Prepared for: Bikkad IT Institute Students

Topic: Python Lists - Complete Guide

Date: February 2026

Python Programming - Beginner Level

Notes

1. What are Lists?

Definition

A list is an ordered, mutable collection that can store items of different data types. Lists are defined using square brackets [].

Feature	Description
Ordered	Elements maintain their position
Mutable	Can be modified after creation
Heterogeneous	Can contain different data types
Duplicates Allowed	Same element can appear multiple times
Indexed	Access elements using index (0-based)

Table 1: List Characteristics

Code Examples

Example 1: Empty List

```
my_list = []
print(my_list)
print(type(my_list))
```

```
Output: []
```

Example 2: Integer List

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

```
Output: [1, 2, 3, 4, 5]
```

Example 3: String List

```
fruits = ['apple', 'banana', 'cherry']
print(fruits)
```

```
Output: ['apple', 'banana', 'cherry']
```

Example 4: Mixed Types

```
mixed = [1, 'hello', 3.14, True]
print(mixed)
```

```
Output: [1, 'hello', 3.14, True]
```

Example 5: Nested List

```
nested = [[1, 2], [3, 4], [5, 6]]
print(nested)
```

```
Output: [[1, 2], [3, 4], [5, 6]]
```

Example 6: List with Duplicates

```
items = [1, 2, 2, 3, 3, 3]
print(items)
```

```
Output: [1, 2, 2, 3, 3, 3]
```

Example 7: List Length

```
numbers = [10, 20, 30, 40]
print(f'Length: {len(numbers)}')
```

```
Output: Length: 4
```

Example 8: Boolean List

```
flags = [True, False, True, True]
print(flags)
```

```
Output: [True, False, True, True]
```

Example 9: List from Range

```
nums = list(range(1, 6))
print(nums)
```

```
Output: [1, 2, 3, 4, 5]
```

Example 10: List from String

```
chars = list('Python')
print(chars)
```

```
Output: ['P', 'y', 't', 'h', 'o', 'n']
```

2. Array vs List

Comparison

Feature	List	Array
Data Types	Can store mixed types	Stores same type only
Size	Dynamic (can grow/shrink)	Fixed size (in some langs)
Memory	More memory overhead	Memory efficient
Performance	Slower for numeric ops	Faster for numeric ops
Module	Built-in Python	Requires array/numpy module

Table 2: Array vs List Comparison

Code Examples

Example 1: List - Mixed Types

```
my_list = [1, 'hello', 3.14]
print(my_list)
```

```
Output: [1, 'hello', 3.14]
```

Example 2: Array - Same Type

```
import array
arr = array.array('i', [1, 2, 3])
print(arr)
```

```
Output: array('i', [1, 2, 3])
```

Example 3: List - Dynamic Size

```
lst = [1, 2, 3]
lst.append(4)
print(lst)
```

```
Output: [1, 2, 3, 4]
```

Example 4: List Flexibility

```
data = [1, 2, 3]
data.extend(['a', 'b'])
print(data)
```

```
Output: [1, 2, 3, 'a', 'b']
```

Example 5: Array Type Check

```
import array
arr = array.array('i', [1, 2, 3])
print(arr.typecode)
```

```
Output: i
```

Example 6: List Memory

```
import sys
lst = [1, 2, 3, 4, 5]
print(f'Size: {sys.getsizeof(lst)} bytes')
```

```
Output: Size: 104 bytes
```

Example 7: List Operations

```
nums = [1, 2, 3]
result = nums + [4, 5]
print(result)
```

```
Output: [1, 2, 3, 4, 5]
```

Example 8: List Versatility

```
items = []
items.append(1)
items.append('text')
print(items)
```

```
Output: [1, 'text']
```

Example 9: List Methods

```
lst = [3, 1, 4, 1, 5]
lst.sort()
print(lst)
```

```
Output: [1, 1, 3, 4, 5]
```

Example 10: Array Limitation

```
import array
# arr = array.array('i', [1, 'text'])  # Error!
print('Arrays need same type')
```

```
Output: Arrays need same type
```

3. How Lists are Stored in Memory

Memory Storage Concept

Lists in Python store references (pointers) to objects, not the actual objects. This allows heterogeneous data types but uses more memory.

