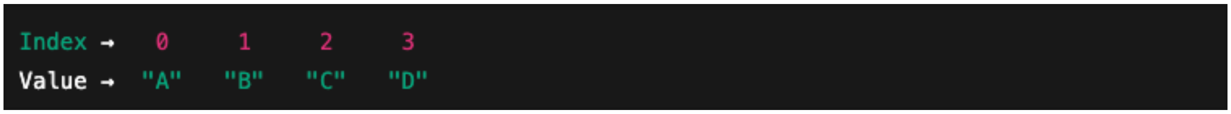
What Are Lists, Abstractly?

In programming, a **list** is an **ordered collection** of items.

That means:

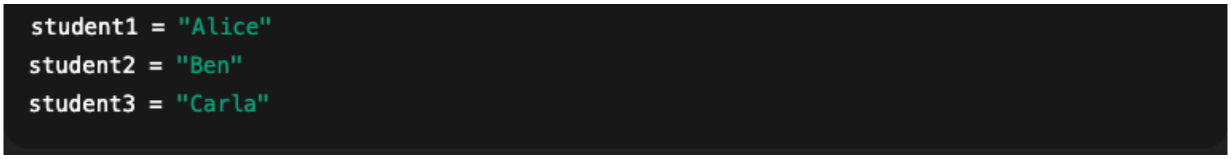
- Each element in the list has a specific **position (index)**.
- The **order matters** — the sequence is preserved.
- You can **add**, **remove**, and **rearrange** elements.

Think of a **list like a row of labeled boxes**, each holding a value.
Example:



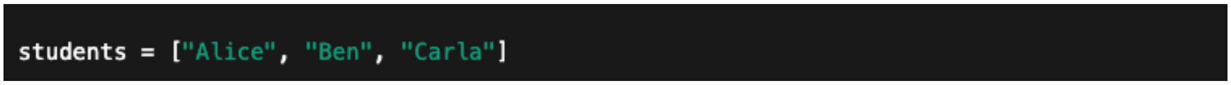
```
Index →   0    1    2    3
Value →  "A"  "B"  "C"  "D"
```

Unlike variables that hold a *single* piece of data, lists let you **group related information together** under one name.
So instead of having:



```
student1 = "Alice"
student2 = "Ben"
student3 = "Carla"
```

you can simply use:



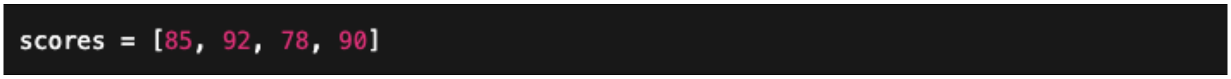
```
students = ["Alice", "Ben", "Carla"]
```

Now, students becomes one logical unit you can loop through, modify, or print — instead of managing separate variables.

Why Do We Use Lists?

1.Organization of Related Data

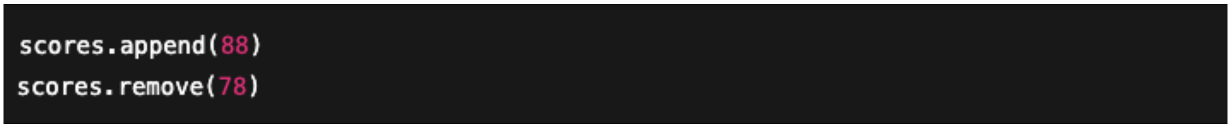
Lists make it easier to store collections of similar or related values — names, scores, numbers, etc.



```
scores = [85, 92, 78, 90]
```

2.Dynamic Size

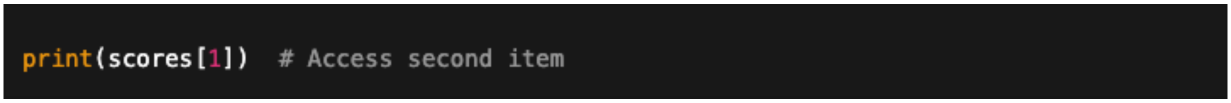
Unlike arrays in many other languages, Python lists can **grow or shrink**.
You can add or remove items as your data changes.



```
scores.append(88)
scores.remove(78)
```

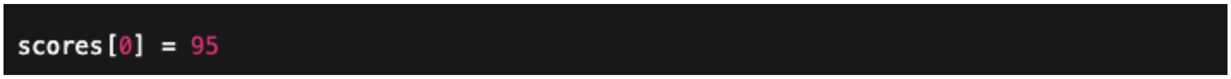
3.Easy Data Access and Modification

You can access any item instantly using its index:



```
print(scores[1])  # Access second item
```

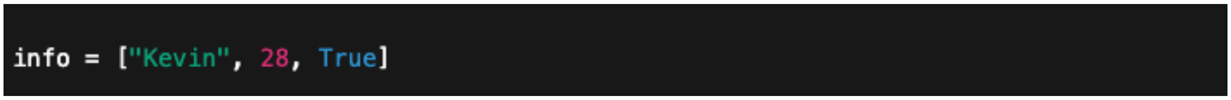
or update values:



```
scores[0] = 95
```

4.Flexibility with Data Types

Lists can hold **mixed data**:

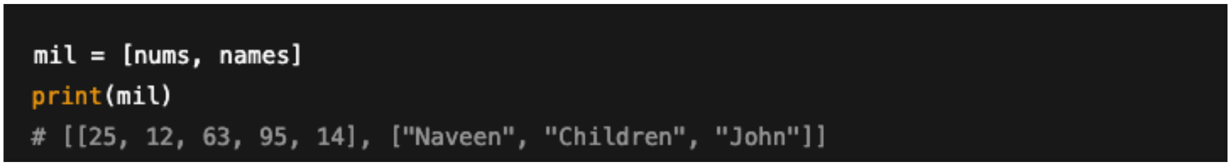


```
info = ["Kevin", 28, True]
```

This flexibility makes Python lists useful for quick data structures in scripts and projects.

