🧩 Course 2 — Data Structures with Python

Week 2: Arrays & Lists

🎯 Learning Objectives

By the end of this week, students will be able to:

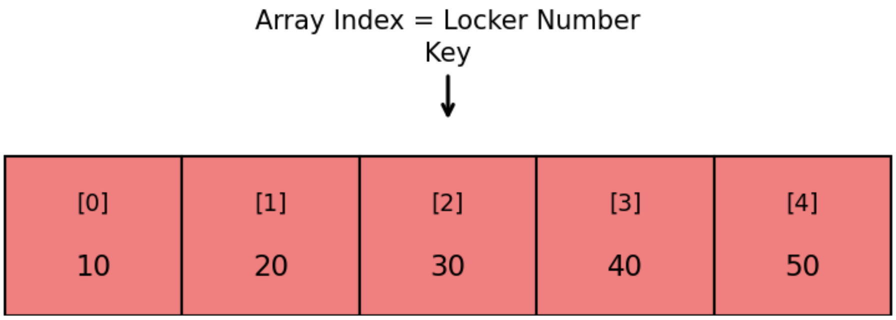
- ✅ Understand the concept of **arrays** and their role as the foundation of data structures.
- ✅ Differentiate between **arrays** and **Python lists** in terms of memory and flexibility.
- ✅ Perform key array operations — creation, traversal, insertion, deletion, and searching.
- ✅ Explain **indexing, slicing, and dynamic resizing** in Python lists.
- ✅ Analyze the **time and space complexity** of list operations.

📘 1. Introduction: Why Arrays Matter

An **array** is a contiguous block of memory storing elements of the **same type**.
It is one of the oldest and most fundamental data structures in computer science.

Real-world analogy:

Think of an **array as a row of lockers**.
Each locker (index) holds a specific value. You can directly access any locker if you know its number — fast and predictable.



Array Locker Analogy

🧠 2. Arrays vs Lists in Python

Python does not have native fixed-size arrays like C, but provides two alternatives:

Feature	Array (via `array` module)	List
Data type	Homogeneous	Heterogeneous
Memory layout	Contiguous	Dynamic, flexible
Performance	Faster for numeric data	More general-purpose
Syntax	`array('i', [1,2,3])`	`[1,2,3]`

Example:

```
from array import array

# Integer array
numbers = array('i', [10, 20, 30])
numbers.append(40)
print(numbers[2])
```

Output:

```
30
```

⚙️ 3. Basic List Operations

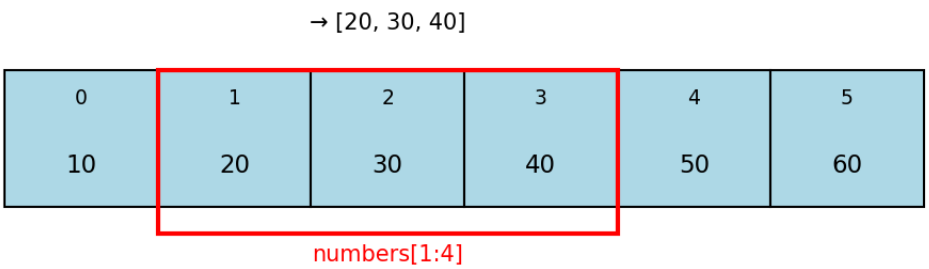
```python
# Creating a list
fruits = ["apple", "banana", "cherry"]

# Accessing
print(fruits[0])   # apple

# Inserting
fruits.insert(1, "mango")

# Removing
fruits.remove("banana")

# Traversal
for fruit in fruits:
    print(fruit)
```

Operation	Syntax	Time Complexity
Access by index	`arr[i]`	O(1)
Search	`x in arr`	O(n)
Insert at end	`append(x)`	O(1) amortized
Insert at index	`insert(i, x)`	O(n)
Delete by value	`remove(x)`	O(n)

🧮 4. Slicing and Indexing

Python lists support **powerful slicing syntax**.

```
numbers = [10, 20, 30, 40, 50, 60]
print(numbers[1:4])      # [20, 30, 40]
print(numbers[::-1])     # reversed
```



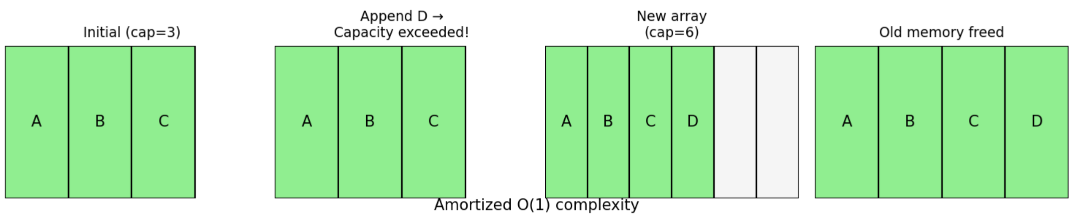
List Slicing Diagram

Slicing is O(k) where *k* is the length of the slice.

