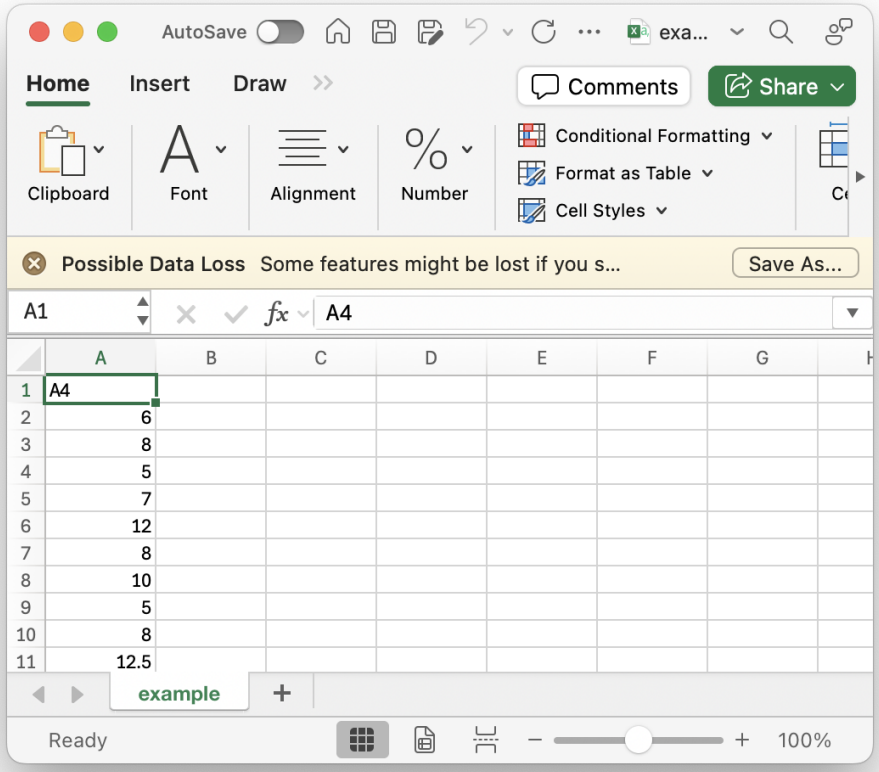
Lecture 11: Dictionaries

CS 50

Spring 2026

Motivating Example: Data Analytics

- Goal: have a dataset and what to compute some statistics about it
  - average (mean)
  - median
  - mode



Example: Data Analytics

- Write a function `mean` that takes a filename (assumed to be a one-column csv file) and returns the mean of the numbers in the first column (excluding the header).

```
def mean(filename):
    sum = 0
    count = 0

    dataset = open(filename, 'r')
    header = dataset.readline() # read header

    for line in dataset: # read remaining lines
        num = float(line)
        sum = sum + num
        count = count + 1

    return sum/count
```

Example: Data Analytics

- Write a function `median` that takes a filename (assumed to be a one-column csv file) and returns the median of the numbers in the first column (excluding the header).

```
def median(filename):
    times = []
    dataset = open(filename, 'r')
    header = dataset.readline() # read header
    for line in dataset: # read remaining lines
        times.append(float(line))

    times.sort()

    if len(times) % 2 == 1: # if odd length
        median_index = len(times)//2
        return times[median_index]
    else: # if even length
        median1 = len(times)//2
        median2 = median1 - 1
        return (times[median1]+times[median2])/2
```

- What about mode?

Dictionaries

- a data structure that associates a key with a value
  - key is a unique identifier (must be immutable)
  - value is something we associate with that key
  - dictionary stores key:value pairs

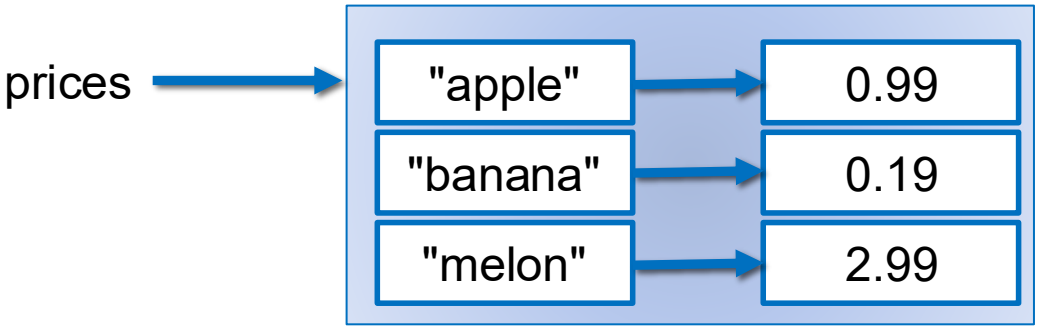
```
d = {'apple': .99, 'banana': .19, 'cantelope': 2.99}
```

