Lone Star
section 4

Dictionaries quiz

1. (8.1) Write a **function** that takes a string *word* and returns a dictionary containing the count of each letter in the word.

   Examples:

   - letter_count("hello") → {"h": 1, "e": 1, "l": 2, "o": 1}
   - letter_count("mississippi") → {"m": 1, "i": 4, "s": 4, "p": 2}
   - letter_count("apple") → {"a": 1, "p": 2, "l": 1, "e": 1}

2. (8.2) Write a **function** named *majority_elment* that takes a list of integers named *nums* and returns the majority element. The majority element is the element that has at least half of the occurrences. You may assume that the majority element always exists and is unique.

   Examples:

   - majority_elment([3,2,3]) → 3
   - majority_elment([2,2,1,1,1,2,2]) → 2
   - majority_elment([2,2,3,2,1,2,1,4,4,1,2,2]) → 2

3. (8.3) Write a **function** that takes a dictionary, called *employee_salaries*, where the keys are employee names and the values are their salaries. The function should return a list of employees earning above a given salary.

   Examples:

   - high_earners({"Alice": 50000, "Bob": 75000, "Charlie": 100000}, 60000) → ["Bob", "Charlie"]
   - high_earners({"David": 30000, "Emma": 45000, "Frank": 50000}, 40000) → ["Emma", "Frank"]
   - high_earners({"George": 25000, "Hannah": 27000, "Ian": 29000}, 30000) → []

Dot Matrix
section 5

Dictionaries quiz

1. (8.1) Write a **function** that takes a dictionary, called *people*, containing the names and ages of a group of people, and returns the name of the oldest person.

   Examples:

   - find_oldest({"Emma": 71, "Jack": 45, "Olivia": 82, "Liam": 39}) → "Olivia"
   - find_oldest({"Sophia": 50, "Mason": 68, "Ava": 67, "Noah": 33}) → "Mason"
   - find_oldest({"Ethan": 25, "Lucas": 30, "Mia": 29}) → "Lucas"

2. (8.2) In each input list, every number repeats at least once, except for two. Write a **function** that takes an array *numbers* and returns the two unique numbers.

   Examples:

   - return_unique([1, 9, 8, 8, 7, 6, 1, 6]) → [9, 7],
   - return_unique([5, 5, 2, 4, 4, 4, 9, 9, 9, 1]) → [2, 1],
   - return_unique([9, 5, 6, 8, 7, 7, 1, 1, 1, 1, 1, 9, 8]) → [5, 6]

3. (8.3) Write a **function** that takes a dictionary called *names* of tech ids and student names as key-value pairs, and returns a list containing just the student names.

   Examples:

   - get_names({"01475": "Steve", "87469": "Alice", "654123": "Bob" }) → ["Steve", "Alice", "Bob"]
   - get_names({ "ID1": "John", "ID2": "Emma", "ID3": "Liam" }) → ["John", "Emma", "Liam"]
   - get_names({}) → []

Dark Helmet
section 4

Dictionaries quiz

1. (8.1) Write a **function** that takes a dictionary, called *people*, containing the names and ages of a group of people, and returns the name of the youngest person.

   Examples:

   - find_youngest({"Emma": 71, "Jack": 45, "Olivia": 82, "Liam": 39}) → "Liam"
   - find_youngest({"Sophia": 50, "Mason": 68, "Ava": 67, "Noah": 33}) → "Noah"
   - find_youngest({"Ethan": 25, "Lucas": 30, "Mia": 29}) → "Ethan"

2. (8.2) In each input list, every number repeats at least once, except for one. Write a **function** that takes an array *numbers* and returns the single unique number.

   Examples:

   - find_unique([1, 2, 2, 3, 3, 4, 4]) → 1,
   - find_unique([7, 8, 8, 9, 9, 10, 10]) → 7,
   - find_unique([5, 6, 6, 7, 7, 8, 8, 5, 9]) → 9

3. (8.3) Write a **function** that takes a dictionary called *names* of tech ids and student names as key-value pairs, and returns a list containing just the student names.

   Examples:

   - get_names({"01475": "Steve", "87469": "Alice", "654123": "Bob" }) → ["Steve", "Alice", "Bob"]
   - get_names({ "ID1": "John", "ID2": "Emma", "ID3": "Liam" }) → ["John", "Emma", "Liam"]
   - get_names({}) → []

President Skroob Dictionaries quiz
section 1

1. (8.1) Write a **function** that takes a list, called *elements*, and returns a dictionary detailing how many times each element is repeated.

   Examples:

   - count_repetitions(["cat", "dog", "cat", "cow", "cow", "cow"]) → { "cow": 3, "cat": 2, "dog": 1 }
   - count_repetitions([1, 5, 5, 5, 12, 12, 0, 0, 0, 0, 0, 0]) → { 0: 6, 5: 3, 12: 2, 1: 1 }
   - count_repetitions(["Infinity", "null", "Infinity", "null", "null"]) → { "null": 3, "Infinity": 2 }

2. (8.2) Write a **function** that takes a list of **fruits** and returns the total **caloric value** of the fruits consumed. You may use the following dictionary named *calories*:

   $$calories = \{ \text{"apple"} : 95, \text{"banana"} : 105, \text{"orange"} : 62, \text{"grape"} \ 3, \text{"pear"} : 102\}$$

   Hint: You can calculate the total calories by summing up the caloric values of all valid fruits in the list. You may assume the *calories* dictionary is defined in your code. You don't need to rewrite it.

   Examples:

   - total_calories(["apple", "banana", "orange"]) → 262 (since 95 + 105 + 62 = 262)
   - total_calories(["grape", "grape", "grape", "grape", "grape"]) → 15
   - total_calories(["banana", "pear", "apple"]) → 302

3. (8.3) Write a **function** that takes a dictionary, called *employee_salaries*, where the keys are employee names and the values are their salaries. The function should return a list of employees earning above a given salary.

   Examples:

   - high_earners({"Alice": 50000, "Bob": 75000, "Charlie": 100000}, 60000) → ["Bob", "Charlie"]
   - high_earners({"David": 30000, "Emma": 45000, "Frank": 50000}, 40000) → ["Emma", "Frank"]
   - high_earners({"George": 25000, "Hannah": 27000, "Ian": 29000}, 30000) → []