Знакомство с языком Python (семинары)

Задание 1. Три списка

Даны три списка.

`array_1 = [1, 5, 10, 20, 40, 80, 100]`

`array_2 = [6, 7, 20, 80, 100]`

`array_3 = [3, 4, 15, 20, 30, 70, 80, 120]`

Нужно выполнить две задачи:

1. найти элементы, которые есть в каждом списке;
2. найти элементы из первого списка, которых нет во втором и третьем списках.

Каждую задачу нужно выполнить двумя способами:

1. без использования множеств;
2. с использованием множеств.

Пример выполнения на других данных:

`array_1 = [1, 2, 3, 4]`

`array_2 = [2, 4]`

`array_3 = [2, 3]`

Вывод:

Задача 1:

Решение без множеств: 2

Решение с множествами: 2

Задача 2:

Решение без множеств: 1

Решение с множествами: 1

Подсказка № 1

Для выполнения первой задачи без использования множеств, объедините все три списка в один и проверьте наличие элементов в каждом из них. Это можно сделать с помощью цикла и проверки каждого элемента на присутствие в исходных списках.

Подсказка № 2

Используйте логическое выражение `all` для проверки наличия элемента в каждом из списков. Это поможет вам избежать множественных проверок внутри цикла и сделать код более читаемым.

Подсказка № 3

Для решения первой задачи с использованием множеств, используйте операцию пересечения (`&`) для нахождения общих элементов. Преобразуйте списки в множества и примените оператор пересечения, чтобы найти элементы, которые есть во всех трех списках.

Подсказка № 4

Для второй задачи без использования множеств, создайте новый список, в который добавляйте только те элементы из первого списка, которых нет во втором и третьем списках. Используйте операторы `not in` для проверки отсутствия элемента.

Подсказка № 5

Для решения второй задачи с использованием множеств, используйте операцию разности (`-`) между множествами. Преобразуйте списки в множества и примените оператор разности для нахождения элементов из первого списка, которые отсутствуют во втором и третьем списках.

Эталонное решение:

```
# Даны три списка

array_1 = [1, 5, 10, 20, 40, 80, 100]

array_2 = [6, 7, 20, 80, 100]

array_3 = [3, 4, 15, 20, 30, 70, 80, 120]


# Задача 1: найти элементы, которые есть в каждом списке


# Решение без использования множеств
```

```
all_elems = array_1 + array_2 + array_3  # Объединяем все списки в один

new_elems_1 = []

for elem in all_elems:

    # Проверяем, если элемент не добавлен ранее и присутствует во всех трех списках

    if elem not in new_elems_1 and all(elem in array for array in [array_1, array_2, array_3]):

        new_elems_1.append(elem)  # Добавляем элемент в результат

print("Решение без множеств:", new_elems_1)



# Решение с использованием множеств

new_elems_1_set = set(array_1) & set(array_2) & set(array_3)  # Пересечение трех множеств

print("Решение с множествами:", new_elems_1_set)



# Задача 2: найти элементы из первого списка, которых нет во втором и третьем списках



# Решение без использования множеств

new_elems_2 = []

for elem in array_1:

    # Проверяем, если элемент отсутствует во втором и третьем списках

    if elem not in array_2 and elem not in array_3:

        new_elems_2.append(elem)  # Добавляем элемент в результат

print("Решение без множеств:", new_elems_2)



# Решение с использованием множеств
```

```
new_elems_2_set = set(array_1) - set(array_2) - set(array_3)  # Разность множеств

print("Решение с множествами:", new_elems_2_set)
```

Задача 2. Палиндром

Пользователь вводит строку. Необходимо написать программу, которая определяет, существует ли у этой строки перестановка, при которой она станет палиндромом. Затем она должна выводить соответствующее сообщение.

Пример 1

Введите строку: aab

Можно сделать палиндромом

Пример 2

Введите строку: aabc

Нельзя сделать палиндромом

Подсказка № 1

Используйте цикл для прохода по каждому символу строки. Для каждого символа увеличивайте его счетчик в словаре. Для этого можно использовать метод `get` словаря, который позволяет избежать ошибки, если символ еще не добавлен в словарь.

