# Знакомство с языком Python (семинары)

### Задание 1. Новые списки

Даны три списка:

1. floats: List[float] = [12.3554, 4.02, 5.777, 2.12, 3.13, 4.44, 11.0001]
2. names: List[str] = ["Vanes", "Alen", "Jana", "William", "Richards", "Joy"]
3. numbers: List[int] = [22, 33, 10, 6894, 11, 2, 1]

Напишите код, который создаёт три новых списка. Вот их содержимое:

1. Каждое число из списка floats возводится в третью степень и округляется до трёх знаков после запятой.
2. Из списка names берутся только имена минимум из пяти букв.
3. Из списка numbers берётся произведение всех чисел.

**Подсказка № 1**

Используйте функцию `map()` для применения одной и той же операции ко всем элементам списка. Для возведения числа в третью степень и округления его до трех знаков после запятой используйте лямбда-функцию внутри `map()`.

**Подсказка № 2**

Функция `filter()` позволяет отфильтровать элементы списка на основе заданного условия. В данном случае используйте её для выбора имен, состоящих из пяти и более букв. Убедитесь, что лямбда-функция правильно проверяет длину строки.

**Подсказка № 3**

Для нахождения произведения всех чисел в списке используйте функцию `reduce()`. Эта функция последовательно применяет операцию (в данном случае умножение) ко всем элементам списка. Убедитесь, что вы импортировали `reduce` из модуля `functools`.

**Эталонное решение:**

```
from functools import reduce



# Исходные списки

floats = [12.3554, 4.02, 5.777, 2.12, 3.13, 4.44, 11.0001]
```

```
names = ["Vanes", "Alen", "Jana", "William", "Richards", "Joy"]

numbers = [22, 33, 10, 6894, 11, 2, 1]



# Применяем функцию map для возведения в третью степень и округления
до трех знаков после запятой

map_result = list(map(lambda x: round(x ** 3, 3), floats))



# Применяем функцию filter для выбора имен из пяти и более букв

filter_result = list(filter(lambda name: len(name) >= 5, names))



# Применяем функцию reduce для нахождения произведения всех чисел в
списке

reduce_result = reduce(lambda num1, num2: num1 * num2, numbers)



# Вывод результатов

print(map_result)

print(filter_result)

print(reduce_result)
```

### Задача 2. Zip

Даны список букв (letters) и список цифр (numbers). Каждый список состоит из N элементов. Создайте кортежи из пар элементов списков и запишите их в список results. Не используйте функцию zip. Решите задачу в одну строку (не считая print(results)).

Примеры списков:
letters: List[str] = ['a', 'b', 'c', 'd', 'e']
numbers: List[int] = [1, 2, 3, 4, 5, 6, 7, 8]

Результат работы программы:
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]

**Подсказка № 1**

Перед созданием кортежей из пар элементов убедитесь, что оба списка имеют одинаковую длину. Если это не так, то результат может быть непредсказуемым. Также проверьте типы данных в списках. Неправильная длина списков может привести к недопониманию в том, как будут сформированы кортежи, и ошибка будет в исходных данных.

**Подсказка № 2**

Используйте функцию `map` вместе с лямбда-функцией, чтобы создать список кортежей. В лямбда-функции укажите, что каждый элемент из первого списка должен быть объединен с элементом из второго списка.

**Подсказка № 3**

Внутри лямбда-функции используйте параметры `x` и `y` для представления элементов из первого и второго списков соответственно. Создайте кортеж `(x, y)`.

**Подсказка № 4**

Приведите результат работы `map` к списку с помощью функции `list()`. Это преобразует объект `map` в список, содержащий кортежи.

**Эталонное решение:**

```
from typing import List, Tuple



# Исходные списки

strings = ['a', 'b', 'c', 'd', 'e']

numbers = [1, 2, 3, 4, 5, 6, 7, 8]



# Создание списка кортежей, состоящих из пар элементов из обоих
списков

results: List[Tuple[str, int]] = list(map(lambda x, y: (x, y),
strings, numbers))



# Вывод результатов

print(results)
```

### Задача 3. Палиндром

Используя модуль collections, реализуйте функцию can_be_poly, которая принимает на вход строку и проверяет, можно ли получить из неё палиндром.

