# Lecture 17: Dictionaries

CS 51P

Fall 2023

# Example: Processing files

- Write a function `count_word` that takes a file and a word and returns the number of times that word appears in the file.

```
def count_word(file, word_in):
    count = 0
    lines = file.readlines()
    for line in lines:
        words = line.split()
        for word in words:
            w = word.strip(string.punctuation)
            if w == word_in:
                count = count + 1
    return count
```

- What if you wanted to process a file so you could answer repeated queries of the form "How many times does the word _____ appear?"

# 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

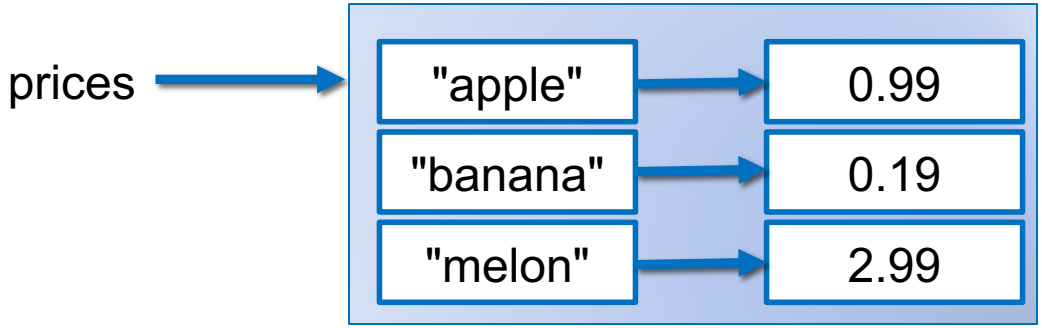
```
d = {'apple': .99, 'banana': .19, 'cantelope': 2.99}
```

- 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 common

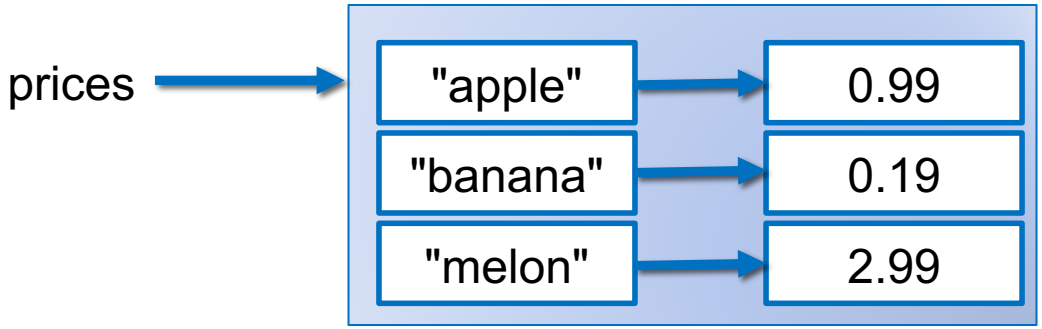
- Examples:
  - empty_dict = {}
  - phone_book = {"Joe": "909-607-9799", "Alexandra": "909-607-0969"}
  - prices = {"apple": .99, "banana": .19, "melon": 2.99}



# 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}



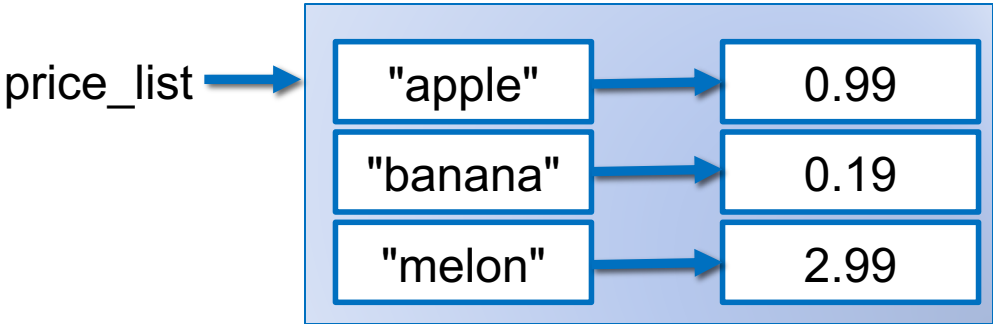

  - prices["apple"] is .99
  - prices["melon"] is 2.99
  - price["grape"] is ??? **KeyError**

# 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 `most_expensive` that takes one parameter, a dictionary `prices`, and returns the item with the highest price.

