Worksheet: Lists

CS 101 Algorithmic Problem Solving

Fall 2023

Name(s): ______________________________

HU ID *(e.g., xy01042)*: ______________________________

1. Report Card

You've received your final report card and want to calculate your percentage. The report card contains the marks you received in each subject. The marks are out of 100. The percentage is calculated as follows:

$$\frac{\text{sum of marks}}{\text{total marks}} \times 100$$

Your marks are stored in a list *grades*. You need to write a function *calculate_percentage()* that has one parameter *grades* and returns the percentage you received.

Make sure to print the function call in your psuedocode.

Constraints

- *grades* is a list of integers between 0 and 100 (inclusive).

Interaction

The input comprises a single line containing a list of integers denoting the elements of *grades*.

The output must contain a single number denoting the percentage.

Sample

Input	Output
[80, 90, 100]	90.0
[92, 89, 100, 72, 88]	88.2

In the first case, the percentage is calculated as (80+90+100)/300 = 90.0.

In the second case, the percentage is calculated as (92+89+100+72+88)/500 = 88.2.

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
[34, 88, 91, 97, 89, 44]	73.8
[66, 77, 18, 45]	51.5
[92, 91, 100, 37, 86, 43, 59]	72.6

Problem Identification

Briefly explain the underlying problem you identified in the above question that led you to

your solution.
Input: `grades`
Output: sum of all elements of `grades` divided by the number of elements.

Pseudocode

```
def calculate_percentage(grades):
    total = 0
    for i in range(len(grades)):
        total += grades[i]
    return total/len(grades)
print(calculate_percentage(grades))
```

Dry Run
Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

```
1. Initialize the variable total as 0.
2. Loop through the indices of the grades list:
   - At the first iteration (i = 0), add the value of grades[0] (66) to the total,
    so total becomes 66.
   - At the second iteration (i = 1), add the value of grades[1] (77) to the total,
    so total becomes 66 + 77 = 143.
   - At the third iteration (i = 2), add the value of grades[2] (18) to the total,
    so total becomes 143 + 18 = 161.
   - At the fourth iteration (i = 3), add the value of grades[3] (45) to the total,
    so total becomes 161 + 45 = 206.
3. Calculate the average by dividing the total by the length of the grades list
(4 in this case): 206/4 = 51.5.
4. Print the calculated percentage (average) which is 51.5, which is the expected output!
```

2. Shopping Cart

You work at a supermarket's IT department. The supermarket gets thousands of customers every single day. A customer's shopping cart is stored in a list *cart*. You must write a function *get_distinct_items()* that has one parameter *cart* and returns a list of only the distinct items in the cart.

Make sure to print the function call in your pseudocode.

Note: You are not allowed to use the ***set*** *data structure.*

Constraints

- *cart* is a list of alphabets.

Interaction

The input comprises a single line containing a list of alphabets denoting the items of *cart*.

The output must be a list containing only the distinct elements from the cart.

Sample

Input	Output
[‘a’, ‘b’, ‘b’]	[‘a’, ‘b’]
[‘a’, ‘b’, ‘c’]	[‘a’, ‘b’, ‘c’]

In the first case, the cart contains three elements; however, ‘b’ is included twice and hence, will be omitted from the output list.

In the second case, the cart contains three elements and each alphabet appears only once. Hence, the output list contains all elements.

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
[‘b’, ‘a’, ‘b’, ‘c’, ‘d’, ‘a’, ‘e’]	[‘b’, ‘a’, ‘c’, ‘d’, ‘e’]
[‘a’, ‘b’, ‘b’, ‘c’, ‘c’, ‘c’, ‘d’, ‘d’, ‘d’, ‘d’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’]	[‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
[‘a’, ‘b’, ‘a’, ‘c’, ‘c’, ‘b’, ‘a’, ‘e’, ‘k’, ‘h’, ‘f’, ‘j’, ‘i’, ‘a’, ‘f’, ‘l’, ‘k’, ‘e’, ‘i’, ‘j’, ‘b’, ‘h’, ‘c’, ‘a’, ‘l’, ‘r’]	[‘a’, ‘b’, ‘c’,‘e’, ‘k’, ‘h’, ‘f’, ‘j’, ‘i’, ‘l’, ‘r’]

Problem Identification

Briefly explain the underlying problem you identified in the above question that led you to your solution.
Input: `cart`
Output: unique elements of list `cart`.