**Пример кода:**

```
print(can_be_poly('abcba'))
```

```
print(can_be_poly('abbbc'))
```

Результат:
True
False

**Подсказка № 1**

Перед реализацией функции `can_be_poly`, импортируйте модуль `Counter` из пакета `collections`. Он понадобится для подсчета частоты вхождения каждого символа в строку. `Counter` поможет создать частотный словарь, который упрощает подсчет и анализ символов в строке.

**Подсказка № 2**

Для проверки, можно ли получить палиндром, вычислите количество символов с нечетной частотой. Используйте функцию `filter` и лямбда-функцию для фильтрации значений частот, оставляя только те, которые имеют нечетное количество вхождений.

**Подсказка № 3**

Подсчитайте количество элементов в отфильтрованном списке и проверьте, что их количество меньше 2. Это условие указывает на то, что строка может быть переставлена в палиндром.

**Эталонное решение:**

```
from collections import Counter


def can_be_poly(val: str) -> bool:
    # Создаем счетчик частот символов в строке
    char_counts = Counter(val)

    # Проверяем количество символов с нечетным количеством вхождений
```

```
    odd_count = len(list(filter(lambda x: x % 2,
char_counts.values())))



    # Условие для проверки возможности формирования палиндрома

    return odd_count < 2



print(can_be_poly('eerru'))  # Ожидаемый результат: True

print(can_be_poly('abbcba'))  # Ожидаемый результат: True

print(can_be_poly('abbbc'))  # Ожидаемый результат: False
```

### Задача 4. Уникальный шифр

Напишите функцию, которая принимает строку и возвращает количество уникальных символов в строке. Используйте для выполнения задачи lambda-функции и map и/или filter.

Сделайте так, чтобы алгоритм НЕ был регистрозависим: буквы разного регистра должны считаться одинаковыми.

Пример:

```
message = "Today is a beautiful day! The sun is shining and the birds are singing."

unique_count = count_unique_characters(message)

print("Количество уникальных символов в строке:", unique_count)
```

Вывод: количество уникальных символов в строке — 5.

#### Подсказка № 1

Перед началом работы убедитесь, что строка преобразована в нижний регистр. Это нужно для того, чтобы подсчет символов был регистронезависимым. Преобразование строки к нижнему регистру помогает избежать различий между символами разного регистра, так как `a` и `A` будут считаться одним и тем же символом.

#### Подсказка № 2

Используйте `filter` для выбора уникальных символов. Примените к каждому символу лямбда-функцию, которая проверяет, встречается ли символ в строке ровно один раз. Функция `filter` отфильтрует символы, у которых количество вхождений в строке ровно 1, благодаря проверке внутри лямбда-функции.

**Подсказка № 3**

Для проверки количества вхождений символа используйте метод `count`. Он возвращает количество вхождений подстроки в строке. Метод `count` позволяет узнать, сколько раз символ встречается в строке, что необходимо для проверки уникальности символов.

**Подсказка № 4**

Преобразуйте результат фильтрации в список с помощью `list()`. Это сделает обработку отфильтрованных символов удобнее. Преобразование результата `filter` в список позволяет работать с ним как с обычным списком и получить доступ к его длине.

**Подсказка № 5**

Возвращайте длину списка уникальных символов с помощью `len()`. Это даст количество уникальных символов в строке.

**Эталонное решение:**

```
def count_unique_characters(string):

    # Приводим строку к нижнему регистру, чтобы сделать подсчет регистронезависимым

    lower_string = string.lower()



    # Используем filter для выбора символов, которые встречаются в строке ровно один раз

    unique_chars = list(filter(lambda c: lower_string.count(c.lower()) == 1, lower_string))



    # Выводим уникальные символы (по желанию, можно удалить эту строку)

    print(unique_chars)
```

```
    # Возвращаем количество уникальных символов
    return len(unique_chars)

# Пример использования:
message = "Today is a beautiful day! The sun is shining and the
birds are singing."

unique_count = count_unique_characters(message)
print("Количество уникальных символов в строке:", unique_count)
```