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

# Exercise

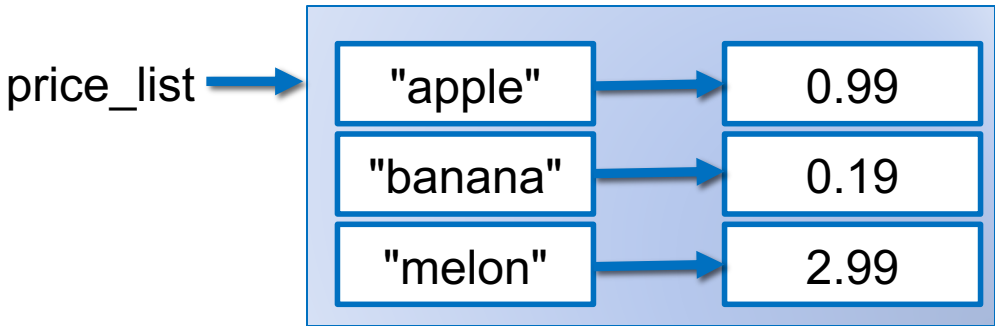
- Define a function `under_price(grocery_store, p)` that takes two parameters, a dictionary `grocery_store` and a price `p`, and returns the number of unique items available for less than (or equal to) price `p`.

# 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}

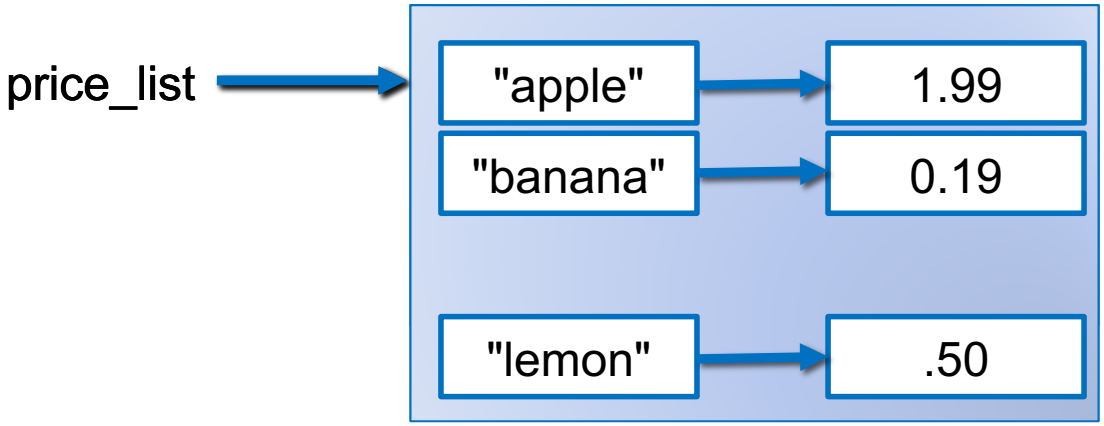
```
if "apple" in prices.keys():
    print("The price of an apple is", price)
```