Code Examples

Example 1: Memory ID

```
lst = [1, 2, 3]
print(f'List ID: {id(lst)}')
print(f'First item ID: {id(lst[0])}')
```

```
Output: List ID: 140234567890
First item ID: 140234567800
```

Example 2: Reference Storage

```
a = [1, 2, 3]
b = a
print(f'a ID: {id(a)}')
print(f'b ID: {id(b)}')
```

```
Output: a ID: 140234567890
b ID: 140234567890
```

Example 3: Shallow Copy

```
original = [1, 2, 3]
copy = original
copy.append(4)
print(f'Original: {original}')
```

```
Output: Original: [1, 2, 3, 4]
```

Example 4: Deep Copy

```
import copy
original = [1, 2, 3]
deep = copy.deepcopy(original)
deep.append(4)
print(f'Original: {original}')
```

```
Output: Original: [1, 2, 3]
```

Example 5: List Size

```
import sys
empty = []
one_item = [1]
print(f'Empty: {sys.getsizeof(empty)} bytes')
print(f'One item: {sys.getsizeof(one_item)} bytes')
```

```
Output: Empty: 56 bytes
One item: 64 bytes
```

Example 6: Reference Count

```
import sys
x = [1, 2, 3]
print(f'Ref count: {sys.getrefcount(x)}')
```

```
Output: Ref count: 2
```

Example 7: Nested List Memory

```
nested = [[1, 2], [3, 4]]
print(f'Outer ID: {id(nested)}')
```

```
print(f'Inner ID: {id(nested[0])}')
```

```
Output: Outer ID: 140234567890
Inner ID: 140234568000
```

Example 8: Mutability Effect

```
lst = [1, 2, 3]
id_before = id(lst)
lst.append(4)
id_after = id(lst)
print(f'Same object: {id_before == id_after}')
```

```
Output: Same object: True
```

Example 9: String Pool vs List

```
a = 'hello'
b = 'hello'
print(f'Strings same: {id(a) == id(b)}')
x = [1, 2]
y = [1, 2]
print(f'Lists same: {id(x) == id(y)}')
```

```
Output: Strings same: True
Lists same: False
```

Example 10: Memory Growth

```
lst = []
for i in range(3):
    lst.append(i)
    print(f'{i}: {sys.getsizeof(lst)} bytes')
```

```
Output: 0: 64 bytes
1: 72 bytes
2: 72 bytes
```

4. Create a List

Different Ways to Create Lists

Lists can be created using square brackets [], list() constructor, list comprehension, or other methods.

Code Examples

Example 1: Square Brackets

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

```
Output: [1, 2, 3, 4, 5]
```

Example 2: list() Constructor

```
my_list = list((1, 2, 3, 4))
print(my_list)
```

```
Output: [1, 2, 3, 4]
```

Example 3: Empty List

```
empty1 = []
empty2 = list()
print(f'{empty1}, {empty2}')
```

```
Output: [], []
```

Example 4: From String

```
chars = list('Python')
print(chars)
```

```
Output: ['P', 'y', 't', 'h', 'o', 'n']
```

Example 5: From Range

```
nums = list(range(5))
print(nums)
```

```
Output: [0, 1, 2, 3, 4]
```

Example 6: List Comprehension

```
squares = [x**2 for x in range(1, 6)]
print(squares)
```

```
Output: [1, 4, 9, 16, 25]
```

Example 7: With Multiplication

```
zeros = [0] * 5
print(zeros)
```

```
Output: [0, 0, 0, 0, 0]
```

Example 8: From Tuple

```
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list)
```

```
Output: [1, 2, 3]
```

Example 9: From Set

```
my_set = {3, 1, 2}
my_list = list(my_set)
print(sorted(my_list))
```

```
Output: [1, 2, 3]
```

Example 10: Nested List

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

```
Output: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

5. Accessing Items from a List

Indexing and Slicing

Access list elements using index (positive or negative) or slicing [start:stop:step].

Code Examples

Example 1: Positive Index

