# Practice Session 6

## Зміст



## 📚 Вступ до списків:

На сьогоднішній практиці ми попрактикуємося працювати зі списками – однією з найбільш фундаментальних структур даних у мові програмування Python. Списки - гнучкий інструмент, який дозволяє нам проводити різноманітні маніпуляції з даними, що є однією з основних задач програмування.

## 📌 План заняття:

- 💪🏻 Warm Up
- 🧩 Алгоритмічна класика
- 📈 Списки, як аналітичний інструмент

- 📐 Вкладені списки, матриці, та інші речі придумані математиками
- ⚙️ Завдання для самостійного опрацювання

## 📝 Lists Cheat Sheet

Тут зібрані основні операції, які знадобляться вам для виконання завдань.

### 🔹 1. Створення та доступ

| Операція | Синтаксис | Опис |
|---|---|---|
| **Створення** | `lst = [1, 2, 3]` | Створити список із даними |
| **Пустий список** | `lst = []` | Ініціалізація пустим списком |
| **Перший елемент** | `lst[0]` | Індексація починається з **0** |
| **Останній елемент** | `lst[-1]` | Негативний індекс рахує з кінця |
| **Довжина** | `len(lst)` | Кількість елементів у списку |

### 🔹 2. Зрізи (Slicing)

Загальний синтаксис: `list[start:stop:step]`

```
arr = [10, 20, 30, 40, 50]

arr[1:4]   # [20, 30, 40] - від індексу 1 до 3 включно
arr[:3]    # [10, 20, 30] - перші 3 елементи
arr[::2]   # [10, 30, 50] - кожен другий елемент
arr[::-1]  # [50, 40, 30, 20, 10] - розворот списку
```

### 🔹 3. Додавання та зміна

```
nums = [1, 2, 3]

nums.append(4)    # [1, 2, 3, 4] -> Додає елемент в кінець (O(1))
nums[0] = 100     # [100, 2, 3, 4] -> Змінює існуючий елемент
```

### 🔹 4. Ітерація (Цикли).

Два основні способи пробігтися по списку: - Коли потрібен тільки елемент

```
for item in nums:
    print(item)
```

- Коли потрібен індекс (наприклад, для формул $y_{i+1} - y_i$):

```python
for i in range(len(nums)):
    print(f"Index: {i}, Value: {nums[i]}")
```

- Коли потрібно і те, і інше (Pythonic way):

```python
for i, val in enumerate(nums):
    print(f"Index: {i}, Value: {val}")
```

## 💪 Warm Up

Перед тим як перейти до серйозніших задач, давайте пригадаємо основи роботи зі списками на більш тривіальних завданнях.

**Задача 1**

**Умова:** Вам надано список, який складається із чисел. Виведіть на екран всі невід'ємні числа, їх суму та середнє арифметичне.

```
Input: arr = [-12, 8, -7, 6, 12, -9, 14]
Output: avg = 10.0, sum = 40
```

**Задача 2**

**Умова:** Вам надано список, який складається із чисел. Створіть два нових списки з тільки парними та тільки непарними числами

```
Input: arr = [54, 43, 2, 5, 14, 17, 18, 9]
Output: even: 54 2 14 18, odd: 43 5 17 19
```

**Задача 3**

**Умова:** Вам надано список, який складається із чисел. Виведіть на екран усі унікальні значення, присутні в списку

```
Input: arr = [1, 2, 2, 3, 4, 5, 5]
Output: [1, 2, 3, 4, 5]
```

> ℹ Примітка
>
> Завдання вищі були взяті з GeeksForGeeks

## 🧩 Алгоритмічна класика

У цьому блоці ми зосередимося не на вивченні синтаксису мови, а на алгоритмічному мисленні. Більшість складних обчислювальних задач — від сортування великих даних

до моделювання фізичних процесів — базуються на вмінні ефективно маніпулювати елементами за їх індексами.

Одним із класичних типів задач, які постають перед нами – це задачі на два вказівники. Використання тактики “двох вказівників” дозволяє швидко та ефективно розв’язати деякі з проблем.

### 👨‍💻 Live Coding 1: Два вказівники

**Контекст:** Ми працюємо над розробкою бази даних. У нас є список заповнений певними значеннями, але от халепа! Виявилось, що значення в ньому знаходяться в оберненому порядку. Нам потрібно “перевернути” значення в списку, при тому не створюючи нового списку.

**Вхідні дані:** список `arr = [50, 40, 30, 20, 10]`

**Вихідні дані:** обернений список `arr = [10, 20, 30, 40, 50]`

> ⚠️ Обмеження
>
> Не можна використовувати list slicing, оскільки це створить новий список. Маємо користуватись тільки циклами та модифікацією поточного списку

### 🛠️ Guided Practice: Списки-паліндроми

**Контекст:** У прикладній математиці та обробці сигналів часто виникає потреба перевірити функцію на парність або симетрію відносно центру. У програмуванні масив, який читається однаково зліва направо та справа наліво, називають **паліндромом**. Ваше завдання — визначити, чи є даний список симетричним.

