Reminders:

Quiz 02 Friday!

Tutoring in Sitterson (SN) 011 from 5–7PM
today (10/23) and tomorrow (10/24)

Virtual review session tomorrow (10/24) @ 7PM
(Zoom link on site's agenda)

Have a pencil + paper ready for class today!

smart &
capable
coder,

CQ08: Get in ~~loser~~
we're going on a road trip!

We’re going on a road trip!!!
… but first, we need to rent a car.

We want to avoid a young renter fee, so we want someone >= 25 to rent it!

Let’s use code to help us find an eligible renter…

We could use two lists to find someone >= 25:

```
1 def find_car_renter(names: list[str], ages: list[int]) -> str:
2     """Find the name of first person who is at least 25"""
3     if len(names) != len(ages):
4         raise ValueError("The length of names and ages must be the same.")
5     # Loop through ages list
6     for idx in range(0, len(ages)):
7         if ages[idx] >= 25:  # If >= 25, print and return their name!
8             print(names[idx] + " is at least 25!")
9             return names[idx]
10    print("Looks like nobody is old enough to rent a car...")
11
12
13 names = ["Allan", "Ken", "Barbie"]
14 ages = [23, 26, 25]
15 driver = find_car_renter(names, ages)
```

Let’s diagram it! →

```python
def find_car_renter(names: list[str], ages: list[int]) -> str:
    """Find the name of first person >= 25"""
    if len(names) != len(ages):
        raise ValueError("Different lengths :(")
    # Loop through ages list
    for idx in range(0, len(ages)):
        if ages[idx] >= 25:  # If >=255, return name
            print(names[idx] + " is at least 25!")
            return names[idx]
    print("We have to pay the young renter fee...")

names = ["Allan", "Ken", "Barbie"]
ages = [23, 26, 25]
driver = find_car_renter(names, ages)
```

Stack

Heap

Output

Why might writing this function with two lists be suboptimal?

1. Why might writing this function with two lists be suboptimal?

What should we consider if we want to use a dictionary?

It feels unfair to tell the first >= 25-year-old we come across to rent the car…

Let’s rewrite this function to return a dict of all people who are >= 25

```python
def find_car_renter(names: list[str], ages: list[int]) -> str:
    """Find the name of first person who is at least 25"""
    if len(names) != len(ages):
        raise ValueError("The length of names and ages must be the same.")
    # Loop through ages list
    for idx in range(0, len(ages)):
        if ages[idx] >= 25:  # If >= 25, print and return their name!
            print(names[idx] + " is at least 25!")
            return names[idx]
    print("Looks like nobody is old enough to rent a car...")


names = ["Allan", "Ken", "Barbie"]
ages = [23, 26, 25]
driver = find_car_renter(names, ages)
```

Let's rewrite this function to return a dict of all people who are >= 25!

Write it *on paper*. Start with…

```python
def find_eligible(names: list[str], ages: list[int]) -> dict[str, int]:
    """Find the names and ages of all people old enough to rent a car!"""
```

One solution:

```
1 def find_eligible(names: list[str], ages: list[int]) -> dict[str, int]:
2   """Find the names and ages of all people old enough to rent a car!"""
3    if len(names) != len(ages):
4        raise ValueError("Diff lengths.")
5    eligible_ppl: dict[str, int] = {}
6    for idx in range(0, len(names)):
7        if ages[idx] >= 25:# If old enough, add to dict
8            eligible_ppl[names[idx]] = ages[idx]
9    return eligible_ppl
10
11 names = ["Allan", "Ken", "Barbie"]
12 ages = [23, 26, 25]
13 renters: dict[str, int] = find_eligible(names, ages)
14 print(renters)
```

After the fact, we realize we *don't* want Ken to rent a car…

```
 1 def find_eligible(names: list[str], ages: list[int]) -> dict[str, int]:
 2   """Find the names and ages of all people old enough to rent a car!"""
 3    if len(names) != len(ages):
 4        raise ValueError("Diff lengths.")
 5    eligible_ppl: dict[str, int] = {}
 6    for idx in range(0, len(names)):
 7        if ages[idx] >= 25:# If old enough, add to dict
 8            eligible_ppl[names[idx]] = ages[idx]
 9    return eligible_ppl
10
11 names = ["Allan", "Ken", "Barbie"]
12 ages = [23, 26, 25]
13 renters: dict[str, int] = find_eligible(names, ages)
14 print(renters)
```

How could we remove him from **`renters`**?