Pseudocode

```
def get_distinct_items(cart):
    unique_list = []
    for i in range(len(cart)):
        if cart[i] not in unique_list:
            unique_list.append(cart[i])
    return unique_list
print(get_distinct_items(cart))
```

Dry Run
Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

```
1. Initialize an empty list unique_list to store unique items.
2. Loop through the indices of the cart list:
   - At the first iteration (i = 0), check if ’b’ is not in unique_list,
    which is True. So, ’b’ is added to unique_list.
   - At the second iteration (i = 1), check if ’a’ is not in unique_list,
    which is True. So, ’a’ is added to unique_list.
   - At the third iteration (i = 2), check if ’b’ is not in unique_list,
    which is False (as ’b’ is already in unique_list). So, ’b’ is not added again.
   - At the fourth iteration (i = 3), check if ’c’ is not in unique_list,
    which is True. So, ’c’ is added to unique_list.
   - At the fifth iteration (i = 4), check if ’d’ is not in unique_list,
    which is True. So, ’d’ is added to unique_list.
   - At the sixth iteration (i = 5), check if ’a’ is not in unique_list,
    which is False (as ’a’ is already in unique_list). So, ’a’ is not added again.
   - At the seventh iteration (i = 6), check if ’e’ is not in unique_list,
    which is True. So, ’e’ is added to unique_list.
3. Return the unique_list containing the distinct items: [’b’, ’a’, ’c’, ’d’, ’e’].
4. Print the resulting list [’b’, ’a’, ’c’, ’d’, ’e’] which is the expected output!
```

3. Cyclic Card Shuffling

You are playing a card game with your friends. You have a deck of cards where each card has a number written on it. You want to shuffle the cards in a cyclic manner. This means that the first card will be moved to the last position, the second card will be moved to the first position, the third card will be moved to the second position, and so on. You need to write a function *cyclic_shuffle()* that takes two arguments: a list *cards* containing the cards and an integer X and returns the list of cards after being shuffled X times.

Make sure to print the function call in your pseudocode.

Constraints

- *cards* is a list of integers between 0 and 9 (inclusive).
- $X \in \mathbb{Z}^+$
- $0 \leq X \leq 10^5$

Interaction

The input comprises of two lines. The first line contains the list comprising the elements of *cards*. The second line contains an integer denoting the value of X.

The output must be a list containing the cards after being shuffled X times.

Sample

Input	Output
[5, 4, 3, 2, 1]	[4, 3, 2, 1, 5]
1	
[8, 3, 4, 1, 9, 7]	[1, 9, 7, 8, 3, 4]
3	

In the first case, the deck is the list [5, 4, 3, 2, 1]. After performing 1 cyclic shift, the values shift one to the left (and the first value goes to the end). Hence, we get the output list as [4, 3, 2, 1, 5].

In the second case, the deck is the list [8, 3, 4, 1, 9, 7]. After performing 1 cyclic shift, the values shift one to the left and we get [3, 4, 1, 9, 7, 8]. After 2 cyclic shifts, we get [4, 1, 9, 7, 8, 3]. After 3 cyclic shifts, we get the output list as [1, 9, 7, 8, 3, 4].

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
[4, 6, 8, 3, 1, 5, 3]	[1, 5, 3, 4, 6, 8, 3]
4	
[2, 9, 4]	[9, 4, 2]
7	
[1, 2]	[2, 1]
13	

Problem Identification

Briefly explain the underlying problem you identified in the above question that led you to your solution.

Input: `cart,X`

Output: The list with elements shifted to the left `X` times.

Pseudocode

```python
def cyclic_shuffle(cards, X):
    n = len(cards)
    if n == 0:
        return cards
    X = X % n  # To handle cases where X is greater than
    #the number of cards
    if X == 0:
        return cards
    shuffled_cards = cards[X:] + cards[:X]
    return shuffled_cards
print(cyclic_shuffle(cards, X))
```

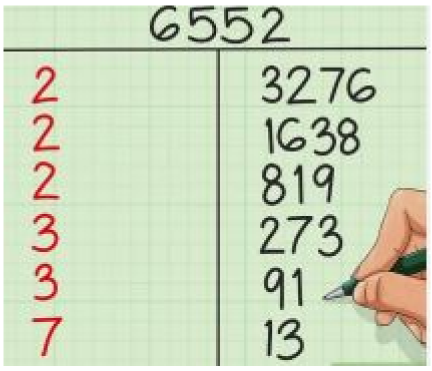
Dry Run
Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.
Input: `cards = [8, 3, 4, 1, 9, 7], X =3`

```
1. Calculate the length of the list cards. Here, the length is 6.
2. Check if the length is 0. It is not, so we proceed.
3. Calculate X modulo the length of cards, which is 3 % 6 = 3.
4. Check if the result of the modulo operation is 0. It is not, so we proceed.
5. Perform the cyclic shuffling: shuffled_cards = cards[3:] + cards[:3].
   - cards[3:] gives the sublist starting from index 3: [1, 9, 7].
   - cards[:3] gives the sublist up to index 3 (excluding): [8, 3, 4].
   - Concatenating these two sublists gives the shuffled cards: [1, 9, 7, 8, 3, 4].
6. Return the shuffled cards.
7. Print the resulting shuffled cards: [1, 9, 7, 8, 3, 4] which is the expected output!
```

4. Prime Factorization Theorem

Your maths professor has just taught you the prime factorization theorem. The theorem states that any integer greater than 1 is either prime itself or is the product of a unique combination of prime numbers.

You want to write a function *prime_factorization()* that takes an integer N and returns a list of all the prime factors of N.



