Урок 2. Проектная деятельность. Работа со списками, кортежами и множествами.

ФИО_____

Тест:

1. *Что из нижнего изменяемо?*

A. tuple
B. str
C. list
D. frozenset

2. *Выберите верные утверждения о set?*

A. Сохраняет порядок
B. Разрешает дубли
C. Операции in в среднем O(1)
D. Можно индексировать

3. *tuple лучше list, когда:*

A. Часто меняем размер
B. Нужна неизменяемость
C. Нужно быстрое добавление в конец
D. Нужны методы изменения

4. *Что выведет данная программа?*

```
set1 = {1, 2, 3}
set2 = {1, 2, 3, 4, 5}
print(set1.issubset(set2))
```

---

5. *Что выведет данная программа?*

```
a = [1, 2, 3]
b = a
a += [4]
print(b is a, b)
```

Теория:

Списки (list)

Список – изменяемая упорядоченная последовательность. Быстрый доступ по индексу, удобно хранить и перебирать элементы.

```
#Создание и копирование
a = []                  # пустой список
b = list(iterable)      # из итерируемого
c = a[:]                # поверхностная копия
d = a.copy()            # поверхностная копия (то же самое)
#Индексация и срезы
x = a[i]                # чтение за O(1)
a[i] = val              # запись за O(1)
sub = a[i:j:k]          # срез → НОВЫЙ список (O(длина среза))
a[i:j] = [1,2,3]        # присваивание срезу меняет исходный список
```

Основные методы списков

- append(x) — в конец (амортизированно O(1))
- extend(iterable) — добавить много (O(k))
- insert(i, x) — вставка в середину (O(n))
- pop() / pop(i) — снять с конца O(1) / из середины O(n)
- remove(x) — удалить первое вхождение (O(n)); clear()
- index(x[, start[, stop]]), count(x)
- reverse() — развернуть in-place
- sort(key=None, reverse=False) — сортировка **на месте**
- sorted(iterable, key=..., reverse=...) — вернуть новый список

Итерация:

```
for i, v in enumerate(a, start=0): ...
for x, y in zip(xs, ys): ...
for x in reversed(a): ...
sq = [x*x for x in a if x%2==0]
```

Кортежи (tuple)

Кортежи – это неизменяемая упорядоченная последовательность. Можно использовать как ключ словаря/элемент множества.

```
t = (1, 2, 3)       # или t = 1, 2, 3  (упаковка)
single = (1,)       # одиночный кортеж — запятая обязательна
a, b, *mid, c = t   # звёздочная распаковка
```

Когда выбирать tuple:

- Набор полей фиксирован и не должен меняться.
- Нужно хранить в set/использовать как ключ dict.
- Чуть компактнее по памяти, чем список, и защищает от случайных изменений.

Под капотом кортежи — просто контейнер ссылок; «неизменяемость» — про сам контейнер, не про вложенные объекты.

Множества (set, frozenset)

Множества - это неупорядоченный набор **уникальных** хешируемых элементов. Быстрая проверка принадлежности.

```
# Создание
s = set()               # пустое множество ({} — это пустой dict!)
s = {1, 2, 3}
```

```
fs = frozenset([1, 2]) # неизменяемое множество (можно класть в set/dict)

# Проверка принадлежности и добавление
2 in s          # O(1) в среднем
s.add(5)

# Операции — создают НОВОЕ множество
# объединение (union)
A | B
A.union(B, C)
# пересечение (A.intersection(B))
A & B
# разность (A.difference(B)) элементы A, которых нет в B
A - B
# симметричная разность(A.symmetric_difference(B)), элементы, которые есть
ровно в одном из двух множеств.
A ^ B

# Важно: метод выше принимает ОДИН аргумент (в отличие от
union/intersection/difference).


# Операции in-place (меняют текущее множество)
A.update(B, C)                    # A |= B | C
A.intersection_update(B, C)       # A &= B & C
A.difference_update(B, C)         # A -= B | C
A.symmetric_difference_update(B)  # A ^= B

# Отношения между множествами
A.issubset(B)     # A ⊆ B (проверка подмножества)
A.issuperset(B)   # A ⊇ B (проверка надмножества)
A.isdisjoint(B)   # A ∩ B = ∅ (проверка отсутствия пересечений)

A <= B; A < B     # подмножество / строгое подмножество
A >= B; A > B     # надмножество / строгое надмножество

# Методы работы с элементами
s.add(x)
s.discard(x)      # не бросает ошибку, если x нет
s.remove(x)       # KeyError, если x нет
s.pop()           # удалить и вернуть ПРОИЗВОЛЬНЫЙ элемент
s.clear()
s.copy()
```

Задачи:

1. **Скользящее среднее:** Напишите moving_avg(a, k) → список средних по каждому окну длины k (целое $k \geq 1$).
   Пример: a=[1,2,3,4], k=2 → [1.5, 2.5, 3.5].
2. **Run-Length Encoding (RLE):** По списку/строке вернуть список кортежей (значение, счётчик) для подряд идущих одинаковых элементов.
   Пример: [1,1,1,2,2,3] → [(1,3),(2,2),(3,1)], "aaabb" → [('a',3),('b',2)].
3. **Слова: операции множеств**
   Даны две строки. Привести к нижнему регистру, разбить по пробелам.
   а) вывести пересечение слов (в алфавитном порядке),
   б) вывести разность S1 - S2,
   в) вывести симметрическую разность.
   Пример: "a b c" и "b d" → пересечение ['b'], разность ['a','c'], симм.разн. ['a','c','d'].

4. **Отсечение экстремумов:** Дан список чисел. Удалите **ровно по одному** вхождению минимального и максимального значений, сохранив порядок остальных. Если все элементы равны — вернуть пустой список.
   Пример: [5,1,3,5,2,1] → удалить один 1 и один 5 → [3,5,2,1] (удаляются первые встретившиеся min и max).
5. **Сколько разных k-подотрезков:** По списку a и числу k посчитать количество **различных** подотрезков длины k (используйте кортежи для хеширования).
   Пример: a=[1,2,1,2,3], k=3 → подотрезки {(1,2,1),(2,1,2),(1,2,3)} → 3.
6. **Топ-K по частоте:** Функция top_k(items, k) возвращает k самых частых элементов. При равной частоте сортируйте лексикографически.
   Пример: ['a','b','a','c','b','a'], k=2 → ['a','b'].

7. По списку целых чисел определите, **можно ли переставить элементы так, чтобы никакие два соседних не были равны**. Верните True/False. Подсказка: это возможно тогда и только тогда, когда максимальная частота любого значения не превышает ceil(n/2).
   Пример: [1,1,2,2,3] → True (например, 1,2,1,2,3), [1,1,1,2] → False. Подсказка: подумайте о сортировке и разбиении на две группы.

8. **Поворот матрицы на 90° по часовой:** Функция rotate90(mat) для квадратной матрицы (список списков) возвращает новую матрицу, повернутую на 90° по часовой стрелке.
   Пример: [[1,2],[3,4]] → [[3,1],[4,2]].