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. 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}

3. Write a **function** that takes a dictionary, called *exams*, containing the course grades of a student, and returns the name of the course with the minimal grade.

   Examples:

   - min_grade({"Physics": 82, "Math": 65, "History": 75, "Biology": 95, "English" : 87}) → "Math"
   - min_grade({"Chemistry": 78, "Algebra": 88, "History": 72, "Geography": 85}) → "History"
   - min_grade({"Art": 90, "Music": 92, "Drama": 89}) → "Drama"

4. 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"

5. 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 }

6. An isogram is a word that has no duplicate letters. Write a **function** that takes a string *word* and returns either True or False depending on whether or not it's an isogram.

   Examples:

   - is_isogram("algorism") → True
   - is_isogram("password") → False

- is_isogram("consecutive") → False

7. 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

8. 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]

9. Below is a receipt from my recent lunch order.

   (a) Initialize an empty dictionary named receipt, and then add the contents of the receipt as key-value pairs.

   (b) Using the dictionary you created in part a, write code that prints the total cost of all the items on the receipt. The code should work regardless of the contents of the receipt. (meaning don't write print(6+12+3))

Item	**Price**
Side Salad	$6
Chicken Parm	$12
Cookie	$3

10. Below is the menu from my favorite restaurant.

    (a) Initialize an empty dictionary named menu, and then add the contents of the menu as key-value pairs.

    (b) Using the dictionary you created in part a, write code that prints each of the items on the menu as key-value pairs. The code should work regardless of the contents of the receipt. (meaning don't write print("burger", 10))

Item	**Price**
burger	$10
fries	$4
soda	$3

```
Your output should be similar to this:
burger cost 10
fries cost 4
soda cost 3
```

11. Write a **function** that takes a dictionary, called *sales*, where the keys are product names and the values are the number of units sold. The function should return the total number of products sold.

    Examples:

    - total_sales({"Laptop": 5, "Phone": 10, "Tablet": 3}) → 18
    - total_sales({"Shoes": 20, "Hats": 15, "Jackets": 10}) → 45

- total_sales({"Book": 1, "Pen": 2, "Notebook": 1}) → 4

12. Write a **function** that takes a dictionary, called *donations*, where the keys are donor names and the values are the amount donated. The function should return the total amount donated.

    Examples:

    - total_donations({"John": 100, "Sarah": 200, "Mike": 50}) → 350
    - total_donations({"Anna": 500, "Tom": 1000, "Jerry": 1500}) → 3000
    - total_donations({"Chris": 25, "Alex": 30, "Morgan": 45}) → 100

13. 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

14. Write a **function** that takes a list of **ingredients** and returns the total **cost** of making a recipe. You may use the following dictionary named *prices*:

    prices = { "flour" : 2.50, "sugar" : 1.80, "eggs" : 3.00, "milk" : 2.00, "butter" : 2.75, "vanilla" : 4.50, "chocolate" : 5.00 }

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

    Examples:

    - total_cost(["flour", "sugar", "eggs", "butter"]) → 10.05
    - total_cost(["milk", "vanilla", "chocolate"]) → 11.50
    - total_cost(["eggs", "eggs", "flour", "sugar"]) → 10.30

15. 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

16. 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({}) → []

17. Write a **function** that takes a dictionary, called *store*, representing items and their prices, and an integer, called *wallet*, representing the amount of money you have. The function should return a list of items you can afford. If you cannot afford anything, return an empty list.

    Examples:

    - items_purchase({"Water": 1, "Bread": 3, "TV": 1000}, 300) → ["Bread", "Water"]
    - items_purchase({"Apple": 4, "Pan": 100, "Spoon": 2 }, 100) → ["Apple", "Pan", "Spoon"]
    - items_purchase({"Phone": 999, "Laptop": 5000, "PC": 1200 }, 1) → []

18. 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) → []