Python Programming - Beginner Level Notes

Prepared for: Bikkad IT Institute Students
Topic: Python Lists
Date: February 2026

---

1. What are Lists?

Definition

A list is an ordered, mutable collection that can store multiple items of different data types.

Key Features:

- Ordered - Items maintain their position
- Mutable - Can be changed after creation
- Allow duplicates
- Can contain any data type
- Defined using square brackets [ ]

---

2. Array vs List

Feature	Array	List
Data Type	Same type only	Any type
Size	Fixed size	Dynamic size
Performance	Faster	Slower
Memory	Less memory	More memory
Module	numpy required	Built-in

Table 1: Array vs List Comparison

Code Example:

List - different data types

my_list = [1, "Python", 3.14, True]

Array - same data type only

import numpy as np
my_array = np.array([1, 2, 3, 4])

---

3. How Lists are Stored in Memory

Memory Storage

Lists store references (memory addresses) to objects, not the actual objects.

Diagram:
List: [10, 20, 30]
↓ ↓ ↓
Address: 1000 → 10
Address: 2000 → 20
Address: 3000 → 30

Code Example:
list1 = [1, 2, 3]
list2 = list1 # Both point to same memory

list2[0] = 100
print(list1) # [100, 2, 3] - changed!
print(list2) # [100, 2, 3]

Check memory address

print(id(list1)) # Same
print(id(list2)) # Same

---

4. Create a List

Different Ways to Create Lists

Code Examples:

Empty list

empty_list = []

List with values

numbers = [1, 2, 3, 4, 5]

Mixed data types

mixed = [1, "Python", 3.14, True, [1, 2]]

Using list() constructor

fruits = list(("Apple", "Banana", "Mango"))

Using range

numbers = list(range(1, 6)) # [1, 2, 3, 4, 5]

List comprehension

squares = [x**2 for x in range(1, 6)] # [1, 4, 9, 16, 25]

---

5. Accessing Items from a List

Indexing and Slicing

Code Examples:

fruits = ["Apple", "Banana", "Mango", "Orange", "Grapes"]

Positive indexing

print(fruits[0]) # Apple
print(fruits[2]) # Mango

Negative indexing

print(fruits[-1]) # Grapes
print(fruits[-2]) # Orange

Slicing

print(fruits[1:4]) # ['Banana', 'Mango', 'Orange']
print(fruits[:3]) # ['Apple', 'Banana', 'Mango']
print(fruits[2:]) # ['Mango', 'Orange', 'Grapes']
print(fruits[::2]) # ['Apple', 'Mango', 'Grapes']
print(fruits[::-1]) # Reverse list

Check if item exists

print("Mango" in fruits) # True

---

6. Adding Items to a List

Method	Description
append(item)	Add single item at end
extend(iterable)	Add multiple items at end
insert(index, item)	Add item at specific position

Table 2: Methods to Add Items

Code Examples

Example 1: append()
fruits = ["Apple", "Banana"]

fruits.append("Mango")
print(fruits) # ['Apple', 'Banana', 'Mango']

fruits.append("Orange")
print(fruits) # ['Apple', 'Banana', 'Mango', 'Orange']

Example 2: extend()
list1 = [1, 2, 3]
list2 = [4, 5, 6]

list1.extend(list2)
print(list1) # [1, 2, 3, 4, 5, 6]

Extend with string (adds each character)

list1.extend("AB")
print(list1) # [1, 2, 3, 4, 5, 6, 'A', 'B']

Example 3: insert()
fruits = ["Apple", "Mango", "Orange"]

fruits.insert(1, "Banana") # Insert at index 1
print(fruits) # ['Apple', 'Banana', 'Mango', 'Orange']

fruits.insert(0, "Grapes") # Insert at start
print(fruits) # ['Grapes', 'Apple', 'Banana', 'Mango', 'Orange']