5. Nested Lists (Lists Inside Lists)

- Lists can contain **other lists** as elements.



```
 mil = [nums, names]
 print(mil)
 # [[25, 12, 63, 95, 14], ["Naveen", "Children", "John"]]
```

6. List Mutability & Common Methods

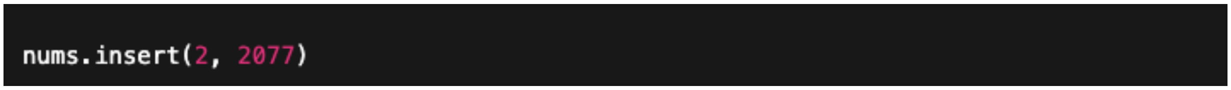
- **Lists are mutable**, meaning you can change their content after creation.
- You can explore available list methods by typing list_name. in your IDE.

Method	Description	Example
append(x)	Add one element to the end	nums.append(45)
clear()	Remove all elements	nums.clear()

copy()	Make a duplicate of the list	nums2 = nums.copy()
count(x)	Count how many times a value appears	nums.count(12)
extend([...])	Add multiple elements at once	nums.extend([29, 12])
index(x)	Find the position of a value	nums.index(63)

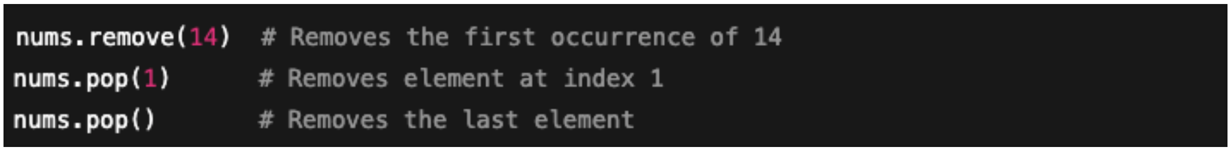
7. Inserting & Removing Elements

- **Insert** an element at a specific position:



```
nums.insert(2, 2077)
```

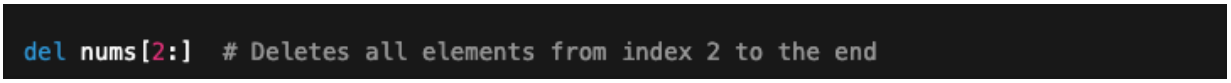
- **Remove elements:**



```
nums.remove(14)  # Removes the first occurrence of 14
nums.pop(1)      # Removes element at index 1
nums.pop()       # Removes the last element
```

8. Deleting Multiple Elements

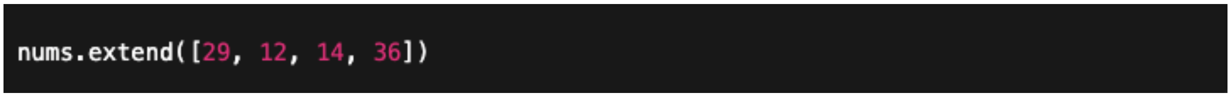
- Use del to delete one or more elements:



```
del nums[2:]  # Deletes all elements from index 2 to the end
```

9. Adding Multiple Elements

- Use extend() to add multiple values at once:



```
nums.extend([29, 12, 14, 36])
```

Python Lists — Global Cheat Sheet

Category	Syntax / Function	Description / Example	Example Output
List Creation	my_list = [1, 2, 3]	Create a list with elements	[1, 2, 3]

Access by Index	my_list[0]	Access first element (zero-based index)	1
Negative Indexing	my_list[-1]	Access last element	3
Slicing	my_list[1:3]	Access elements from index 1 up to (but not including) 3	[2, 3]
Nested Lists	nested = [[1,2], [3,4]]	Lists can contain other lists	[[1,2],[3,4]]
Heterogeneous Lists	values = [42, "Hi", 9.5]	Lists can hold mixed data types	[42, 'Hi', 9.5]
Mutability	my_list[1] = 99	Lists can be changed after creation	[1, 99, 3]
Appending	my_list.append(4)	Adds an element to the end	[1, 2, 3, 4]
Inserting	my_list.insert(1, 50)	Adds element at index 1	[1, 50, 2, 3]
Removing by Value	my_list.remove(2)	Removes the first matching value	[1, 3]
Popping by Index	my_list.pop(0)	Removes and returns element at index 0	Returns 1
Deleting Multiple	del my_list[1:3]	Deletes elements in slice range	Remaining list only
Extending	my_list.extend([7, 8])	Adds multiple elements	[1, 2, 3, 7, 8]
Clearing	my_list.clear()	Removes all elements	[]
Copying	copy_list = my_list.copy()	Returns a shallow copy	Duplicate of list
Counting	my_list.count(2)	Counts how many times a value appears	1
Finding Index	my_list.index(3)	Returns position of first match	2

Sorting	my_list.sort()	Sorts list in place	[1, 2, 3, 4]
Reversing	my_list.reverse()	Reverses order in place	[4, 3, 2, 1]
Built-in Functions	len(my_list)	Number of elements	4
	min(my_list)	Smallest element	1
	max(my_list)	Largest element	4
	sum(my_list)	Sum of numeric elements	10
Membership Test	x in my_list	Check if value exists	True / False
Iteration	for item in my_list:	Loop through each element	Prints all elements
Concatenation	list1 + list2	Join two lists	[1,2,3,4,5]
Repetition	list1 * 3	Repeat elements	[1,2,1,2,1,2]