- Real world examples
  - dictionary (key: word, value: definition)
  - phonebooks (key: name, value: phone number)
  - price list (key: product, value: price)

Creating Dictionaries

- Dictionaries start/end with curly braces
- Kev:value pairs have colons in between
- Each pair is separated by a comma

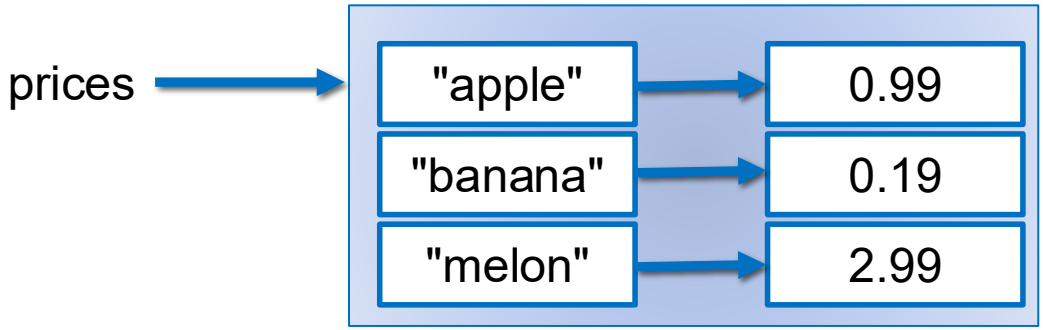
- Examples:
  - empty_dict = {}
  - phone_book = {"Eleanor": "909-607-9963", "Alexandra": "909-607-9963"}
  - prices = {"apple": .99, "banana": .19, "melon": 2.99}



Accessing Dictionary Elements

- Given a dictionary d, use any key k to access the associated value d[k]

- Example:
  - prices = {"apple": .99, "banana": .19, "melon": 2.99}



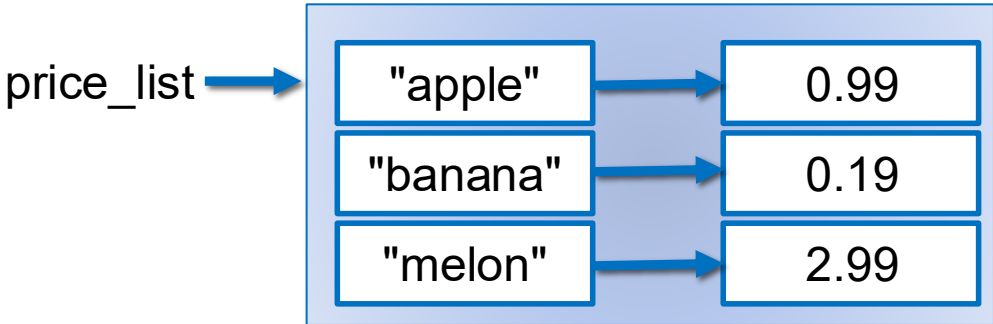
  - prices["apple"] is .99
  - prices["melon"] is 2.99
  - price["grape"] is ??? **KeyError**

Accessing all Dictionary Elements

- Given a dictionary d:
  - d.keys() returns a list of all the keys in d
  - d.values() returns a list of all the values in d
  - d.items() returns a list of all the (key, value) pairs in d

- Example:
  - prices = {"apple": .99, "banana": .19, "melon": 2.99}

```
for key in prices.keys():
    price = prices[key]
    print("The price of a", key, "is", price)
```



Example

- Define a function `least_expensive` that takes one parameter, a dictionary `prices`, and returns the item with the lowest price.

- You may assume that `prices` contains at least one item.

Exercise 1

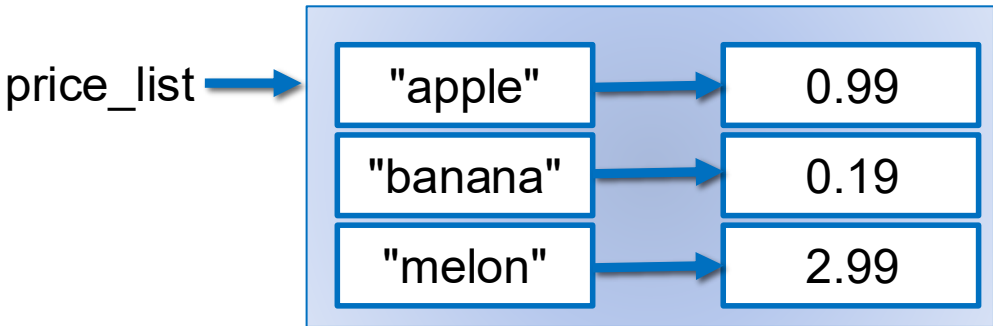
- Define a function `under_price(grocery_store, p)` that takes two parameters, a dictionary `grocery_store` and a price `p`, and returns the list of items available for less than (or equal to) price `p`.

