



Основы работы с данными для ИИ

Лекция 4

Основы Python для работы с данными для ИИ

Доц

Екатерина Александровна

Преподаватель кафедры информатики и вычислительного эксперимента

Зачем Python в данных и ИИ

- Язык общего назначения с простым синтаксисом и большой экосистемой библиотек для вычислений, статистики, визуализации и ML.
- Рабочие уровни: исследовательские ноутбуки (быстрая проверка гипотез) и продакшн-скрипты/сервисы (повторяемость и автоматизация).
- Ключевые библиотеки: NumPy (массивы), pandas (таблицы), Matplotlib (графики), scikit-learn (классический ML), PyTorch/TensorFlow (нейросети).



Объект — единица данных с типом и поведением. В Python всё — объект: числа, строки, списки, функции, модули, даже сам тип. Переменная — это не «коробка с данными», а **имя-ссылка** на объект. Одно и то же имя в разное время может ссылаться на разные объекты; несколько имён могут ссылаться на **один и тот же** объект.

Типы значений: целые и вещественные числа, логические, строки, даты/время (через библиотеки), контейнеры.

Контейнеры:

- список (изменяемая упорядоченная последовательность),
- кортеж (неизменяемая последовательность),
- множество (уникальные элементы, операции теории множеств),
- словарь (отображение «ключ → значение»).

Скалярные (простые) типы

- **int (целые)**: счёт предметов, идентификаторы. Точные, без переполнения (можно очень большие числа).
- **float (вещественные)**: измерения и средние. Быстрые, но с погрешностью ($0.1+0.2 \neq 0.3$ ровно).
- **bool (логические)**: да/нет, условия. Значения True/False.
- **str (строки)**: текст в Юникоде (русский, эмодзи).
- **Неизменяемые** (каждая «правка» даёт новую строку).

Специальные типы

- **None:** «нет значения»/пусто (служебный маркер, не ноль и не пустая строка)
- **bytes / bytearray:**сырой байтовый поток (файлы, сеть, кодировки). **bytes** — неизменяемые, **bytearray** — изменяемые

Даты

- **datetime/date/time:**календарь и время, часовые пояса — для временных рядов.

Контейнеры (собирают много значений)

- `list` (список): упорядоченный, изменяемый, допускает повторы. Когда важен порядок и часто меняете данные.
Напр.: оценки за недели: `[4, 5, 3, 5]`.
- `tuple` (кортеж): упорядоченный, неизменяемый. Удобен для фиксированных «пакетов» (координаты, дата).
Напр.: (год, месяц, день).
- `set` (множество): только уникальные, без порядка. Быстро проверить «входит/не входит».
Напр.: теги: `{"python", "data"}`.
- `dict` (словарь): пары ключ → значение; быстрый доступ «по имени», а не по номеру.
Напр.: `{"name": "Аня", "age": 20}`.

Контейнеры (собирают много значений)

- `list` (список): упорядоченный, изменяемый, допускает повторы. Когда важен порядок и часто меняете данные.
Напр.: оценки за недели: `[4, 5, 3, 5]`.
- `tuple` (кортеж): упорядоченный, неизменяемый. Удобен для фиксированных «пакетов» (координаты, дата).
Напр.: (год, месяц, день).
- `set` (множество): только уникальные, без порядка. Быстро проверить «входит/не входит».
Напр.: теги: `{"python", "data"}`.
- `dict` (словарь): пары ключ → значение; быстрый доступ «по имени», а не по номеру.
Напр.: `{"name": "Аня", "age": 20}`.

```
lst = [1, 2, 3]; lst.append(4)
tup = (10, 20)
st = {1, 2, 2, 3}          # {1, 2, 3}
d = {"age": 20, "city": "Msk"}
sq_even = [x*x for x in lst if x % 2 == 0]
```

Цикл — это конструкция повторения, позволяющая выполнить одно и то же действие для множества значений. В Python основной практический цикл — `for`, который ходит по итерируемому объекту

Итерабельность и итератор

- **Итерируемый объект** (`iterable`) — то, по чему можно пройти поэлементно (последовательности, диапазоны, файлы, словари и т.п.).
- **Итератор** (`iterator`) — сущность, которая умеет возвращать следующий элемент и сигнализировать об окончании.
- `for` скрывает эти детали: он сам получает итератор и «вытягивает» элементы до конца.

Диапазон значений (`range`) — модель счётчика

- `range(n)` задаёт конечную последовательность индексов от 0 до $n-1$.
- Важно: последняя граница не включается — это правило «полуинтервала» [начало, конец).

```
for i in range(3):  
    print(i)  
  
def mean(a, b):  
    return (a + b) / 2  
  
nums = [1,2,3,4]  
squares = [x*x for x in nums]
```

CSV — «comma-separated values»: строки текста, **разделённые** символом (запятая/точка с запятой/таб). Часто первая строка — **заголовки** столбцов.

`csv.DictReader(f)` читает файл **построчно** и на каждой строке выдаёт **словарь**:

ключ = имя колонки из заголовка, значение = строка из ячейки.

Важно: все значения строковые. Типы (число/дата/логика) надо приводить отдельно (иначе «42» остаётся строкой).

