Foundations of Data Science and AI

Hrachya Asatryan

Motivation

- Data is growing exponentially across all global industries.
- Without analysis tools, this massive data is merely noise.
- Data Science transforms raw information into valuable knowledge.
- It provides insights essential for modern decision-making.
- These foundational skills prepare you for a career in AI and ML.

Course Structure

NumPy: Computational Foundations

The first part of our course will cover NumPy, a fundamental library in Python that powers nearly all scientific computing. We will:

- Master array operations for speed and memory efficiency.
- Gain practical, hands-on skills for large dataset manipulation.
- Apply core vectorization concepts to solve complex problems.
- Establish the essential computational foundation for data science.
- Prepare for advanced modules like Pandas and Machine Learning.

Course Structure

Pandas: From Raw Data to Insights

The second part of our course will cover Pandas, a library in Python that enables working with large datasets and structuring them efficiently. We will:

- Learn essential data structures like Series and DataFrames.
- Master data cleaning, preparation, and manipulation techniques.
- Gain skills to handle missing values and messy datasets efficiently.
- Apply powerful analysis tools to uncover key business insights.
- Build a robust foundation for Exploratory Data Analysis (EDA).

Course Structure

Data Visualization & Exploratory Data Analysis

The third part of our course will cover Data Visualization and EDA. We will explore the Matplotlib, Seaborne and Plotly libraries and learn how to visualize and analyze data to gain insights on information. We will:

- Translate complex datasets into clear, impactful visual stories.
- Master the art of asking and answering data-driven questions.
- Learn to identify patterns, trends, and anomalies in data.
- Gain skills in selecting the right visualization for every insight.
- Develop the critical foundation needed before building any predictive model.