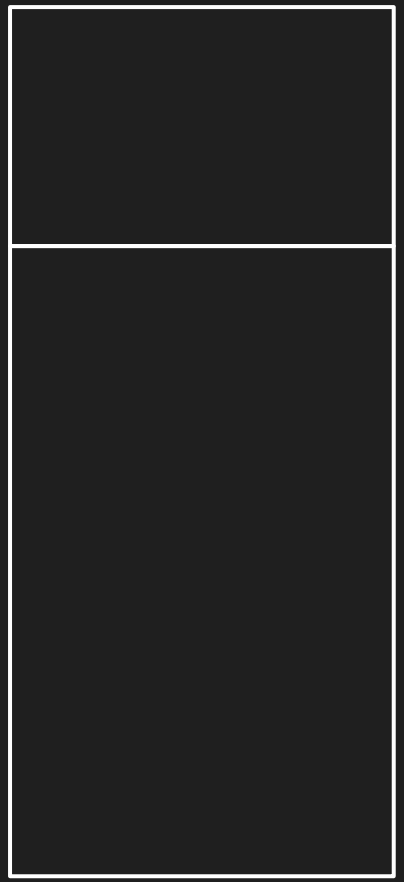
After the fact, we realize we *don't* want Ken to rent a car…

```
1 def find_eligible(names: list[str], ages: list[int]) -> dict[str, int]:
2   """Find the names and ages of all people old enough to rent a car!"""
3    if len(names) != len(ages):
4        raise ValueError("Diff lengths.")
5    eligible_ppl: dict[str, int] = {}
6    for idx in range(0, len(names)):
7        if ages[idx] >= 25:# If old enough, add to dict
8            eligible_ppl[names[idx]] = ages[idx]
9    return eligible_ppl
10
11 names = ["Allan", "Ken", "Barbie"]
12 ages = [23, 26, 25]
13 renters: dict[str, int] = find_eligible(names, ages)
14 print(renters)
15
16 # Let's not let Ken rent a car...
17 if "Ken" in renters:
18    renters.pop("Ken")
19
20 print(renters)
```

```python
def find_eligible(names: list[str], ages: list[int]) -> dict[str, int]:
    """Find the names and ages of all people old enough to rent a car!"""
    if len(names) != len(ages):
        raise ValueError("Diff lengths.")
    eligible_ppl: dict[str, int] = {}
    for idx in range(0, len(names)):
        if ages[idx] >= 25:# If >=25, add to dict
            eligible_ppl[names[idx]] = ages[idx]
    return eligible_ppl

names = ["Allan", "Ken", "Barbie"]
ages = [23, 26, 25]
renters: dict[str, int] = find_eligible(names, ages)
print(renters)

# Let's not let Ken rent a car...
if "Ken" in renters:
    renters.pop("Ken")

print(renters)
```

Output

Stack



Heap

```python
def find_eligible(names: list[str], ages: list[int]) -> dict[str, int]:
  """Find the names and ages of all people old enough to rent a car!"""
    if len(names) != len(ages):
        raise ValueError("Diff lengths.")
    eligible_ppl: dict[str, int] = {}
    for idx in range(0, len(names)):
        if ages[idx] >= 25:# If >=25, add to dict
            eligible_ppl[names[idx]] = ages[idx]
    return eligible_ppl

names = ["Allan", "Ken", "Barbie"]
ages = [23, 26, 25]
renters: dict[str, int] = find_eligible(names, ages)
print(renters)

# Let's not let Ken rent a car...
if "Ken" in renters:
    renters.pop("Ken")

print(renters)
```

Please submit a .pdf of your completed memory diagram to Gradescope!