Checking for Dictionary Elements

- Given a dictionary d:
  - d.keys() returns a list of all the keys in d
  - d.values() returns a list of all the values in d
  - d.items() returns a list of all the (key, value) pairs in d

- Example:
  - prices = {"apple": .99, "banana": .19, "melon": 2.99}

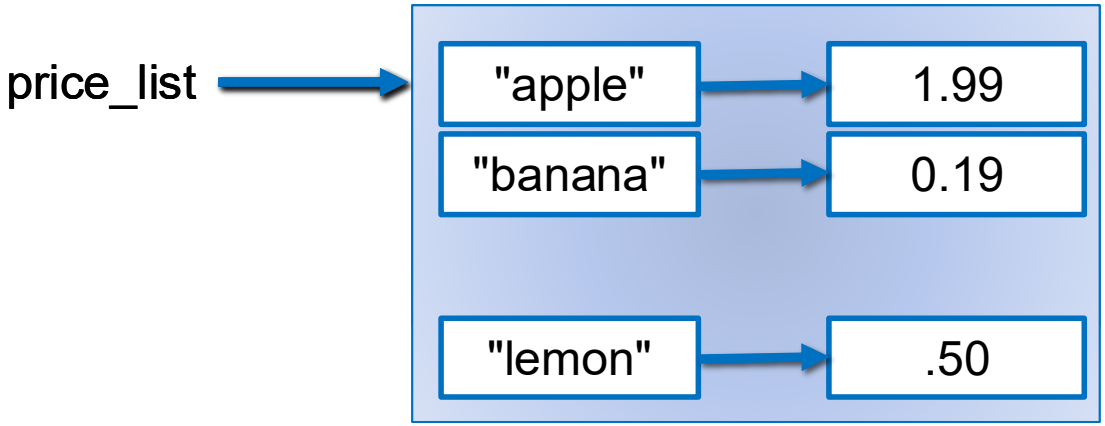
```
if "apple" in prices.keys():
    print("The price of an apple is", price)
```



Exercise 2

- Define a function `compute_cost` that takes two arguments, a dictionary `prices` and list of items `shopping_list` and returns the total cost of buying all the items on the shopping list. If an item on the shopping list is not in the prices dictionary, just exclude it from the total cost.

Modifying Dictionaries



- add: `d["lemon"] = .50`
- update: `d["apple"] = 1.99`
- delete: `d.pop("melon")` **returns 2.99**

Example

- Define a function `add_items` that takes two arguments, a dictionary `grocery_store` and list of item-price pairs new_items and adds the new items (with their associated prices) to the grocery store.

Exercise 3

```
def mystery(my_dict):
    d = {}
    for i in my_dict.keys():
        if my_dict[i] in d:
            d[my_dict[i]].append(i)
        else:
            d[my_dict[i]] = [i]
    return d

def main():
    d = {"a":1, "b":2, "c":1, "d":0, "e":2}
    print(mystery(d))

main()
```

Dictionary Operations

adding to a dictionary

- a_dict[key] = value
- a_dict.update(*b_dict*)

removing from a dictionary

- del (a_dict[*key*])
- a_dict.pop(*key*)
  - returns a_dict[key]

other

- len(a_dict)
- a_dict.keys()
  - returns list
- a_dict.values()
  - returns list
- a_dict.items()
  - returns list of tuples
- b_dict = a_dict.copy()
  - shallow copy!

Lists, dictionaries

- Both data structures.
- Why would you choose one over the other?
  - a data structure is something that holds a collection of data and that supports certain operations for working with that data

- Lists: sequential access
- Dictionaries: fast lookup

Example: Data Analytics

- Write a function `median` that takes a filename (assumed to be a one-column csv file) and returns the median of the numbers in the first column (excluding the header).

```
def mode(filename):
    counts = {}
    file = open(filename, "r")
    header = file.readline() # read the header
    for line in file:
        time = int(float(line)) # round down to hour
        if time in counts.keys():
            counts[time] = counts[time] + 1
        else:
            counts[time] = 1
    file.close()

    most_yet = None
    for (time, count) in counts.items():
        if most_yet == None or count > counts[most_yet]:
            most_yet = time
    return most_yet
```