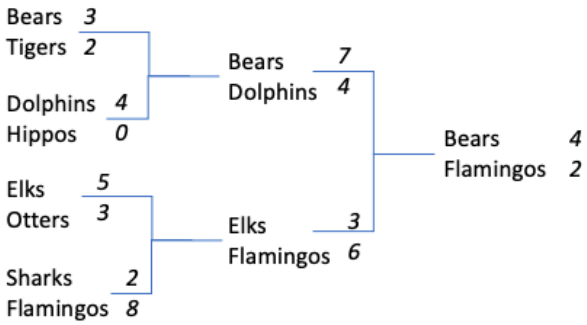
Question (40 Points) – Structuring Data

You need to manage data about a sports tournament that gets played in single-elimination rounds (like World Cup Soccer, Baseball playoffs, Tennis majors, etc). Each team plays in one game in round 1. The winners of each round 1 game advance to round 2, and so on until there is a final winner.

Here is a sample table of data for a tournament involving 8 teams in round 1. The Bears won the tournament in round 3. The diagram on the right provides a different view of the same rounds.

Round	Team 1	Team 2	Score
1	Bears	Tigers	3-2
1	Dolphins	Hippos	4-0
2	Bears	Dolphins	7-4
1	Elks	Otters	5-3
2	Elks	Flamingos	3-6
3	Bears	Flamingos	4-2
1	Sharks	Flamingos	2-8



Your organization wants to answer three questions about tournaments in general (which could have many more teams and rounds than in the above example):

Q1: Which team won the tournament?
Q2: Which teams did the winner play, in order by rounds (from round 1), during the tournament?
Q3: What was the largest difference in score across all games in the tournament? (for the above table, it would be 6, from the Sharks-Flamingos game in round 1)

Keep these questions in mind as you work on the following problems.

1. (10 points) The table has scores written as 3-2, where the first number is for Team 1 and the second is for Team 2. Data of this shape would be captured as a string (like "3-2"). Propose a better way to store the scores in the table for purposes of answering question Q3. Assume you are in Pyret (not plain CSV). Your answer may involve changing or adding columns, adding datatypes, etc. Modify the sample row below to show concretely how the table would appear when you are done, then justify your approach.

Round	Team 1	Team 2	Score
1	Bears	Tigers	3-2

Justification:

2. (15 points) The tournament diagram on the previous page suggests that we could also organize the tournament into a tree. Here's a possible Pyret datatype for tournament trees. (The type of score is left blank since answers to part 1 might differ.)

```
data Tourney:
  | no-match
  | round(
      team1 :: String, team2 :: String, score :: ...,
      feeder1 :: Tourney, feeder2 :: Tourney)
end
```

For each of the questions Q1 through Q3 (repeated below), indicate whether one of the table or tree seems a better data structure for answering the question (you may also say that they are equally good). Justify your choice.

Q1: Which team won the tournament?

Better representation (circle one): Table Tree They're equally good

Why?

Q2: Which teams did the winner play, in order by rounds (from round 1), during the tournament?

Better representation (circle one): Table Tree They're equally good

Why?

Q3: What was the largest difference in score across all games in the tournament?

Better representation (circle one): Table Tree They're equally good

Why?

3. (10 points) Write a expression that will compute the largest difference in score across all games in the tournament, using the table representation. Assume the table uses the score representation you chose in part 1, but does not initially contain the score differences (you may compute those differences as part of your answer).

   Refer to the summary of table operations as needed.

   ```
   # assume the table is named tourn
   ```

4. (5 points) A friend proposes creating a hashtable in which the keys are the round numbers (1, 2, 3, ...) and the value for each key is a dataclass of Match info (containing team names and scores). They suggest turning each row in the table into a key-value entry in the hash table. State whether you would defend or reject this idea, with a sentence or two of justification.