```
fruits = ['apple', 'banana', 'cherry']
print(fruits[0])
print(fruits[2])
```

```
Output: apple
cherry
```

Example 2: Negative Index

```
numbers = [10, 20, 30, 40, 50]
print(numbers[-1])
print(numbers[-3])
```

```
Output: 50
30
```

Example 3: Slicing

```
nums = [0, 1, 2, 3, 4, 5]
print(nums[1:4])
print(nums[:3])
```

```
Output: [1, 2, 3]
[0, 1, 2]
```

Example 4: Step Slicing

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(numbers[::2])
print(numbers[1::2])
```

```
Output: [0, 2, 4, 6, 8]
[1, 3, 5, 7, 9]
```

Example 5: Reverse with Slice

```
lst = [1, 2, 3, 4, 5]
print(lst[::-1])
```

```
Output: [5, 4, 3, 2, 1]
```

Example 6: Nested Access

```
matrix = [[1, 2], [3, 4], [5, 6]]
print(matrix[0][1])
print(matrix[2][0])
```

```
Output: 2
5
```

Example 7: Multiple Elements

```
colors = ['red', 'green', 'blue', 'yellow']
print(colors[1:3])
```

```
Output: ['green', 'blue']
```

Example 8: Last Three Elements

```
nums = [1, 2, 3, 4, 5, 6, 7]
print(nums[-3:])
```

```
Output: [5, 6, 7]
```

Example 9: Every Third Element

```
numbers = list(range(1, 11))
print(numbers[::3])
```

```
Output: [1, 4, 7, 10]
```

Example 10: Range Access

```
data = list(range(20))
print(data[5:10])
```

```
Output: [5, 6, 7, 8, 9]
```

6. Adding Items to a List

Methods: append(), extend(), insert()

Method	Description	Example
append()	Add single element at end	lst.append(5)
extend()	Add multiple elements at end	lst.extend([4, 5, 6])
insert()	Add element at specific position	lst.insert(0, 10)

Table 3: Adding Methods

Code Examples

Example 1: append() Basic

```
fruits = ['apple', 'banana']
fruits.append('cherry')
print(fruits)
```

```
Output: ['apple', 'banana', 'cherry']
```

Example 2: append() Multiple

```
numbers = [1, 2, 3]
for i in range(4, 7):
    numbers.append(i)
print(numbers)
```

```
Output: [1, 2, 3, 4, 5, 6]
```

Example 3: append() List

```
lst = [1, 2]
lst.append([3, 4])
print(lst)
```

```
Output: [1, 2, [3, 4]]
```

Example 4: extend() Basic

```
lst1 = [1, 2, 3]
lst2 = [4, 5, 6]
lst1.extend(lst2)
print(lst1)
```

```
Output: [1, 2, 3, 4, 5, 6]
```

Example 5: extend() String

```
letters = ['a', 'b']
letters.extend('cd')
print(letters)
```

```
Output: ['a', 'b', 'c', 'd']
```

Example 6: insert() Beginning

```
numbers = [2, 3, 4]
numbers.insert(0, 1)
print(numbers)
```

```
Output: [1, 2, 3, 4]
```

Example 7: insert() Middle

```
lst = [1, 2, 4, 5]
lst.insert(2, 3)
```

```
print(lst)
```

```
Output: [1, 2, 3, 4, 5]
```

Example 8: insert() End

```
fruits = ['apple', 'banana']
fruits.insert(len(fruits), 'cherry')
print(fruits)
```

```
Output: ['apple', 'banana', 'cherry']
```

Example 9: Comparison

```
lst1 = [1]
lst1.append(2)
print(f'append: {lst1}')
lst2 = [1]
lst2.extend([2])
print(f'extend: {lst2}')
```

```
Output: append: [1, 2]
extend: [1, 2]
```

Example 10: Chain Operations

```
nums = []
nums.append(1)
nums.extend([2, 3])
nums.insert(0, 0)
print(nums)
```

```
Output: [0, 1, 2, 3]
```

7. Editing Items in a List

Modifying List Elements

Lists are mutable, so elements can be changed using index assignment or slicing.

