Creating, Indexing & Slicing Lists

AiCE Software Engineering Foundations — Python Programming

A variable holds one value. A list holds many. This lesson opens Python's most versatile and widely used data structure — the foundation for storing, organizing, and processing collections of related data that every real application relies on.

1. What Is a List?

A list is an ordered, mutable collection of items. Three words matter in that definition:

Ordered — the items in a list have a defined sequence. The first item is always first, the second is always second.

Mutable — a list can be changed after it's created. You can add, remove, and update items freely.

Collection — a list stores multiple values in a single variable, under one name.

Without lists, storing multiple related values would require a separate variable for each one — ten students would need ten separate variables, a hundred would need a hundred. A list solves this entirely: you store all of them in one variable and access each by its position.

Lists are written with square brackets, items separated by commas:

```
countries = ["Nigeria", "Ghana", "Kenya", "Senegal", "Rwanda"]
scores    = [78, 85, 92, 67, 95]
mixed     = ["Amara", 22, 3.85, True]
```

> **Keep Lists to One Consistent Type**
> *Python lists can hold any combination of data types, including other lists, as the mixed example above shows. In practice, you will almost always keep a list to one consistent type — all strings, or all numbers — for clarity and predictability.*

2. Creating and Printing Lists

```
# Creating lists
african_cities = ["Lagos", "Nairobi", "Accra", "Kigali",
"Dakar", "Cairo"]
exam_scores    = [82, 75, 91, 68, 88, 79, 95]
tech_stack     = ["Python", "JavaScript", "SQL", "Git",
"Linux"]

# Printing the entire list
print(african_cities)
print(type(african_cities))

# Checking the length
print(f"Number of cities: {len(african_cities)}")
```

This produces:

```
['Lagos', 'Nairobi', 'Accra', 'Kigali', 'Dakar', 'Cairo']
<class 'list'>
Number of cities: 6
```

> **Printing a Raw List vs. Looping Over It**
> *Printing a list directly shows Python's raw representation — square brackets, commas, quotes around strings. In most real programs, you will loop through a list to display its items one by one, exactly as covered in the previous unit, rather than printing the entire list at once.*

3. Accessing List Items by Index

Every item in a list has a position called an index. Python uses zero-based indexing — the first item is at index 0, the second at index 1, and so on — the same indexing system used for strings in an earlier unit.

```
african_cities = ["Lagos", "Nairobi", "Accra", "Kigali",
"Dakar", "Cairo"]

# Accessing by positive index
print(african_cities[0])    # Lagos — first item
print(african_cities[5])    # Cairo — last item

# Accessing by negative index (counting from the end)
print(african_cities[-1])   # Cairo — last item
print(african_cities[-2])   # Dakar — second to last
```

Negative indexing counts from the end of the list. Index -1 always refers to the last item, -2 to the second-to-last, and so on — extremely useful when you want the last item but don't know the list's length in advance.

> **IndexError — Going Out of Range**
> *Trying to access an index that doesn't exist — for example african_cities[10] on a 6-item list — raises an IndexError. Always ensure your index is within the valid range: 0 to len(list) - 1 for positive indexes, or -1 to -len(list) for negative indexes.*

4. List Slicing

Just like strings, lists support slicing — extracting a portion using [start:stop] syntax, where start is inclusive and stop is exclusive:

```
tech_stack = ["Python", "JavaScript", "SQL", "Git", "Linux",
"Docker", "AWS"]

print(tech_stack[0:3])     # First 3 items
print(tech_stack[3:])      # From index 3 to the end
print(tech_stack[-3:])     # Last 3 items
print(tech_stack[:])       # Entire list (a copy)

# Slicing with a step
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(numbers[::2])        # Every second item
print(numbers[::-1])       # Reverse the list
```

This produces:

```
['Python', 'JavaScript', 'SQL']
['Git', 'Linux', 'Docker', 'AWS']
['Docker', 'AWS']
['Python', 'JavaScript', 'SQL', 'Git', 'Linux', 'Docker',
'AWS']
[0, 2, 4, 6, 8]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

> **Slicing Never Modifies the Original**
> *The step value in slicing works the same way as in range(), and [::-1] is a commonly used technique for reversing a list. Slicing always returns a brand new list containing the selected items — the original list is never changed.*

Key Takeaways

A list is an ordered, mutable collection written with square brackets, and every item inside it has a zero-based index you can access directly or with negative indexing from the end. Slicing with [start:stop:step] extracts a portion of a list without ever modifying the original. The next lesson covers changing a list's contents — updating items, adding new ones, removing existing ones, and searching for what's already there.