Особенности:

Типичные нюансы CSV

- Разделитель может быть не „, а ; или \t → его нужно задавать (`delimiter=';`’).
- Цитирование/экранировка: в ячейках могут быть запятые и переносы строк, поэтому значения обрамляют кавычками. Модуль `csv` сам это обрабатывает.
- Переводы строк в разных ОС (CRLF/LF). Рекомендация из доков: для **записи CSV** открывать файл с `newline=`”, чтобы модуль управлял переводами строк сам.

- with open(...) as f: — **контекстный менеджер**. Он гарантирует, что файл закроется автоматически (даже при ошибках).
- encoding="utf-8" — задаёт **кодировку**: как переводить байты ↔ символы. Для русских букв в 2025 году «правильный по умолчанию» — **UTF-8**.
- **JSON** — текстовый формат для **иерархических** структур: объект (словарь), массив (список), строки, числа, true/false, null
- json.dump(...) пишет **сразу в файл** (в отличие от json.dumps, который возвращает строку).
- ensure_ascii=False — оставляет **Unicode-символы как есть** (а не \u041f...). Нужна правильная кодировка файла (у вас UTF-8).
- indent=2 — «красивые» переносы и отступы → читаемо человеком, больше размер.

Важно: CSV → JSON меняет **модель данных**: из плоских строк в словари/списки. Так проще работать в API и современных библиотеках.

```
import csv, json

# CSV чтение
with open("data.csv", encoding="utf-8") as f:
    rows = list(csv.DictReader(f))

# JSON запись
with open("out.json", "w", encoding="utf-8") as f:
    json.dump(rows[:5], f, ensure_ascii=False,
              indent=2)
```

`ndarray`: базовый объект NumPy

- `ndarray` — N-мерный массив однотипных элементов в непрерывной (обычно) памяти.
- Ключевые атрибуты: `dtype` (тип элемента), `shape` (форма), `ndim` (число осей), `size` (кол-во элементов), `strides` (шаги по памяти).
- Векторные операции работают сразу над целыми массивами → обычно быстрее, чем Python-циклы.

`np.arange` и `reshape`

- `np.arange(start, stop, step)` создаёт равномерную последовательность как в `range` (верхняя граница не включается). С целочисленными шагами даёт целочисленный `dtype` (часто `int64`).
- `reshape(r, c)` меняет представление формы без копии, если возможно. Общее число элементов должно совпадать: $r * c == \text{size}$.

```
import numpy as np
a = np.array([1,2,3],
dtype=np.float32)
M = np.arange(12).reshape(3,4)
# матрица 3x4: 0..11
print(M[0, 1:3])
# срез
print(M.mean(axis=0))
# среднее по столбцам
```

Pandas: датафреймы

Pandas — библиотека для работы с **табличными** и **временными** данными.

Базовые объекты:

- **Series** — один столбец с **индексом** (метками строк).
- **DataFrame** — таблица из нескольких Series, разделяющих общий индекс.
- **Index** — упорядоченный набор меток (может быть числовым, строковым, датами, MultiIndex и т.д.).

Типы данных (dtypes) и «настоящие» NA

- Основные типы столбцов: int/float/bool, **строки** (string — предпочтительнее старого object), **категориальные** (category), даты/время (datetime64[ns]), интервалы (Timedelta/Period).
- В современных версиях есть **nullable** типы: Int64, Boolean, string — они поддерживают **NA** (пустые значения) без деградации типа.
- **NA**: pandas.NA/NaN означают «нет значения»; арифметика и сравнения с NA дают NA (или игнорируют в агрегатах).

```
import pandas as pd

df = pd.read_csv("data.csv")

df = df.drop_duplicates()

df["temp"]

=df["temp"].fillna(df["temp"].median())

print(df.groupby("city")["temp"].mean())
```

Pandas: выбор, пропуски, новые столбцы



```
df["temp_c"] = df["temp_c"].astype(float)
subset = df.loc[df["city"]=="Msk",
["date", "temp_c"]]
df["temp_c"] =
df["temp_c"].fillna(df["temp_c"].mean())
df = df.drop_duplicates()
```

```
g = df.groupby("city")["temp_c"].agg(["count", "mean"])
left = pd.DataFrame({"id": [1, 2],
                    "city": ["Msk", "Spb"]})
right = pd.DataFrame({"id": [1, 2], "pop": [13.1, 5.6]})
joined = left.merge(right, on="id", how="left")

pivot = pd.pivot_table(df, index="city",
                       columns="month", values="temp_c", aggfunc="mean")
```

Временной ряд — последовательность наблюдений, упорядоченная по времени (timestamp → значение).

Типы временных индексов в pandas:

- **DatetimeIndex** — метки времени (момент).
- **PeriodIndex** — период (месяц, квартал...) как целая единица.
- **TimedeltaIndex** — разность/продолжительность времени.

Временной индекс даёт «магические» возможности: ресемплинг, оконные функции по времени, группировки по календарю.

```
df["date"] = pd.to_datetime(df["date"])
df["year"] = df["date"].dt.year
daily =
df.set_index("date").resample("D").mean(numeric_only=True)
```