Course Structure

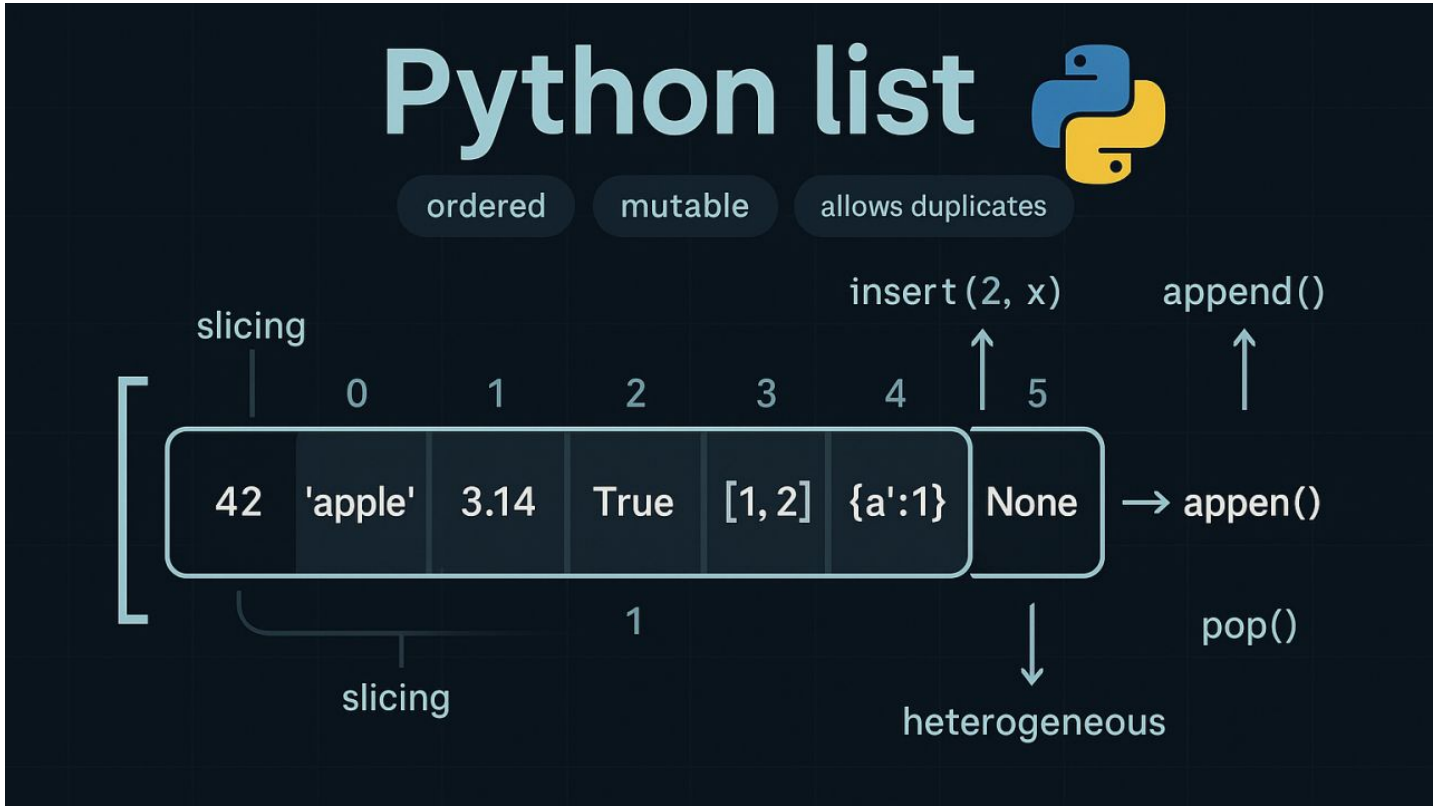
Introduction to AI and its applications

The last part of our course will be related to AI, what it is, where it’s used, etc. We will discuss:

- What is Artificial Intelligence?
- Overview of the AI Landscape and Key Concepts
- Real-World Applications of AI (e.g., Computer Vision, NLP, Robotics)
- Ethical Considerations in AI
- Learning Methods (Supervised / Unsupervised / Reinforcement Learning)

Part 1: NumPy

Flexibility of Python lists



The downside of Python’s Flexibility

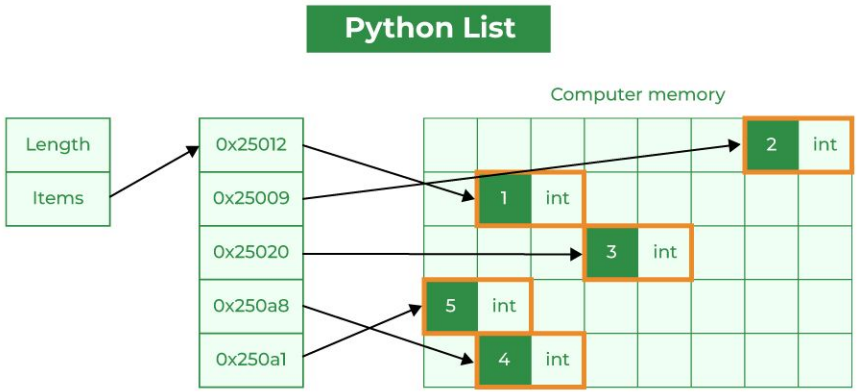
A Python data type is a pointer to a position in memory containing all the object information. This extra information in the Python structure is what allows Python to be coded so dynamically.

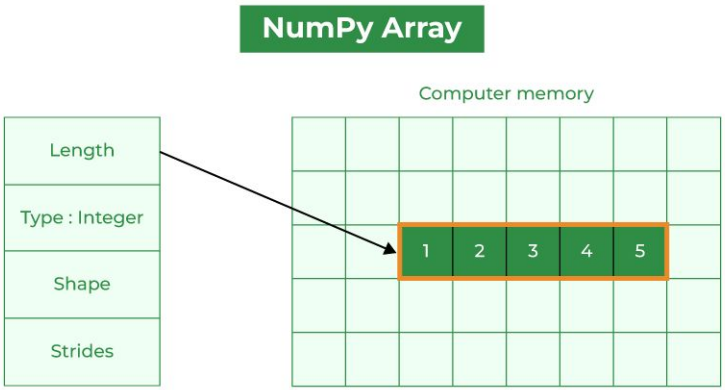
For example, a single integer in Python actually contains 4 pieces:

