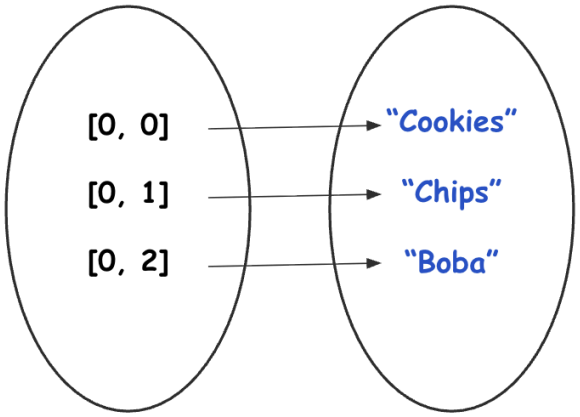
CSCI 134: quiz 2 practice 2

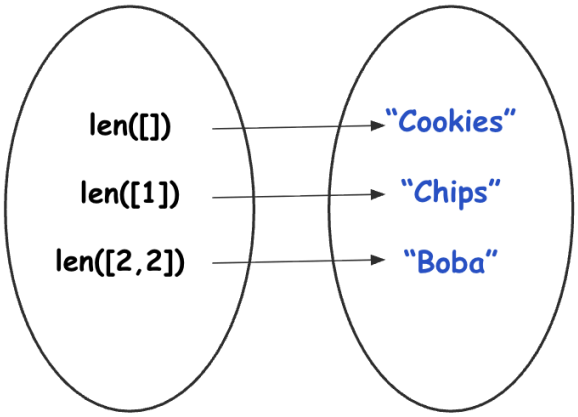
This is a practice quiz for quiz 2. The length and difficulty of the real quiz will be about the same level or easier :)

---

0. reading relations and defining dictionaries



Relation 1



Relation 2

Which of the above relations correspond to valid key-value mappings in a dictionary? This could be neither of the above, one of the above, or both. Leave a comment in the box below with your answer. In addition, for the valid relations, define dictionary variables in the box below that correctly encode the relation.

1. reading 2d lists, complexity, sorting, dictionaries, and mutability (list of topics that could appear here!)

What do the following snippets of code print and what is the computational complexity of the function with respect to the size of the input list or integer

1a

```
def door_dawdle(food_order):
    food_order = food_order[0:len(food_order)-3]
    N = len(food_order)
    for i in range(0, N):
        j = 0
        while j < N-i-1:
            if food_order[j] > food_order[j+1]:
                food_order[j+1] = food_order[j]
                food_order[j] = food_order[j+1]
            j += 1
    print(food_order)

customer_order = ["pizza", "and", "pineapple", "nom", "!", "!", "!"]
door_dawdle(customer_order)
print(customer_order)
```

1b

```
def mystery(img_2d_list):
        # can assume img_2d_list has the same number of rows and columns (NxN)
        N = len(img_2d_list)
        mystery_list = [ ]
        for i in range(0, N):
                j = N-1
                row = [ ]
                while j >= 0:
                        row += [ img_2d_list[i][j] ]
                        j -= 1
                mystery_list += [row]
        return mystery_list

z  =  [ [“this”, “got”, “you”], [“you”, “go”, “yay”], [“woo!”, “woo”, “woo”] ]
print(mystery(z))
```

2. writing 2d lists, classes, sorting, dictionaries, (list of topics that could appear here!)

Write a method in the Movie class below that returns whether the movie is “famous” in the following sense. The method should return

- “classic” if it released before 2015 and has won more than 7 awards
- “instant classic” if it has more than 7 awards but did not release before 2015
- “meh” if neither of the above conditions are true