Подсказка № 2

После подсчета количества вхождений всех символов, пройдите по значениям словаря и посчитайте, сколько из них имеют нечетное количество вхождений. Для этого используйте оператор `%` для определения четности количества.

Подсказка № 3

Для строки, чтобы быть палиндромом, все символы должны быть сбалансированы по количеству, за исключением, возможно, одного символа. Проверьте, не превышает ли количество символов с нечетными вхождениями один. Это условие должно быть выполнено для возможности создания палиндрома.

Подсказка № 4

Функция `is_poly` должна возвращать `True`, если количество символов с нечетными вхождениями не превышает одного, и `False` в противном случае. В основной части

программы, после вызова функции, выводите соответствующее сообщение на основе ее результата.

Эталонное решение:

```
def is_poly(string):

    # Создаем словарь для подсчета количества вхождений каждого
символа

    char_dict = dict()

    # Проходим по каждому символу в строке

    for i_sym in string:

        # Увеличиваем счетчик для текущего символа

        char_dict[i_sym] = char_dict.get(i_sym, 0) + 1



    # Переменная для подсчета символов с нечетным количеством
вхождений

    odd_count = 0

    # Проходим по значениям словаря (количеству вхождений символов)

    for i_value in char_dict.values():

        # Если количество вхождений нечетное, увеличиваем счетчик
нечетных

        if i_value % 2 != 0:

            odd_count += 1



    # Палиндром может быть составлен, если не более одного символа
имеет нечетное количество вхождений

    return odd_count <= 1



# Запрашиваем у пользователя ввод строки

my_string = input('Введите строку: ')

# Проверяем, можно ли сделать палиндром из введенной строки
```

```
if is_poly(my_string):

    print('Можно сделать палиндромом')

else:

    print('Нельзя сделать палиндромом')
```

Задача 3. Словарь синонимов

Одна библиотека поручила вам написать программу для оцифровки словарей синонимов. На вход в программу подаётся N пар слов. Каждое слово является синонимом для своего парного слова.

Реализуйте код, который составляет словарь синонимов (все слова в словаре различны), затем запрашивает у пользователя слово и выводит на экран его синоним. Обеспечьте контроль ввода: если такого слова нет, выведите ошибку и запросите слово ещё раз. При этом проверка не должна зависеть от регистра символов.

Пример

Введите количество пар слов: 3

Первая пара: Привет — Здравствуйте

Вторая пара: Печально — Грустно

Третья пара: Весело — Радостно

Введите слово: интересно

Такого слова в словаре нет.

Введите слово: здравствуйте

Синоним: Привет

Подсказка № 1

Перед созданием словаря синонимов, убедитесь, что вы корректно принимаете пары слов. Разделите входные данные по символу « - » и приведи слова к нижнему регистру для унификации ввода. Это позволит избежать ошибок, связанных с разным регистром символов.

Подсказка № 2

Для добавления слов в словарь, каждую пару слов следует добавить как ключ и значение в словарь. Не забудьте добавить оба направления синонимов (например, если «Привет» является синонимом для «Здравствуйте», то «Здравствуйте» также должно быть синонимом для «Привет»).

Подсказка № 3

После создания словаря синонимов, запускайте бесконечный цикл для ввода слова от пользователя. Убедитесь, что слово, введенное пользователем, также приводится к нижнему регистру для проверки.

Эталонное решение:

```
# Создаем пустой словарь для хранения синонимов

synonyms_dict = dict()



# Запрашиваем количество пар слов у пользователя

pairs_count = int(input('Введите количество пар слов: '))



# Проходим по каждой паре слов

for i_pair in range(pairs_count):

    # Запрашиваем пару слов, разделенных " - "

    # Приводим слова к нижнему регистру для корректной работы с
различными регистрами

    first_word, second_word = input(f'{i_pair + 1} пара:
').lower().split(' - ')

    # Добавляем пары в словарь

    synonyms_dict[first_word] = second_word

    synonyms_dict[second_word] = first_word



# Запускаем бесконечный цикл для ввода слова и поиска синонима

while True:

    # Запрашиваем слово у пользователя и приводим его к нижнему
регистру
```

```
    word = input('Введите слово: ').lower()

    # Проверяем, есть ли слово в словаре

    if word in synonyms_dict:

        # Если есть, выводим синоним, приводя его к начальной букве заглавной

        print('Синоним: ', synonyms_dict[word].capitalize())

        break

    else:

        # Если нет, выводим сообщение об ошибке

        print('Такого слова в словаре нет.')
```