Question (40 Points) – Updating and Testing Data

You want to track data on how many votes each party ("Orange" and "Banana") got in a recent election. Votes are reported by individual districts. You want to track both the vote count per district, and the total vote count across the entire state as results come in. Here are two dataclasses to help with this:

```
@dataclass
class District:
    orange: int  # number of votes for the orange party
    banana: int  # number of votes for the banana party

@dataclass
class StateVotes:
    tot_orange: int  # total votes across all districts for orange party
    tot_banana: int  # total votes across all districts for banana party
    districts: list  # list of District data that are included in totals
```

For example, the following StateVotes data shows the vote count in a state with two districts reporting:

```
StateVotes(150, 350, [District(100, 250),
                      District(50, 100)])
```

Here also is a proposed function to update a StateVote record as new districts report their votes. The function takes a list of District vote tallies and iterates through them to update the state tally (the lines are numbered so we can refer to them later).

```
1  def record_votes(sv: StateVotes, new_districts: list):
2     """takes list of vote counts from districts and adds them
3           to the state vote count tally"""
4     for new_dist in new_districts:
5         new_tot_orange = sv.tot_orange + new_dist.orange
6         new_tot_banana = sv.tot_banana + new_dist.banana
7         sv = StateVotes(new_tot_orange, new_tot_banana,
8                         sv.districts.extend(new_districts))
9         # extend in the previous line appends multiple items
10        # marker for question 3
11
12 # Testing: the test case correctly shows the behavior that we want
13 RIVotes = StateVotes(0, 0, [])
14 # marker for question 2
15 record_votes(RIVotes, [District(100, 250), District(50, 100)])
16 test("check orange votes added", RIVotes.tot_orange, 150)
```

1. (5 points) What is the value of `RIVotes.tot_orange` when execution gets to the test in line 16? Just write your answer, don't explain it (yet).

2. (5 points) Here are the contents of the program dictionary and memory just before `record_votes` is called in the proposed solution (so at line 14). Update this to show the contents of the dictionary and memory after the function has been called (from line 15) but before the for-loop starts executing (at line 4).

Program Dictionary	Memory
`record_votes → function`	`loc 1001 → []`
`RIVotes → loc 1002`	`loc 1002 → StateVotes(0, 0, loc 1001)`

3. (10 points) **Using a different pen/pencil color**, update the dictionary and memory once more (in the area above) to show their contents the first time execution reaches line 10 (after the for loop has run one time). Keep writing on the area above, but **use a different color than you did for question 2**.

4. (11 points) The test case we wrote assumes that changes to `sv` inside the function affect the contents of `RIVotes` outside the function.

    a. Does the for loop change the contents of `RIVotes`? (just answer yes or no)

    b. Did the relationship between `RIVotes` and `sv` in your dictionary/memory diagram change between part 2 and part 3? If so, in what way?

    c. What specific part of lines 7-8 in the original code is responsible for changing or maintaining the relationship you describe in part (b)?

5. (9 points) Assume that the `record_votes` function works as expected (it could have been edited from the original if necessary). Our initial code provided one test case. Describe (in a few words) three additional test cases that you feel are important for testing the function well.

   For example, the test case we gave could be described as "adding multiple districts to an empty state record". Your three descriptions should be similar in precision and length.

   Test 1:

   Test 2:

   Test 3:

For Staff Only

Question	Score	Grader
1		
2		
3		

Question (20 Points) – Organizing Data and Computations

You are working for a city-wide department of education. Every year, they get a table (CSV file) of data on how students at different schools did on state math tests. They keep each year's data in a separate CSV file. Here are a few rows from one year's table (for level, HS=high school, MS=middle school).

school	level	charter	num-students	percent-pass-math
Lakewood	HS	no	650	57
Everest	MS	yes	60	45
Central	MS	no	300	43

The city wants to track two things:

- How city-wide pass rates change over time (they have 10 years of data so far, each in own file)
- How charter schools compare to non-charter schools in math performance

1. (8 points) Finish defining `charters-only16` in the code below (in Pyret), so the city can compute the average percentage passing math across the charter schools in 2016. You may refer to the summary of table operators as needed.

```
# do NOT edit the next line
data2016 = load_table(... "2016-data.csv" ...)

# you may define helpers here (if you wish, they are not required)
```

```
# define this table to contain only the charter schools from data2016
charters-only16 =
```

```
# this line shows how your table is used to compute the average.
# don't edit it
charter-avg16 = mean(charters-only16, "percent-pass-math")
```

2. (12 points) The city needs to create a list of city-wide average pass rates over time. To do this, they have written the following code (since each year of data is in its own file).

```
# 2016 data
data2016 = load_table(... "2016-data.csv" ...)
avg16 = mean(data2016, "percent-pass-math")

# 2015 data
data2015 = load_table(... "2015-data.csv" ...)
avg15 = mean(data2015, "percent-pass-math")

... # and so on through the ten years
all-avgs = [list: avg16, avg15, ..., avg07]
```

You want to help them avoid the duplication of code, making it easier to extend their analysis to include future years as well.

