# Практическое задание 2.3.2

**Задание 1**

Какое слово написано? ннооросссооообьбтп

Напишите программный код для подсчёта количества вхождений каждой буквы в слово и постройте гистограмму числа вхождений букв. Сделайте подписи к рисунку и осям. В подпись к рисунку должно входить определенное Вами слово.

## Комментарии и подсказка

1). Для подсчета количества букв рекомендую класс `Counter` модуля `collections` в Python.

Если нужно что-то посчитать, определить количество вхождений или наиболее (наименее) часто встречающихся элементов, используйте объекты класса `Counter`. Создаются они с помощью конструктора `collections.Counter()`.

Синтаксис: `collections.Counter([iterable-or-mapping])`, параметр `iterable-or-mapping` – итерируемая последовательность или словарь.

Функция принимает итерируемый аргумент и возвращает словарь, в котором ключами служат индивидуальные элементы, а значениями – количества повторений элемента в переданной последовательности.

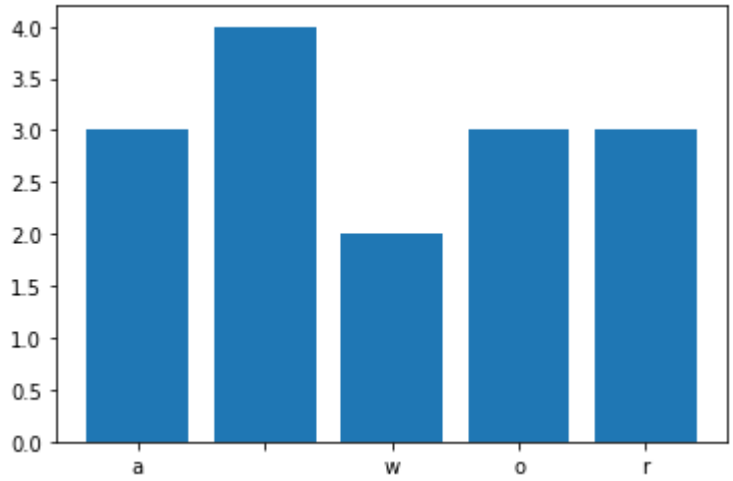
Программный код Python:

```
import collections
s = 'a word and another word'
c = collections.Counter(s)
```

```
c
Результат: Counter({' ': 4, 'a': 3, 'd': 3, 'o': 3, 'r': 3, 'n':
2, 'w': 2, 'e': 1, 'h': 1, 't': 1})
```

2) Вспомним построение гистограммы

```
import matplotlib.pyplot as plt
x = list(c.keys())[:5]
y = list(c.values())[:5]
plt.bar(x, y)
plt.show()
```




[: 5] указывает на то, чтобы выведено было первых 5 столбиков гистограммы.

## Задание 2

Разработайте приложение для поиска анаграмм в тексте. На входе у приложения – текстовый файл, на выходе – списки анаграмм.

**Анагра́мма** (от греч. ανα- «пере» + γράμμα «буква») – литературный приём, состоящий в перестановке букв или звуков определённого слова (или словосочетания), что в результате даёт другое слово или словосочетание.

**Входной текст:**

```
Аз есмь строка, живу я, мерой остр.
За семь морей ростка я вижу рост.
Я в мире – сирота.
Я в Риме – Ариост.
```

**Вывод** (каждая строка содержит анаграммы):

```
['Аз', 'За']
['есмь', 'семь']
['строка', 'ростка']
```

```
['живу', 'вижу']
['мерой', 'морей']
['остр', 'рост']
['мире', 'Риме']
['сирота', 'Ариост']
```

## Комментарии и подсказка

1). Создайте список слов. Для этого удалите пунктуацию в тексте (пробелы, знаки препинания и прочие знаки).

Кому сложно в программе создать список, сделайте это вручную, используя шаблон примера ниже.

2). Удалите стоп-слова.

3). Пример самого простого подхода к поиску анаграмм с помощью циклов и встроенной функции сортировки:

Работаем с английскими словами:

`percussion` – перкуссия, `supersonic` – сверхзвуковой, `car` – машина, `tree` – дерево, `boy` – мальчик, `girl` – девочка, `arc` – дуга.

