Чтобы наглядно посмотреть, реально ли множества выигрывают в скорости, изучили быстрым пробегом по туториалу модуль timeit, который подключает свой таймер для расчёта времени команд

Первая задачка по нему: Написать программу — аналог команды python3 -m timeit, которая с помощью .Timer.autorange() определяет быстродействие введённого сниппета за разумное время.

```
import timeit

def my_timeit(command):
    return timeit.Timer(command).autorange()

print(my_timeit(input()))
```

-- **Задачка на Теоретико-множественные операции:** Посчитать, сколько уникальных гласных и сколько уникальных согласных в строке; считать только строчные латинские буквы. Для строки "aabABbcD123" результат будет: гласных - одна ("a"), согласных - две ("b", "c")

```
from string import ascii_lowercase # Это просто строчка ‘abcdefg..xyz’ вот так красиво её можно импортировать

wovel = set('eyuioa')
consonant = set(ascii_lowercase) - wovel

string = input()

print(len(set(string) & wovel), len(set(string) & consonant))
 # Просто из строки делаем множество, а затем смотрим, где оно пересекается с гласными и согласными и считаем длину
```

-- **Та же прога, но теперь в виде функции, чтобы посчитать её быстродействие**

```
from string import ascii_lowercase
import timeit
wovel = set('eyuioa')
consonant = set(ascii_lowercase) - wovel

def letters(string):
    return (len(set(string) & wovel), len(set(string) & consonant))

cyc, time = timeit.Timer('letters("qwertyuiop")', globals =
globals()).autorange()
print(cyc * 1 / time) # В таком формате выводится количество срабатываний функции в секунду
```

-- **Задачка на словарик, на количество каждого элемента:** Посчитать количество вхождений каждого уникального слова в тексте (слова разделены пробелами) с помощью dict

```
text = input().split()
d = {}

for word in text:
    d[word] = d.setdefault(word, 0) + 1

print(d)
```

-- Основная прелесть что лекции (там я забыл упомянуть), что семинара - модуль ***collections***, который открывает нам доступ к прекрасным хранилищам

<u>На лекции мы обсудили:</u>

- ***deque*** - Дека, или Двусторонняя очередь, отличается от простого стека \ динамического массива константной сложностью удаления и добавления элементов что в конец деки, что в начало (У списка, н-р, удаление в начале линейное. Ни хухры-мухры ускоряемся)

```
>>> from collections import deque
>>> dir(deque)
['__add__', '__class__', '__class_getitem__', '__contains__', '__copy__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'appendleft',
'clear', 'copy', 'count', 'extend', 'extendleft', 'index', 'insert', 'maxlen', 'pop',
'popleft', 'remove', 'reverse', 'rotate']
>>> a = deque()
>>> a.append(1)
>>> a.append(2)
>>> a.appendleft(3)
>>> a
deque([3, 1, 2])
>>> a.popleft()
3
>>> a.popleft()
1
>>> a
deque([2])
```

- ***Counter*** - Это подвид словариков, который сам сразу считает количество каждого значения в себе, собственно, его значением по ключу является количество встреченных ключей. С помощью метода ***most_common()*** можно создать список отсортированных по убыванию повторяемости всех пар значений.

```
>>> from collections import Counter
>>> dir(Counter)
['__add__', '__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__',
'__delitem__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__',
'__iand__', '__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__', '__le__',
'__len__', '__lt__', '__missing__', '__module__', '__ne__', '__neg__', '__new__', '__or__',
'__pos__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__ror__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__weakref__', '_keep_positive', 'clear', 'copy', 'elements', 'fromkeys', 'get', 'items',
'keys', 'most_common', 'pop', 'popitem', 'setdefault', 'subtract', 'total', 'update',
'values']

>>> b = Counter("a b c d c b d e f a a a".split())
>>> b
Counter({'a': 4, 'b': 2, 'c': 2, 'd': 2, 'e': 1, 'f': 1})
>>> b |= Counter("f f f f".split())
>>> b
Counter({'f': 5, 'a': 4, 'b': 2, 'c': 2, 'd': 2, 'e': 1})
>>> b.most_common()
[('f', 5), ('a', 4), ('b', 2), ('c', 2), ('d', 2), ('e', 1)]
```

- ***defaultdict*** и ***namedtuple*** мы тоже обсуждали, но, честно, боюсь соврать сейчас. Потом подразберусь и напишу, что они есть. Я как бы понимаю суть, но пока что хз, как её описать. Если кратко, то ***namedtuple*** даёт имена значениям в кортеже, под которыми можно обращаться к этим элементам кортежа, а ***defaultdict*** задаёт дефолтное значение, которое отрабатывает в необходимых случаях
Про всех них можно почитать в мануальчике, ссылки есть на Вики-лекции

Задачка непосредственно на Counter: Предыдущий подсчёт слов сделать с его помощью (двустрочник буквально)

```
from collections import Counter
print(Counter(input().split()))
```

-- **Снова реализация того же самого в виде функций, чтобы сравнить быстродействие**

```
from collections import Counter
import timeit

def count_dict(s):
    d = {}
    for word in s:
        d[word] = d.setdefault(word, 0) + 1
    return d

def count_Counter(s):
    return Counter(s)

s = input().split()
print(len(s))

cyc, time = timeit.Timer(f'count_dict({s})', globals = globals()).autorange()
print("dict: ", cyc * 1 / time)

cyc, time = timeit.Timer(f'count_Counter({s})', globals = globals()).autorange()
print('Counter: ', cyc * 1 / time)
```

-- **Последняя задачка на прогон знаний о словарях в функциях:** Ввести строку вида выражение с переменными ***x*** и ***y***; затем ввести два числа, ***a*** и ***b***. Вычислить выражение и вывести результат дважды, сначала при ***x=a*** и ***y=b***, а затем при ***x=b*** и ***y=a***, пользуясь пространством имён ***eval()***.

```
exp = input()
a, b = map(int, input().split(","))

print(eval(exp, {'x': a, 'y': b}), eval(exp, {'x': b, 'y': a}), sep = "\n")
```