---

7. Editing Items in a List

Code Examples

```
fruits = ["Apple", "Banana", "Mango"]

# Edit single item

fruits[1] = "Grapes"
print(fruits) # ['Apple', 'Grapes', 'Mango']

# Edit multiple items using slicing

numbers = [1, 2, 3, 4, 5]
numbers[1:4] = [20, 30, 40]
print(numbers) # [1, 20, 30, 40, 5]

# Replace with different number of items

fruits = ["Apple", "Banana", "Mango", "Orange"]
fruits[1:3] = ["Grapes"]
print(fruits) # ['Apple', 'Grapes', 'Orange']
```

---

8. Deleting Items in a List

Method	Description
del	Delete by index or slice
remove(item)	Remove first occurrence of value
pop(index)	Remove and return item at index
clear()	Remove all items

Table 3: Methods to Delete Items

Code Examples

Example 1: del keyword

```
fruits = ["Apple", "Banana", "Mango", "Orange"]

# Delete by index

del fruits[1]
print(fruits) # ['Apple', 'Mango', 'Orange']
```

Delete slice

del fruits[0:2]
print(fruits) # ['Orange']

Delete entire list

del fruits

Example 2: remove()
fruits = ["Apple", "Banana", "Mango", "Banana"]

fruits.remove("Banana") # Removes first occurrence
print(fruits) # ['Apple', 'Mango', 'Banana']

Example 3: pop()
fruits = ["Apple", "Banana", "Mango"]

Remove and return last item

removed = fruits.pop()
print(removed) # Mango
print(fruits) # ['Apple', 'Banana']

Remove and return at specific index

removed = fruits.pop(0)
print(removed) # Apple
print(fruits) # ['Banana']

Example 4: clear()
fruits = ["Apple", "Banana", "Mango"]

fruits.clear()
print(fruits) # []

---

9. Operators on a List

Arithmetic Operators

Code Examples:

Concatenation (+)

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
result = list1 + list2
print(result) # [1, 2, 3, 4, 5, 6]
```

Repetition (*)

```
numbers = [1, 2]
result = numbers * 3
print(result) # [1, 2, 1, 2, 1, 2]
```

Membership Operators

Code Examples:

```
fruits = ["Apple", "Banana", "Mango"]

print("Apple" in fruits) # True
print("Grapes" in fruits) # False
print("Grapes" not in fruits) # True
```

Loop Through List

Code Examples:

```
fruits = ["Apple", "Banana", "Mango"]
```

Using for loop

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

Using for loop with index

```
for i in range(len(fruits)):
print(f"{i}: {fruits[i]}")
```

Using enumerate

```
for index, fruit in enumerate(fruits):
print(f"{index}: {fruit}")
```

Using while loop

```
i = 0
while i < len(fruits):
print(fruits[i])
i += 1
```

---

10. List Functions

Function	Description
len(list)	Number of items
min(list)	Minimum value
max(list)	Maximum value
sorted(list)	Return sorted copy
count(item)	Count occurrences
index(item)	Find index of first occurrence
reverse()	Reverse list in place
sort()	Sort list in place
copy()	Create shallow copy

Table 4: Common List Functions

Code Examples

Example 1: len(), min(), max()

```
numbers = [10, 50, 30, 20, 40]

print(len(numbers)) # 5
print(min(numbers)) # 10
print(max(numbers)) # 50
```

With strings

```
fruits = ["Apple", "Banana", "Mango"]
print(min(fruits)) # Apple (alphabetically)
print(max(fruits)) # Mango
```

Example 2: sorted()

```
numbers = [50, 10, 30, 20, 40]
```

Returns new sorted list (original unchanged)

```
sorted_nums = sorted(numbers)
print(sorted_nums) # [10, 20, 30, 40, 50]
print(numbers) # [50, 10, 30, 20, 40]
```

Descending order

```
desc = sorted(numbers, reverse=True)
print(desc) # [50, 40, 30, 20, 10]
```

Example 3: count() and index()