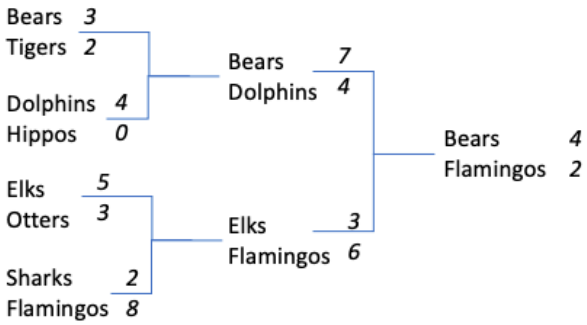
Sketch out an alternative approach that still builds an `all-avgs` list from the collection of csv files, but that doesn't involve writing the same code for multiple years. You don't need to provide full code, but you should show which (a) language constructs and built-in list operations you need (b) how they fit together, and (c) which data/inputs they need. You may summarize or sketch other details with ellipses (...) and comments.

You may write your sketch in either Pyret or Python, as you prefer. Just assume that `mean` and `load-table` work in both languages as they are written above.

Copies of Code and Diagrams

These are copies of tables and code so you don't have to flip pages back and forth while working.

Round	Team 1	Team 2	Score
1	Bears	Tigers	3-2
1	Dolphins	Hippos	4-0
2	Bears	Dolphins	7-4
1	Elks	Otters	5-3
2	Elks	Flamingos	3-6
3	Bears	Flamingos	4-2
1	Sharks	Flamingos	2-8



```
@dataclass
class District:
    orange: int  # number of votes for the orange party
    banana: int  # number of votes for the banana party

@dataclass
class StateVotes:
    tot_orange: int  # total votes across all districts for orange party
    tot_banana: int  # total votes across all districts for banana party
    districts: list  # list of District data that are included in totals
```

```
1  def record_votes(sv: StateVotes, new_districts: list):
2      """takes list of vote counts from districts and adds them
3            to the state vote count tally"""
4      for new_dist in new_districts:
5          new_tot_orange = sv.tot_orange + new_dist.orange
6          new_tot_banana = sv.tot_banana + new_dist.banana
7          sv = StateVotes(new_tot_orange, new_tot_banana,
8                          sv.districts.extend(new_districts))
9          # extend in the previous line appends multiple items
10         # marker for question 3
11
12 # Testing: the test case correctly shows the behavior that we want
13 RIVotes = StateVotes(0, 0, [])
14 # marker for question 2
15 record_votes(RIVotes, [District(100, 250), District(50, 100)])
16 test("check orange votes added", RIVotes.tot_orange, 150)
```

Reference Summary of Useful Operations

If you remember other operations, you are welcome to use them – this is just for reference so you don't have to remember core operations.

Table operations

```
A colname is a string
```

- `filter-by(Table, Row → Boolean) → Table`
- `sort-by(Table, colname, true-if-ascending-order) → Table`
- `transform-column(Table, colname, Row → value) → Table`
- `build-column(Table, colname, Row → value) → Table`
- `table.select-columns(List[colnames]) → Table`
- `table.get-column(colname) → List`
- `row[colname] to access a cell`
- `mean(Table, colname) → Number  # average of nums in a column`
- `sum(Table, colname) → Number  # sum nums in a column`

List operations -- Pyret

- `L.map(elt → value, List[elt]) → List[value]`
- `L.filter(elt → Boolean, List[elt]) → List[elt]`
- `link(elt, List[elt]) → List[elt]`
- `L.append(List, List) → List`
- `L.member(elt, List) → Boolean`
- `length(List) → Number`

List operations -- Python

- `lst.append(elt) → None`
- `lst.extend(list) → None  # like append, but adds multiple elements`
- `lst.contains(elt) → bool`
- `len(list) → int`