', '"Learning new languages"', '"Teacher"', '"Morning"']
```

Concatenating lists

```python
# Find the person in the data who is most similar to you.
# (Put your data on the 2nd line of the .csv file)
import pathlib


# 1. Open the data file
#   Wave have to tell Python where to find the file, and the file's name.
#   a) Get name of folder where this code is saved
folder_of_code = pathlib.Path(__file__).parent.resolve()
#   b) Build the full name of the `favourites.csv` file in that folder
full_file_name = f"{folder_of_code}/favourites.csv"
#   c) Open the file
file = open(full_file_name)
junk_header = file.readline()

# 2. Read the first line and get my list of favourite things
# (Skip the first 2 columns: date, and name)
my_favourites = file.readline().strip().split

# 3. Initialize variables
top_friend = ""
top_score = 0
top_friends = []

# Go through all remaining people in the file
for line in file:

    # Get their name and list of favourites
    person_data = line.strip().split(",")
    their_name = person_data[1]
    their_favourites = person_data[2:]

    # Calculate similarity score
    common_fav_tally = 0
    for favourite in my_favourites:
        if favourite in their_favourites:
            common_fav_tally += 1

    # Are they more similar than previous people?
    if common_fav_tally > top_score:
        top_score = common_fav_tally
        top_friend = their_name

    # If their score was similar (best or not), add to friends
    if common_fav_tally > 2:
        top_friends.append(their_name)

# Print results
print(f"Top friend: {top_friend} with score {top_score}")
print(f"List of friends: {top_friends}")
```

Initialize an empty list

Accumulate to make a list of people instead of just choosing 1 person

This will only find the first person in the file with the *highest* score. What small thing could we change to get the last person in the file with the highest score?

Let's review some concepts

What does it mean to have a **nested** loop?

To find the person with a top score in a file, how many variables do you need to initialize? What data types?

How do you extend a list `num_list` to contain another element, let's say `4.0`?

How do you access the second to last element in a list called `favourites`?

How do you access the second and last elements in a list called `favourites`?

Who is the person most similar to you in the class (based on the survey)?