```
numbers = [1, 2, 3, 2, 4, 2, 5]

print(numbers.count(2)) # 3
print(numbers.index(3)) # 2 (first occurrence at index 2)
```

Example 4: reverse()

```
fruits = ["Apple", "Banana", "Mango"]

fruits.reverse() # Modifies original list
print(fruits) # ['Mango', 'Banana', 'Apple']
```

Example 5: sort()

```
numbers = [50, 10, 30, 20, 40]

numbers.sort() # Modifies original list
print(numbers) # [10, 20, 30, 40, 50]

numbers.sort(reverse=True) # Descending
print(numbers) # [50, 40, 30, 20, 10]
```

Example 6: copy()

```
list1 = [1, 2, 3]
list2 = list1.copy() # Create independent copy

list2[0] = 100
print(list1) # [1, 2, 3] - unchanged
print(list2) # [100, 2, 3]
```

---

11. List Comprehension

What is List Comprehension?

Concise way to create lists using a single line of code.

Syntax: `[expression for item in iterable if condition]`

Code Examples

Example 1: Basic Comprehension

```
# Traditional way

squares = []
for x in range(1, 6):
squares.append(x**2)
print(squares) # [1, 4, 9, 16, 25]

# List comprehension

squares = [x**2 for x in range(1, 6)]
print(squares) # [1, 4, 9, 16, 25]
```

Example 2: With Condition

```
# Even numbers only

evens = [x for x in range(1, 11) if x % 2 == 0]
print(evens) # [2, 4, 6, 8, 10]

# Numbers divisible by 3

div_by_3 = [x for x in range(1, 21) if x % 3 == 0]
print(div_by_3) # [3, 6, 9, 12, 15, 18]
```

Example 3: String Manipulation

```
# Convert to uppercase

fruits = ["apple", "banana", "mango"]
upper_fruits = [fruit.upper() for fruit in fruits]
print(upper_fruits) # ['APPLE', 'BANANA', 'MANGO']

# Get lengths

lengths = [len(fruit) for fruit in fruits]
print(lengths) # [5, 6, 5]
```

Example 4: Nested Comprehension

Create 3x3 matrix

matrix = [[i*j for j in range(1, 4)] for i in range(1, 4)]
print(matrix)

[[1, 2, 3], [2, 4, 6], [3, 6, 9]]

---

12. Ways to Reverse a List

Item-wise Reversal

Code Examples:

numbers = [1, 2, 3, 4, 5]

Method 1: Using reverse()

numbers.reverse()
print(numbers) # [5, 4, 3, 2, 1]

Method 2: Using slicing

numbers = [1, 2, 3, 4, 5]
reversed_list = numbers[::-1]
print(reversed_list) # [5, 4, 3, 2, 1]

Method 3: Using reversed()

numbers = [1, 2, 3, 4, 5]
reversed_list = list(reversed(numbers))
print(reversed_list) # [5, 4, 3, 2, 1]

Index-wise Reversal

Code Examples:

numbers = [1, 2, 3, 4, 5]

Using loop with negative indexing

reversed_list = []
for i in range(len(numbers)-1, -1, -1):
reversed_list.append(numbers[i])
print(reversed_list) # [5, 4, 3, 2, 1]

Swap elements from both ends

numbers = [1, 2, 3, 4, 5]
for i in range(len(numbers)//2):
numbers[i], numbers[-(i+1)] = numbers[-(i+1)], numbers[i]
print(numbers) # [5, 4, 3, 2, 1]

---

13. zip() Function

What is zip()?

Combines multiple iterables into tuples.

Code Examples:

Combine two lists

names = ["Rahul", "Priya", "Amit"]
ages = [25, 22, 28]

combined = list(zip(names, ages))
print(combined)

[('Rahul', 25), ('Priya', 22), ('Amit', 28)]

Unzip

names, ages = zip(*combined)
print(list(names)) # ['Rahul', 'Priya', 'Amit']
print(list(ages)) # [25, 22, 28]

Three lists

subjects = ["Math", "Science", "English"]
marks = [85, 90, 88]
grades = ["A", "A+", "A"]

result = list(zip(subjects, marks, grades))
print(result)

[('Math', 85, 'A'), ('Science', 90, 'A+'), ('English', 88, 'A')]

Different length lists (stops at shortest)

list1 = [1, 2, 3, 4]
list2 = ['a', 'b']
print(list(zip(list1, list2))) # [(1, 'a'), (2, 'b')]

---

14. List Can Contain Any Kind of Object

Code Examples

Mixed data types

mixed_list = [
42, # Integer
3.14, # Float
"Python", # String
True, # Boolean
[1, 2, 3], # List
(4, 5, 6), # Tuple
{"name": "Raj"}, # Dictionary
{7, 8, 9} # Set
]

print(mixed_list)

Nested lists (2D list)

matrix = [
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
]

print(matrix[0][1]) # 2
print(matrix[2][2]) # 9

List of dictionaries

students = [
{"name": "Rahul", "age": 25},
{"name": "Priya", "age": 22},