Задача 4. Гистограмма частоты

Создайте программу для лингвистов, которая будет инвертировать полученный словарь. То есть в качестве ключа будет частота, а в качестве значения — список символов с этой частотой. Вам нужно реализовать:

1. получить текст и создать из него оригинальный словарь частот;
2. создать новый словарь и заполнить его данными из оригинального словаря частот, используя количество повторов в качестве ключей, а буквы — в качестве значений, добавляя их в список для хранения.

Пример

Введите текст: здесь что-то написано

Оригинальный словарь частот:

: 2

- : 1

3 : 1

а : 2

д : 1

е : 1

и : 1

н : 2

о : 3

п : 1

с : 2

т : 2

ч : 1

ь : 1

Инвертированный словарь частот:

1 : ['З', 'д', 'е', 'ь', 'ч', '-', 'п', 'и']

2 : ['с', ' ', 'т', 'н', 'а']

3 : ['о']

Подсказка № 1

Перед созданием словаря частот, убедитесь, что ввод текста корректно обрабатывается. Сначала получите текст от пользователя, а затем создайте словарь частот, где ключами будут символы, а значениями — количество их вхождений в текст.

Подсказка № 2

При создании словаря частот используйте цикл, чтобы пройти по каждому символу строки и обновить частоту его появления. Если символ уже присутствует в словаре, увеличьте его частоту. Если нет, добавьте символ в словарь с частотой 1.

Подсказка № 3

После создания оригинального словаря частот, создайте инвертированный словарь, где ключами будут частоты, а значениями — списки символов, которые имеют эти частоты. Пройдите по оригинальному словарю и для каждого символа добавляйте его в соответствующий список по частоте.

Подсказка № 4

Для вывода данных сначала отсортируйте ключи словаря перед печатью. Это позволит вам вывести символы и их частоты в упорядоченном виде. Используйте функцию `sorted()` для сортировки ключей словаря.

Подсказка № 5

При инвертировании словаря частот, если частота уже присутствует в инвертированном словаре, добавляйте новые символы в существующий список. Если частота не присутствует, создайте новый список для этой частоты и добавьте в него символ.

Эталонное решение:

```
# Функция для создания словаря частот символов

def histogram(string):

    sym_dict = dict()  # Инициализируем пустой словарь для частот

    for sym in string:

        # Если символ уже есть в словаре, увеличиваем его частоту

        if sym in sym_dict:

            sym_dict[sym] += 1

        else:

            # Если символа нет в словаре, добавляем его с частотой 1

            sym_dict[sym] = 1

    return sym_dict



# Функция для инвертирования словаря частот

def invert_dict(d):

    inv = dict()  # Инициализируем пустой словарь для
инвертированных данных

    for key in d:

        val = d[key]

        # Если частота еще не встречалась, создаем новый список

        if val not in inv:
```

```
            inv[val] = [key]
        else:
            # Если частота уже есть в словаре, добавляем символ в существующий список
            inv[val].append(key)
    return inv

# Запрашиваем текст у пользователя
text = input('Введите текст: ')
# Создаем словарь частот
hist = histogram(text)

# Выводим оригинальный словарь частот
print('Оригинальный словарь частот:')
for i_sym in sorted(hist.keys()):
    print(i_sym, ':', hist[i_sym])

# Создаем инвертированный словарь частот
inv_hist = invert_dict(hist)

# Выводим инвертированный словарь частот
print('\nИнвертированный словарь частот:')
for i_sym in sorted(inv_hist.keys()):
    print(i_sym, ':', inv_hist[i_sym])
```