**Математична модель:** Вектор $A$ є симетричним, якщо для всіх $i$ виконується умова:

$$A[i] == A[n-1-i]$$

**Вхідні дані:**

- Варіант 1: `arr = [1, 2, 3, 2, 1]`
- Варіант 2: `arr = [1, 2, 3, 4, 5]`

**Вихідні дані:**

- Варіант 1: `True`
- Варіант 2: `False`

## 📈 Списки, як аналітичний інструмент

### 👨‍💻 Live Coding 2: Ковзне середнє

**Контекст:** Ми працюємо з даними температурного датчика. Датчик дешевий, тому іноді він видає різкі стрибки. Якщо побудувати графік, він буде різким та не відображатиме справжніх тенденцій. Нам потрібно згладити цей ряд, щоб побачити справжній тренд.

Для цього використовують фільтр ковзного середнього. Суть полягає в підміні групи точок (вікно) на їх середнє арифметичне.

$y_i = \frac{1}{k} \sum_{j=0}^{k-1} x_{i+j}$.

**Вхідні дані:** `noisy_data = [10, 12, 30, 14, 11, 13, 12, 18]`, Розмір вікна $k = 3$.

**Вихідні дані:** `[17.3, 18.7, 18.3, 12.7, 12.0, 14.3]`

### 🛠️ Guided Practice:

**Контекст:** Сучасні фітнес-браслети та смартфони рахують кроки за допомогою акселерометра. Цей датчик вимірює прискорення (G-force).Коли ви стоїте, датчик показує $\approx 1.0G$ (сила тяжіння).Коли ви робите крок (удар ногою об землю), виникає різкий сплеск перевантаження - пік.Ваша задача — написати ядро алгоритму для підрахунку кроків.

Логіка імплементації:

1. Ми вважаємо, що крок відбувся в момент часу $i$, якщо: Значення сигналу є локальним максимумом (більше за сусідів):
$$a_{i-1} < a_i > a_{i+1}$$
2. Сила удару має бути достатньо великою, щоб це не було випадковим шумом. Встановимо поріг $1.5G$:
$$a_i > 1.5$$
**Вхідні дані:** `g_forces = [0.98, 1.02, 1.85, 1.20, 0.99, 1.01, 2.10, 1.60, 0.98]`

**Вихідні дані:** кількість кроків та список зі значеннями піків `peaks = [1.85, 2.10]`

> ⚠️ Обережно з індексами!
>
> Продумайте, як саме запустити цикл, аби не вилізти за межі списку.

## 📐 Вкладені списки, матриці, та інші речі придумані математиками

У Pyhon, як і в будь-якій іншій мові програмування, ми можемо представляти математичні одиниці, зокрема матриці, за допомогою різних інструментів. В контексті сьогоднішнього заняття, нам буде зручно зробити це за допомогою вкладених списків:

```
matrix = [
    [1, 2],
    [3, 4],
    [5, 6]
]
```

### 👨‍💻 Live Coding 3: Транспонування

**Контекст:** Уявіть, що ви працюєте з набором даних, представленим вкладеним списком. Зазвичай рядок стовпець - це дані про об'єкт, я рядок це набір різних об'єктів Але для деяких алгоритмів машинного навчання (або для візуалізацій) нам потрібно поміняти їх місцями: щоб рядки стали набором об'єктів, а стовпці — даними про ці об'єкти.

Математично це операція **транспонування** ($A^T$).

**Вхідні дані:** матриця $3 \times 2$: `data = [[1, 2], [3, 4], [5, 6]]`

**Вихідні дані:** матриця $2 \times 3$: `transposed_data = [[1, 3, 5], [2, 4, 6]`

### 🛠️ Guided Practice: множення матриці на вектор

**Контекст:** І насамкінець, головна операція лінійної алгебри – множення. У робототехніці та комп'ютерній графіці постійно потрібно перераховувати координати. Наприклад, рука робота повертається на певний кут. Щоб знати, де опиниться її кінець, ми множимо матрицю повороту на вектор поточних координат.

**Вхідні дані:** `matrix = [[1, 2], [3, 4], [5, 6]]`, `vector = [10, 20]`

**Вихідні дані:** вектор довжини 3.

> 💡 Підказка
>
> Нижче надано скелет коду, який може стати вам у нагоді

```
if len(matrix[0]) != len(vector):
    print("Error")
else:
    result = []
    for i in range(len(matrix)):
        row_sum = 0
        # ToDo: Complete the code below this comment



        # ........................................
        result.append(row_sum)
```

## ⚙️ Завдання для самостійного опрацювання

- E-Olymp 8956
- E-Olymp 8963
- E-Olymp 8967