# Exercise

- 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

```
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**

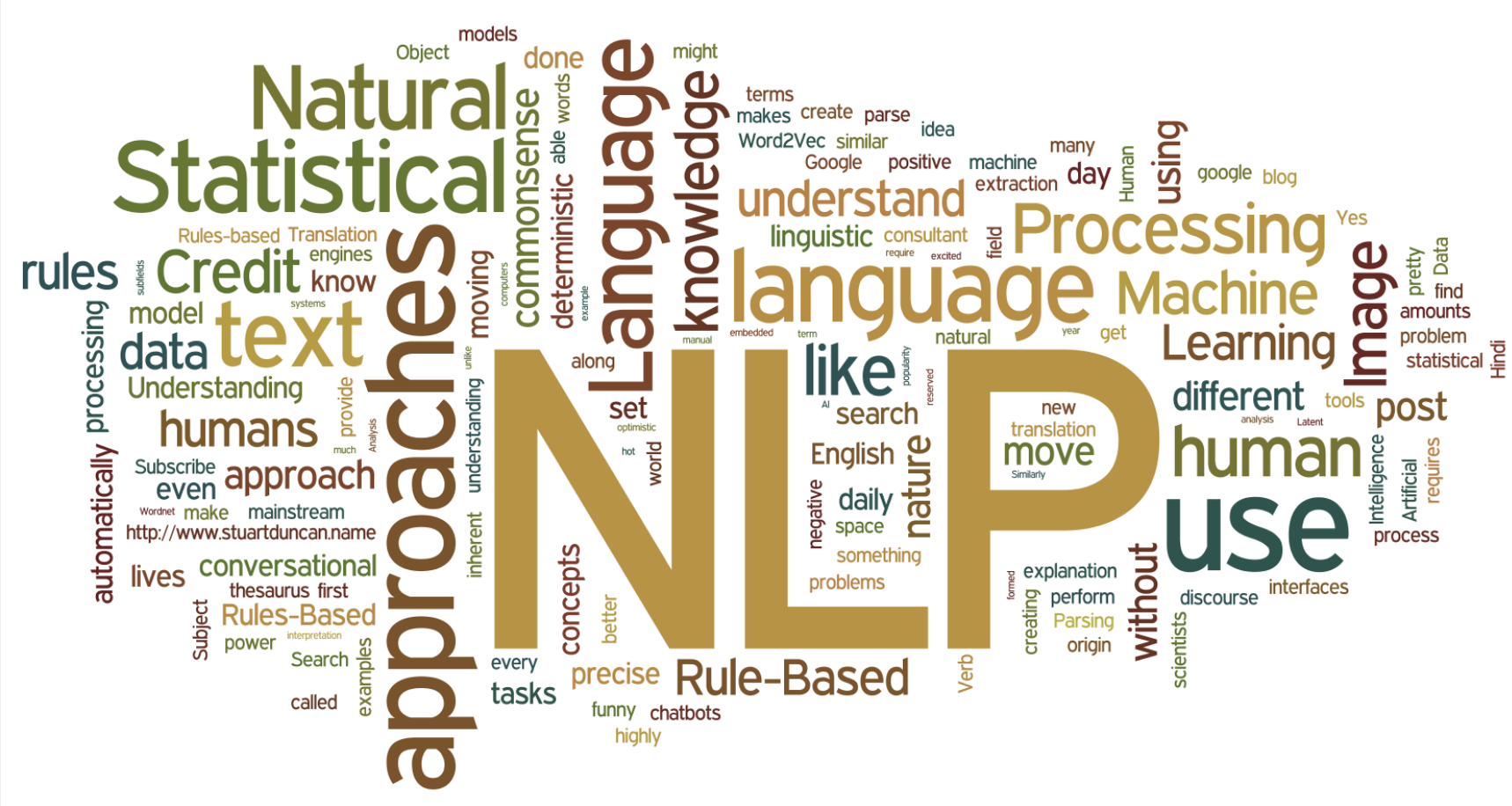
- len(a_dict)
- a_dict.keys()
  - returns list
- a_dict.values()
  - returns list
- a_dict.items()
  - returns list of tuples
- b_dict = a_dict.copy()
  - shallow copy!

# 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: Word Count




- Write a function that processes a file and returns a dictionary for handling repeated queries of the form "How many times does the word _____ appear?"

# Example: Word Count

- Write a function that processes a file and returns a dictionary for handling repeated queries of the form "How many times does the word _____ appear?"

```
def count_words(filename):
    counts = {}
    f = fopen("file.txt", "r")
    text = f.readlines()
    for line in text:
        words = line.split()
        for w in words:
            w2 = w.strip(string.punctuation)
            if w2 in word_counts:
                word_counts[w2] = word_counts[w2] + 1
            else:
                word_counts[w2] = 1
    f.close()
    return counts
```