**Python Programming Challenges: A Practical Assessment**

**Instructions:** For each question, write a Python program that fulfills the requirements. Aim for clean, readable code with meaningful variable names, comments where necessary, and appropriate use of functions to promote modularity.

**Question 1: List Manipulation and Conditionals**

**Problem:** Write a Python function called filter\_numbers that takes a list of integers as input. The function should return a new list containing only the numbers from the input list that are both positive and even.

**Requirements:**

* The function must accept one argument: numbers (a list of integers).
* It should return a new list.
* Use a for loop and an if statement for filtering.

Example Input:

my\_list = [-2, 0, 1, 4, 7, 10, 15, 20]

Expected Output:

[4, 10, 20]

**Hints/Considerations:**

* Remember how to check if a number is even (% operator).
* Think about how to build a new list by appending elements.

**Question 2: String Processing and Slicing**

**Problem:** Create a Python function named reverse\_words that takes a sentence (a string) as input. The function should reverse the order of words in the sentence, but not the characters within each word.

**Requirements:**

* The function must accept one argument: sentence (a string).
* It should return the modified string.
* Assume words are separated by single spaces.

Example Input:

text = "Python is fun"

Expected Output:

"fun is Python"

**Hints/Considerations:**

* How can you split a string into a list of words?
* How can you reverse the order of elements in a list?
* How can you join a list of strings back into a single string with spaces?

**Question 3: Looping with break and continue**

Problem: Write a Python function called find\_first\_vowel\_index that takes a string (a word) as input. The function should return the index of the first vowel ('a', 'e', 'i', 'o', 'u' - case-insensitive) found in the word. If no vowel is found, it should return -1.

Additionally, modify the function to print a message "Skipping non-alphabetic character" if it encounters any character that is not an alphabet (using continue), but still proceed with the search.

**Requirements:**

* The function must accept one argument: word (a string).
* It should return an integer (index or -1).
* Use a for loop with enumerate to get both index and character.
* Implement break to stop searching once the first vowel is found.
* Implement continue to skip non-alphabetic characters.

Example Input 1:

text1 = "rhythm"

Expected Output 1:

-1

Example Input 2:

text2 = "strength"

Expected Output 2:

3 (index of 'e')

Example Input 3 (for continue):

text3 = "pyth0n"

Expected Output 3 (for print statement and return):

Skipping non-alphabetic character

-1 (as no vowel is found after skipping)

**Hints/Considerations:**

* Create a set or string of vowels for efficient checking.
* Use char.isalpha() to check if a character is an alphabet.
* Use char.lower() for case-insensitivity.

**Question 4: Function with Multiple Arguments and Default Values**

**Problem:** Create a function calculate\_order\_total that calculates the total cost of an order. The function should take price and quantity as required arguments. It should also have an optional argument discount\_percentage which defaults to 0 if not provided.

**Requirements:**

* The function must accept price (float or int), quantity (int).
* It must have discount\_percentage (float or int) as an optional argument with a default value of 0.
* The function should return the final calculated total (float).
* Ensure the calculated total is non-negative.

Example Input 1:

total1 = calculate\_order\_total(price=100, quantity=2)

Expected Output 1:

200.0

Example Input 2:

total2 = calculate\_order\_total(price=50, quantity=3, discount\_percentage=10)

Expected Output 2:

135.0 (50 \* 3 = 150; 150 \* 0.10 = 15; 150 - 15 = 135)

**Hints/Considerations:**

* How do you define default arguments in Python functions?
* Handle the discount calculation correctly (e.g., total \* (1 - discount\_percentage / 100)).

**Question 5: Basic Object-Oriented Programming (OOP)**

**Problem:** Design a simple Python class named Book to represent a book. The class should have attributes for title, author, and pages. It should also have a method get\_info that returns a formatted string containing the book's details.

**Requirements:**

* Define a class Book with an \_\_init\_\_ method that takes title, author, and pages as arguments and initializes corresponding instance variables.
* Implement a method get\_info() that returns a string like: "Title: <title>, Author: <author>, Pages: <pages>".
* Create at least two instances (objects) of the Book class and call their get\_info() methods.

**Example Usage:**

Python

book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", 180)

book2 = Book("1984", "George Orwell", 328)

print(book1.get\_info())

print(book2.get\_info())

Expected Output (similar to):

Title: The Great Gatsby, Author: F. Scott Fitzgerald, Pages: 180

Title: 1984, Author: George Orwell, Pages: 328

**Hints/Considerations:**

* Remember the self keyword in class methods.
* Use f-strings for easy string formatting.

**Question 6: Using List Comprehensions (Functional Flavor)**

**Problem:** Write a Python function square\_positive\_numbers that takes a list of integers. Using a list comprehension, it should return a new list containing the squares of only the positive numbers from the input list.