Программный код Python:

```
word_list = ["percussion", "supersonic", "car", "tree", "boy",
"girl", "arc"]

# initialize a list
anagram_list = []
for word_1 in word_list:
    for word_2 in word_list:
        if word_1 != word_2 and (sorted(word_1)==sorted(word_2)):
            anagram_list.append(word_1)
print(anagram_list)
```

Результат: `['percussion', 'supersonic', 'car', 'arc']`

Если вы посмотрите на внутренний цикл `for`, выделенный желтым, код `word_1 != word_2` проверяет, не совпадают ли слова.

Встроенная функция `sorted` преобразует каждое слово из строки в список символов, который вы можете увидеть в примере ниже.

Поскольку `sorted('percussion') == sorted('supersonic')` – True, они являются анаграммами друг друга.

```
print(sorted('percussion'))
```
Результат: `['c', 'e', 'i', 'n', 'o', 'p', 'r', 's', 's', 'u']`
```
print(sorted('supersonic'))
```
Результат: `['c', 'e', 'i', 'n', 'o', 'p', 'r', 's', 's', 'u']`

## Задание 3

Разработайте приложение, которое принимает на вход текстовый файл (или просто текст) с обычным текстом и выводит этот же текст, но с перемешанными буквами внутри слов. Первая и последняя буквы каждого слова должны остаться на своих местах.

Обычный текст:

По результатам исследований одного английского университета, не имеет значения, в каком порядке расположены буквы в слове. Главное, чтобы первая и последняя буквы были на месте. Остальные буквы могут следовать в полном беспорядке, все равно текст читается без проблем. Причиной этого является то, что мы не читаем каждую букву по отдельности, а все слово целиком.

Примерный результат: По рзелульаттам илссеовадний одонго анлигйсокго унвиертисета, не иеемт занчнеия, в кокам пряокде рсапожолены бкувы в солве. Галвоне, чотбы преавя и пслоендяя бквуы блыи на мсете. Осатьлыне бкувы мгоут селдовтаь в плоонм бсепордяке, все-рвано ткест чтаитсея без побрелм. Пичрионй эгото ялвятеся то, что мы не чиатем кдаужю бкуву по отдльенотси, а все солво цликеом.

### Комментарии и подсказка

1). Кому сложно выполнить это задание и написать полноценный программный код, возьмите следующее предложение:

Обычный текст:

Мы не читаем каждую букву по отдельности, а все слово целиком.

Используйте метод `re.sub(pattern, repl, string)`

Создавайте вручную шаблоны (`pattern`) для слов и заменяйте их на указанное сочетание (`repl`).

**2). Для продвинутых слушателей:**

План:

1. Разбить предложения на слова.
2. Выделить первую и последнюю букву в каждом слове.
3. Перемешать случайным образом все, что осталось.

Так как символы перемешивать необходимо случайным образом – подключите библиотеку `random`. Кроме этого, необходимо указать тип данных обработанного предложения – список:

4. Итак, Вы имеете список, состоящий из всех слов в предложениях. Теперь с помощью цикла пройдите по каждому слову и обработайте его согласно требованиям: первая и последняя буквы не должны изменяться; буквы внутри перемешиваются.

### Случайное перемешивание массива в Python

Для случайного перемешивания массива в английском языке используется термин *shuffle*, что по-русски означает *"перетасовать"*.

Задача случайного перемешивания не так уж и тривиальна, так как для этого требуется выбрать случайную перестановку.

В библиотеке `random` реализована функция, которая перемешивает массив на месте:

```
random.shaffle(A)
```

Другой вариант перемешивания (использование случайного ключа для сортировки):

```
B = sorted(yourList, key=lambda A: random.random())
```

Следует понимать, что даже для сравнительно небольших длин массива `len(A)`, общее число перестановок A больше, чем период наилучших генераторов случайных чисел; это означает, что большинство перестановок длинной последовательности никогда не будут созданы.

И, на всякий случай, примеры.

Пример 1

```
import random
slovo = 'Привет'
slovo_list = list(slovo)
abrakadabra = random.sample(slovo_list,  len(slovo_list))

print(abrakadabra)
```

Результат: `['е', 'т', 'р', 'и', 'в', 'П']`

Пример 2

```
import random
text = "some text"
words = text.split()
for i, word in enumerate(map(list, words)):
    random.shuffle(word)
    words[i] = ''.join(word)

print(*words)  # -> eosm txet
```

Результат: `esom tetx`