🔁 5. Dynamic Resizing and Memory Efficiency

Unlike static arrays, Python lists **grow dynamically**. When you append beyond capacity, Python internally allocates a new memory block, copies old data, and frees the old one.

Visualization:



Dynamic List Resizing

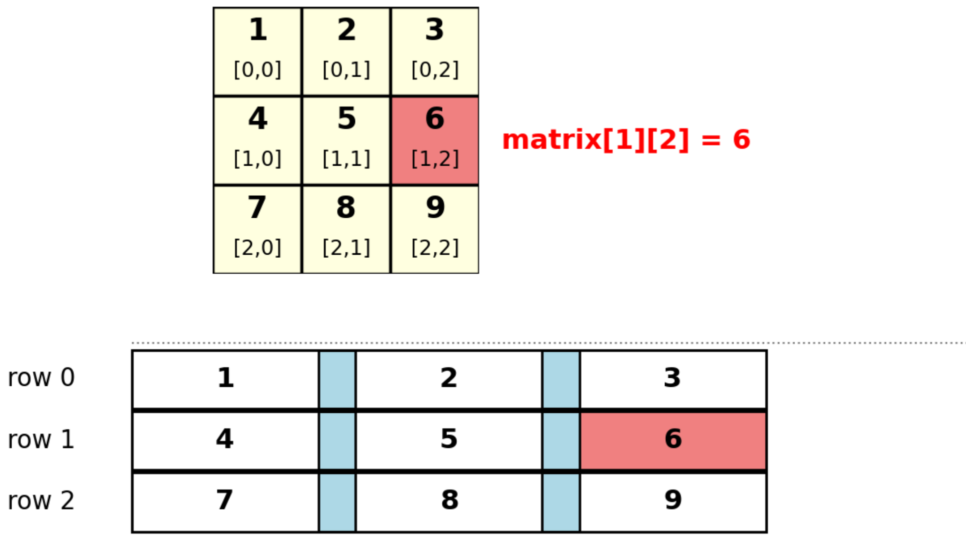
This mechanism is efficient on average but can be costly during resizing bursts — known as **amortized cost**.

---

6. Multidimensional Lists

You can represent a **matrix** or **grid** using nested lists.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print(matrix[1][2])  # 6
```



Matrix Representation

---

7. Example: Searching and Sorting

```
# Linear search
def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1
```

```python
# Sorting
numbers = [5, 2, 9, 1]
numbers.sort()
print(numbers)
```

Output:

```
[1, 2, 5, 9]
```

Complexity:

- Linear search → O(n)
- Python's `sort()` → O(n log n)

🧩 In-Class Exercises

Exercise 1 — Array Operations

Create a list of 5 student names. Perform these operations:

1. Add one new name at the end.
2. Remove the second element.
3. Insert a new name at index 2.
4. Print the final list.

Exercise 2 — Search Function

Implement a function `find_max(arr)` that returns the largest element in a list.

Exercise 3 — Two-Dimensional Access

Using a 3×3 matrix, print all diagonal elements.

🏠 Take-Home Assignments

Assignment 1 — Custom Array Class

Implement a class `MyArray` that supports:

- `insert(value)`
- `delete(index)`
- `search(value)`
- `display()`

Assignment 2 — Reverse List Function

Write a function `reverse_list(lst)` that reverses a list **without using** `reversed()` or slicing.

Assignment 3 — Performance Comparison

Use the `timeit` module to compare:

- List append performance
- Insertion at index 0 Record and explain results.

❌ Common Mistakes & 💡 Best Practices

Common Mistakes

- ❌ Mixing data types in numeric arrays (`array('i', [1, '2'])` invalid)
- ❌ Using `insert(0, x)` repeatedly — poor performance
- ❌ Assuming slicing is O(1)
- ❌ Forgetting list indices start from 0

Best Practices

- ✅ Use `array` module for homogeneous data (numeric-heavy)
- ✅ Use list comprehensions for concise creation
- ✅ Benchmark critical code with `timeit`

- ✅ Understand when to copy lists vs reference them

🔭 Next Week Preview

In **Week 3**, we'll explore:

1. **Stacks and Queues** — LIFO & FIFO principles
2. **Implementation using lists and deque**
3. **Practical use-cases in recursion and buffering**
4. **Performance profiling**