Make sure to print the function call in your pseudocode.

Constraints

- $N \in \mathbb{Z}^+$
- $2 \leq N \leq 10^5$

Interaction

The input comprises of a single line containing an integer N.

The output must be a list containing the prime factors of N.

Sample

Input	Output
6552	[2, 2, 2, 3, 3, 7, 13]
756	[2, 2, 3, 3, 3, 7]

In the first case, 6552 = 2 * 2 * 2 * 3 * 3 * 7 * 13 and hence, the output is [2, 2, 2, 3, 3, 7, 13].

In the second case, 756 = 2 * 2 * 3 * 3 * 3 * 7 and hence, the output is [2, 2, 3, 3, 3, 7].

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
672	[2, 2, 2, 2, 2, 3, 7]
169	[13, 13]
1430	[2, 5, 11, 13]

Problem Identification

Briefly explain the underlying problem you identified in the above question that led you to your solution.

Input: `N`
Output: prime factors of `N`

Pseudocode

```
def prime_factorization(N):
    factors = []
    divisor = 2
    while N > 1:
        while N % divisor == 0:
            factors.append(divisor)
            N //= divisor
        divisor += 1
    return factors
print(prime_factorization(N))
```

Dry Run
Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

1. Set the variable factors as an empty list.
2. Set the variable divisor as 2.
3. Enter the while loop since N is greater than 1.
4. Check if 1430 is divisible by 2. Since it is divisible, append 2 to the factors list, and update N to 1430 // 2 = 715.
5. Check if 715 is divisible by 2. It is not, so increment the divisor to 3.
6. Check if 715 is divisible by 3. It is not, so increment the divisor to 4.
7. Continue this process until you find a divisor that divides 715 without leaving a remainder. In this case, it is 5. Append 5 to the factors list and update N to 715 // 5 = 143.
8. Continue this process until you find a divisor that divides 143 without leaving a remainder. In this case, it is 11. Append 11 to the factors list and update N to 143 // 11 = 13.
9. Continue this process until you find a divisor that divides 13 without leaving a remainder. In this case, it is 13 itself. Append 13 to the factors list and update N to 13 // 13 = 1.
10. As N is now 1, the while loop stops.
11. Return the list factors, which contains the prime factors of 1430: [2, 5, 11, 13].
12. Print the list of prime factors: [2, 5, 11, 13], which is the expected output!

LET’S LEARN TO DEBUG

5. Good Deal

Going back to your job at the supermarket’s IT department, you have a new task. You want to figure out which items are typically purchased in bulk. You have a list *cart* containing the items purchased by a customer. You want to write a function *get_repeated_items()* that takes one argument *cart* and returns a list of all the items that are purchased more than once (there should be atmost one occurence of any item).

Make sure to print the function call in your pseudocode.

Constraints

- *cart* is a list of alphabets.

Interaction

The input comprises a list containing alphabets denoting the items of *cart*.

The output must be a list containing the items that are purchased more than once. Note that the output should contain only one occurence of the item.

Sample

Input	Output
[‘a’, ‘b’, ‘b’]	[‘b’]
[‘a’, ‘b’, ‘c’]	[ ]

In the first case, ‘a’ is purchased once and ‘b’ is purchased twice; hence, the output list is [‘b’].

In the second case, neither ‘a’, ‘b’, nor ‘c’ are purchased more than once; hence, the output list is empty.

Proposed Solution

```
def get_repeated_items(cart):
  repeated_items = []
  for item in range(len(cart)):
    if item in cart:
      repeated_items.append(item)

  return repeated_items

print(get_repeated_items(cart))
```

Dry Run
Using the inputs provided in the Sample section above, dry run the proposed code solution below.

```
    Given input: cart = [‘a’, ‘b’, ‘b’]

1. Initialize an empty list repeated_items.
2. Loop through the indices of cart (0, 1, 2).
3. At index 0, the item is ‘a’. Check if 0 is in cart, which is False.
So, do not append anything.
4. At index 1, the item is ‘b’. Check if 1 is in cart, which is False.
```

```
So, do not append anything
5. At index 2, the item is ‘b’. Check if 2 is in cart, which is False.
So, do not append anything
6. Exit the loop.
7. Return the list repeated_items
This is not the expected output. The expected output is
the repeated elements of list.
```

Error Identification

Briefly explain the errors you identified in the proposed code solution. Mention the line number and the errors in each line.

1. Instead of checking actual element of list, the code checks if index of element is in cart.
2. For each element, the rest of the list needs to be checked if element is present elsewhere. This determines repeated elements.
3. There is no condition added which checks if element has already been added to `repeated_items` .

Correct Solution

Rewrite the lines of code you mentioned above with their errors corrected.

```python
def get_repeated_items(cart):
    repeated_items = []
    for item in range(len(cart)):
        if cart[item] in cart[item+1:] and cart[item] not in repeated_items:
            repeated_items.append(cart[item])
    return repeated_items
print(get_repeated_items(cart))
```

Rough Work