```
{"name": "Amit", "age": 28}
]

print(students[0]["name"]) # Rahul
```

---

15. Disadvantages of Python Lists

Disadvantage	Description
Slower Performance	Slower than arrays for large data
High Memory Usage	Stores type info for each element
Not Type-Safe	Can mix types, leading to errors
No Built-in Math	Can't do element-wise operations
Index-based Access	No key-based access like dict

Table 5: Disadvantages of Lists

Code Examples

Example 1: Memory Comparison

```
import sys

# List uses more memory
my_list = [1, 2, 3, 4, 5]
print(sys.getsizeof(my_list)) # More bytes

# Array uses less memory
import array
my_array = array.array('i', [1, 2, 3, 4, 5])
print(sys.getsizeof(my_array)) # Fewer bytes
```

Example 2: Type Safety Issue

```
# Lists allow mixed types (can cause errors)
numbers = [1, 2, "three", 4, 5]

# This will fail
```

total = sum(numbers) # Error: can't add int and str

Example 3: No Element-wise Operations

Lists don't support element-wise math

list1 = [1, 2, 3]
list2 = [4, 5, 6]

This doesn't work

result = list1 + list2 # Concatenation, not addition

Need to use loop

result = [list1[i] + list2[i] for i in range(len(list1))]
print(result) # [5, 7, 9]

With NumPy arrays (easier)

import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = arr1 + arr2
print(result) # [5 7 9]

Practice Exercises

Exercise 1: List Creation

Create a list of first 10 even numbers using list comprehension.

Exercise 2: List Operations

Given two lists, merge them, remove duplicates, and sort.

Exercise 3: Find Maximum and Minimum

Find the second largest and second smallest number in a list.

Exercise 4: List Manipulation

Remove all odd numbers from a list without using list comprehension.

Exercise 5: Reverse List

Reverse a list without using reverse() or slicing.

Exercise 6: Count Elements

Count how many times each element appears in a list.

Exercise 7: Nested Lists

Create a 4x4 multiplication table using nested list comprehension.

Exercise 8: Zip Function

Combine three lists (names, ages, cities) and print formatted output.

Quick Reference

Operation	Syntax
Create	[1, 2, 3] or list()
Access	list[index]
Slice	list[start:stop:step]
Add	append(), extend(), insert()
Remove	del, remove(), pop(), clear()
Length	len(list)
Sort	sort(), sorted()
Reverse	reverse(), [::-1]
Copy	list.copy()
Comprehension	[x for x in range()]
Zip	zip(list1, list2)

Table 6: Python Lists Quick Reference

End of Notes

Prepared by: Bikkad IT Institute
For: Python Beginner Level Students
Date: February 2026