- `ob_rfcnt`, a reference count that helps Python silently handle memory allocation and deallocation
- `ob_type`, which encodes the type of the variable
- `ob_size`, which specifies the size of the following data members
- `ob_digit`, which contains the actual integer value that we expect the Python variable to represent

Python lists vs NumPy arrays

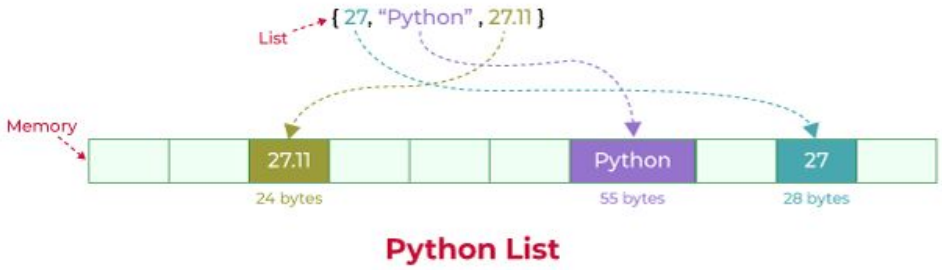
To allow all this flexibility, all this information is stored for each item in a list. When the datatype is fixed, this is often redundant.

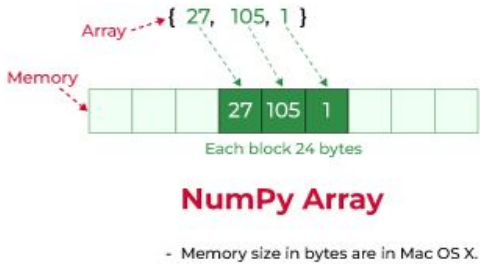




Python lists vs NumPy arrays

The Python list is a pointer to a block of pointers. This allows the flexibility Python is renowned for.
The array is a pointer to a contiguous block of data.
NumPy arrays are thus much less flexible, but much better for fixed-type (e.g. numeric) data

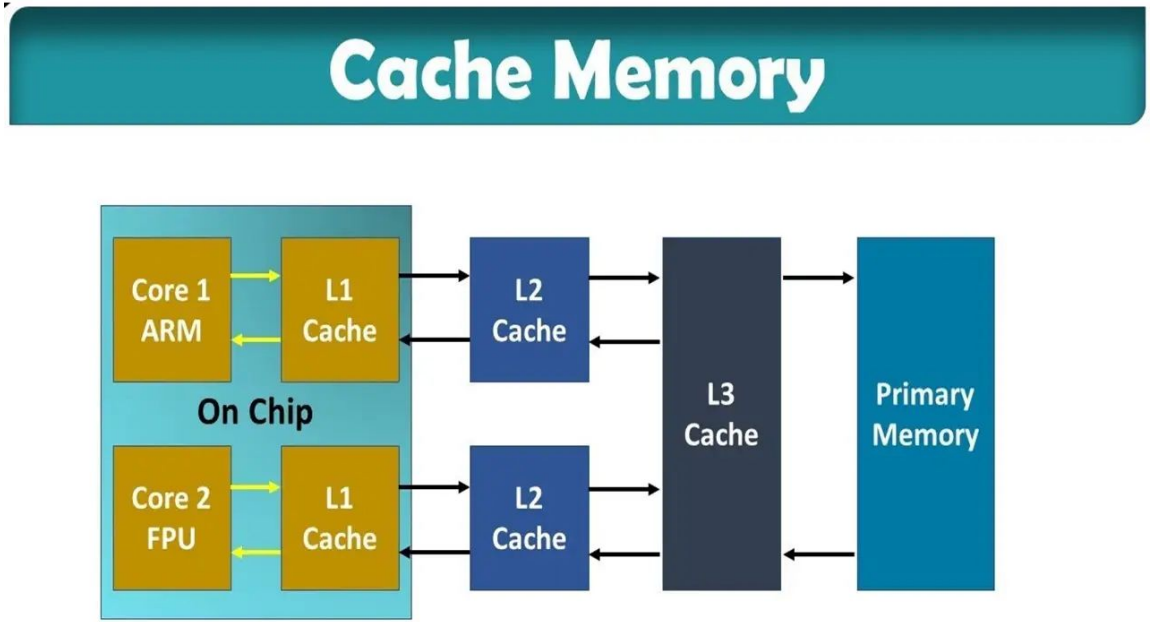




- Memory size in bytes are in Mac OS X.