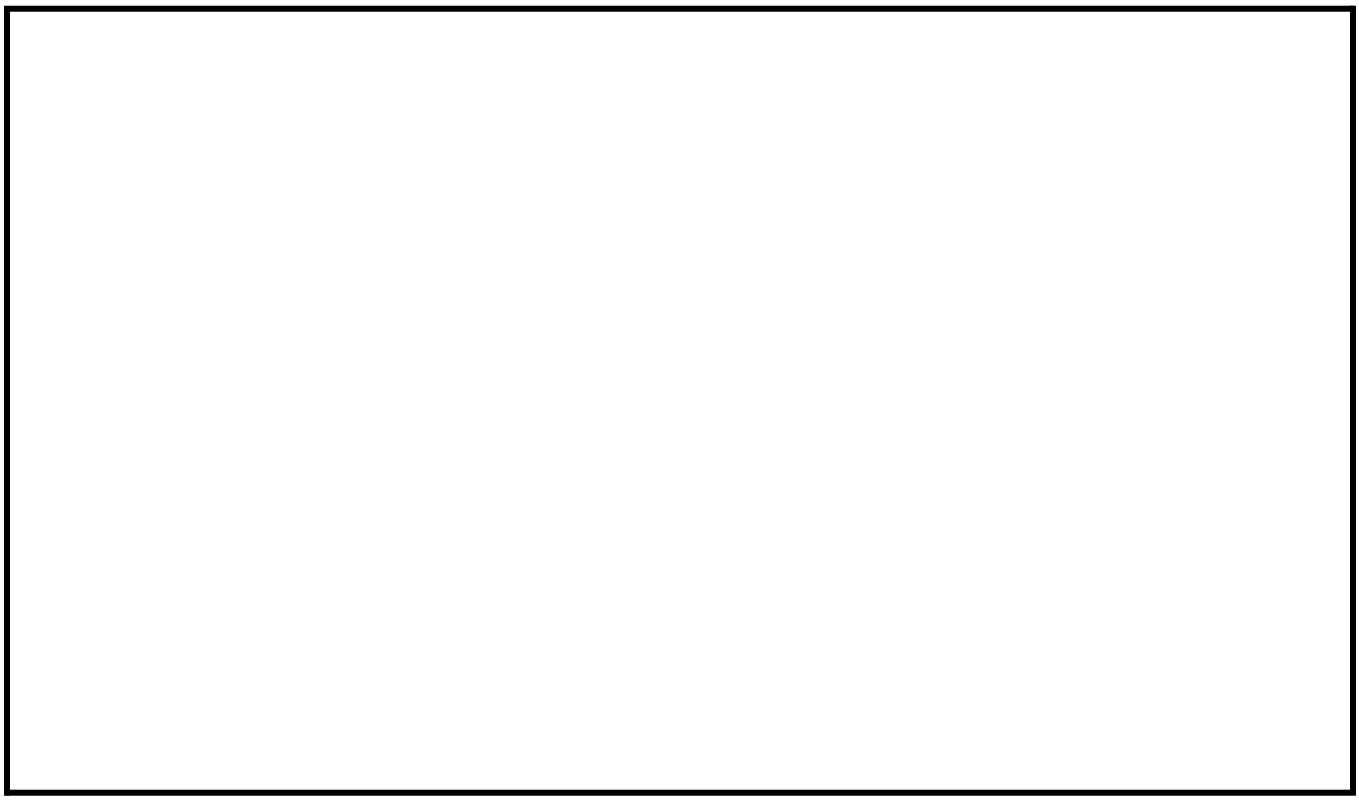
The Movie class is defined below.

```
class Movie:
    def __init__(self, name, year, num_awards):
        self.name = name # string
        self.year = year # int, e.g., 2000 and 2020
        self.num_awards = num_awards # int
    def fame_category(self):
```

3. writing 2d lists, classes, sorting, dictionaries

HBOMin wants to make better recommendations to its customers. Write a function that takes in a list of Movie objects and returns a dictionary mapping from movie names to their fame category as defined in part 2—**the dictionary should only contain movies that are considered a "classic" or an "instant classic"**. An example is shown below.

```
def create_classic_movies_map(movie_object_list):
        # movie_object_list is a list of Movie objects
```



```
movies = [ Movie("Shawshank", 1994, 9), Movie("Princess Bride", 1987, 12),
                Movie("Barbie", 2023, 8), Movie("Wicked", 2024, 3) ]
print(create_classic_movies_map(movies))
# the above should print
{"Shawshank": "classic", "Princess Bride": "classic", "Barbie": "instant classic"}
```