**Requirements:**

* The function must accept one argument: numbers (a list of integers).
* It should return a new list.
* **Must use a list comprehension.**

Example Input:

data = [-3, -1, 0, 2, 5, 8]

Expected Output:

[4, 25, 64] (Squares of 2, 5, 8)

**Hints/Considerations:**

* Recall the syntax for list comprehensions with conditional filtering.
* This demonstrates a more "functional" approach by transforming a list without explicit for loops and if statements (though they are implicitly part of the comprehension).

Okay, here's another set of Python programming questions with their solutions, building on the previous topics and introducing some new concepts like dictionaries and basic error handling.

**Python Programming Challenges: Set 2**

**Instructions:** For each question, write a Python program that fulfills the requirements. Aim for clean, readable code with meaningful variable names, comments where necessary, and appropriate use of functions to promote modularity.

**Question 7: Dictionary Manipulation and Counting**

**Problem:** Write a Python function called count\_word\_frequency that takes a list of strings (words) as input. The function should return a dictionary where keys are the unique words (case-insensitive) and values are their respective frequencies (counts) in the input list.

**Requirements:**

* The function must accept one argument: words\_list (a list of strings).
* It should return a dictionary.
* Word counting should be case-insensitive (e.g., "Apple" and "apple" should be counted as the same word).

Example Input:

word\_data = ["apple", "banana", "Apple", "orange", "banana", "apple"]

Expected Output:

{'apple': 3, 'banana': 2, 'orange': 1} (Order of keys might vary)

**Hints/Considerations:**

* How do you convert a string to lowercase?
* How do you add/update entries in a dictionary?
* Consider using dictionary.get(key, default\_value) to simplify counting.

**Question 8: List Manipulation and Set Operations (or Looping)**

**Problem:** Create a Python function named get\_common\_elements that takes two lists as input. The function should return a new list containing only the elements that are present in *both* input lists, without any duplicates. The order of elements in the output list does not matter.

**Requirements:**

* The function must accept two arguments: list1 and list2 (both lists).
* It should return a new list with common, unique elements.
* The solution should be efficient.

Example Input:

list\_a = [1, 2, 3, 4, 5]

list\_b = [4, 5, 6, 7, 8]

Expected Output:

[4, 5] (or [5, 4])

**Hints/Considerations:**

* Think about Python's set data structure and its operations (& for intersection). Sets are very efficient for uniqueness and common element finding.
* Alternatively, you can use nested loops with conditional checks and a separate list to track added elements for uniqueness.

**Question 9: Basic Error Handling (Try-Except)**

**Problem:** Write a Python function called safe\_divide that takes two arguments, numerator and denominator. The function should perform division and return the result. However, it must handle potential ZeroDivisionError if the denominator is zero and TypeError if either input is not a number (e.g., a string). In case of an error, it should print an appropriate error message and return None.

**Requirements:**

* The function must accept two arguments: numerator and denominator.
* It should return the division result (float) or None if an error occurs.
* Use try-except blocks to handle specific exceptions.

Example Input 1:

result1 = safe\_divide(10, 2)

Expected Output 1:

5.0

Example Input 2:

result2 = safe\_divide(10, 0)

Expected Output 2 (prints and returns):

Error: Cannot divide by zero.

None

Example Input 3:

result3 = safe\_divide("abc", 5)

Expected Output 3 (prints and returns):

Error: Invalid input types. Please provide numbers.

None

**Hints/Considerations:**

* How do you catch multiple specific exceptions in except blocks?
* What happens if you try to divide a string?

**Question 10: String Formatting and Conditional Logic**

**Problem:** Create a function format\_contact\_info that takes a name (string), phone\_number (string), and an optional email (string, default to empty string). The function should return a formatted string containing the contact information. If email is provided, include it; otherwise, just include name and phone. Also, if the phone number is not exactly 10 digits long (after removing non-digit characters), print a warning message "Invalid phone number format."

**Requirements:**

* The function accepts name, phone\_number (required), and email (optional, default "").
* It returns a formatted string.
* Check phone number length (after stripping non-digits) and print a warning if invalid, but still return the formatted string.

Example Input 1:

info1 = format\_contact\_info("Alice Smith", "123-456-7890")

Expected Output 1:

Contact: Alice Smith, Phone: 123-456-7890

(Prints warning if phone not 10 digits after cleaning)

Example Input 2:

info2 = format\_contact\_info("Bob Johnson", "9876543210", "bob@example.com")

Expected Output 2:

Contact: Bob Johnson, Phone: 9876543210, Email: bob@example.com

**Hints/Considerations:**

* How can you remove non-digit characters from a string (e.g., spaces, hyphens)? Loops or string methods.
* Use an if condition to check if email is not empty.