Data processing

From a CPU's perspective, reading from RAM is slow. Caching is needed to speed things up. Minimizing the number of calls to memory, thus enabling caching is essential for speeding up calculations.



Data types in Python vs NumPy

Native Python

- `bool` 28 Bytes
- `int` 28 Bytes
- `float` 24 Bytes
- `NoneType` 16 Bytes
- `list` 56 Bytes
- `tuple` 48 Bytes
- `dict` 72 Bytes

NumPy

- `bool` 1B
- `int8, int16, int32, int64` 1B, 2B, 4B, 8B
- `uint8, uint16, uint32, uint64` same as int
- `float16, float32, float64` 2B, 4B, 8B

Example: 1 million integers

Python List

- 1 List Object: 56B
- 1 million pointers: 1 million * 8B = 8MB
- 1M integer objects: 1M * 28B = 28MB

- **Total: 56+8M+28M=36MB of memory**

NumPy Array

- No per-element objects
- No per-element pointers
- 1 million * 8B (int64 size) = 8MB

- **Total: 8MB of memory**

Vectorization

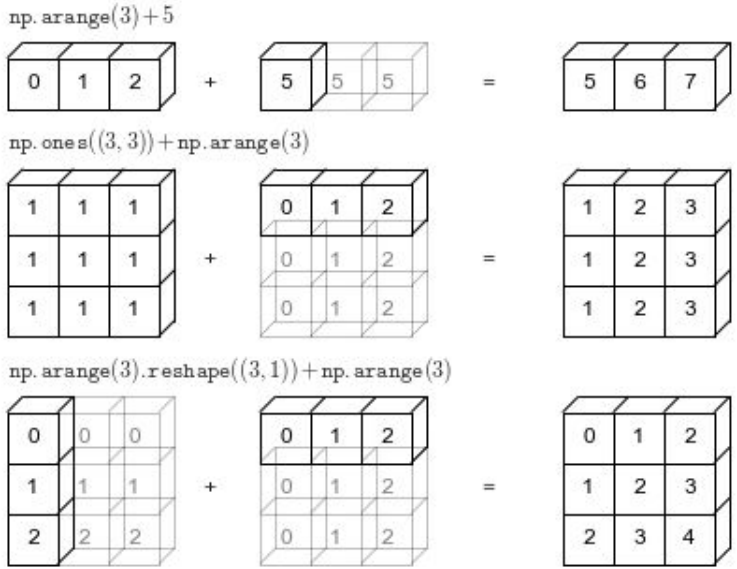
Vectorization in NumPy refers to applying operations on entire arrays at once, rather than using explicit Python loops to process individual elements. This approach leverages highly optimized, pre-compiled C code internally, resulting in significantly faster and more efficient numerical computations, especially for large datasets.

The paradigm shifts from.

For each element, do X ⟶ **Apply X to all elements**

Broadcasting

Broadcasting is a mechanism in NumPy that allows arithmetic operations on arrays of different shapes and sizes without explicitly creating copies of the data to match the larger array's shape.



Broadcasting rules

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing (i.e. **rightmost**) dimension and works its way left. Two dimensions are compatible when

- they are equal, or
- one of them is 1.

If these conditions are not met, a `ValueError: operands could not be broadcast together` exception is thrown, indicating that the arrays have incompatible shapes.

Jupyter Notebook Demo

In this jupyter notebook, we will be demoing numpy arrays, specifically demonstrating how vectorization and broadcasting can help during computations.

Resources

- NumPy Official Documentation
- Jake VanderPlas Python Data Science Handbook