Code Examples

Example 1: Single Element

```
numbers = [1, 2, 3, 4, 5]
numbers[2] = 30
print(numbers)
```

```
Output: [1, 2, 30, 4, 5]
```

Example 2: Negative Index

```
fruits = ['apple', 'banana', 'cherry']
fruits[-1] = 'date'
print(fruits)
```

```
Output: ['apple', 'banana', 'date']
```

Example 3: Multiple Elements

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

```
Output: [1, 20, 30, 40, 5]
```

Example 4: Replace with More

```
numbers = [1, 2, 3]
numbers[1:2] = [20, 25, 30]
print(numbers)
```

```
Output: [1, 20, 25, 30, 3]
```

Example 5: Replace with Fewer

```
lst = [1, 2, 3, 4, 5]
lst[1:4] = [99]
print(lst)
```

```
Output: [1, 99, 5]
```

Example 6: Nested List Edit

```
matrix = [[1, 2], [3, 4]]
matrix[0][1] = 20
print(matrix)
```

```
Output: [[1, 20], [3, 4]]
```

Example 7: Update All Even

```
nums = [1, 2, 3, 4, 5]
for i in range(len(nums)):
    if nums[i] % 2 == 0:
        nums[i] *= 10
print(nums)
```

```
Output: [1, 20, 3, 40, 5]
```

Example 8: Swap Elements

```
lst = [10, 20, 30]
lst[0], lst[2] = lst[2], lst[0]
print(lst)
```

```
Output: [30, 20, 10]
```

Example 9: Increment All

```
numbers = [1, 2, 3, 4]
for i in range(len(numbers)):
    numbers[i] += 10
print(numbers)
```

```
Output: [11, 12, 13, 14]
```

Example 10: Conditional Update

```
scores = [45, 67, 89, 34, 92]
for i in range(len(scores)):
    if scores[i] < 50:
        scores[i] = 50
print(scores)
```

```
Output: [50, 67, 89, 50, 92]
```

8. Deleting Items from a List

Methods: del, remove(), pop(), clear()

Method	Description	Returns
del	Delete by index or slice	Nothing
remove()	Delete by value (first occurrence)	Nothing
pop()	Delete by index (default: last)	Deleted element
clear()	Delete all elements	Nothing

Table 4: Deletion Methods

Code Examples

Example 1: del by Index

```
numbers = [1, 2, 3, 4, 5]
del numbers[2]
print(numbers)
```

```
Output: [1, 2, 4, 5]
```

Example 2: del Slice

```
lst = [1, 2, 3, 4, 5, 6]
del lst[1:4]
print(lst)
```

```
Output: [1, 5, 6]
```

Example 3: del Negative Index

```
fruits = ['apple', 'banana', 'cherry']
del fruits[-1]
print(fruits)
```

```
Output: ['apple', 'banana']
```

Example 4: remove() Value

```
colors = ['red', 'green', 'blue', 'green']
colors.remove('green')
print(colors)
```

```
Output: ['red', 'blue', 'green']
```

Example 5: pop() Default

```
stack = [1, 2, 3, 4]
popped = stack.pop()
print(f'Popped: {popped}, Stack: {stack}')
```

```
Output: Popped: 4, Stack: [1, 2, 3]
```

Example 6: pop() Index

```
lst = [10, 20, 30, 40]
item = lst.pop(1)
print(f'Removed: {item}, List: {lst}')
```

```
Output: Removed: 20, List: [10, 30, 40]
```

Example 7: pop() First

```
queue = ['a', 'b', 'c']
first = queue.pop(0)
```

```
print(f'First: {first}, Queue: {queue}')
```

```
Output: First: a, Queue: ['b', 'c']
```

Example 8: clear() Method

```
numbers = [1, 2, 3, 4, 5]
numbers.clear()
print(numbers)
```

```
Output: []
```

Example 9: Remove All Occurrences

```
lst = [1, 2, 3, 2, 4, 2]
while 2 in lst:
    lst.remove(2)
print(lst)
```

```
Output: [1, 3, 4]
```

Example 10: del vs remove

```
nums1 = [1, 2, 3]
del nums1[1]
print(f'del: {nums1}')
nums2 = [1, 2, 3]
nums2.remove(2)
print(f'remove: {nums2}')
```

```
Output: del: [1, 3]
remove: [1, 3]
```

9. Operators on a List

Arithmetic, Membership, and Loop Operations

Operator	Description	Example
+	Concatenation	[1, 2] + [3, 4]
*	Repetition	[1, 2] * 3
in	Membership	3 in [1, 2, 3]
not in	Not in list	5 not in [1, 2, 3]
for	Iteration	for x in list:

Table 5: List Operators

Code Examples

Example 1: Concatenation

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

```
Output: [1, 2, 3, 4, 5, 6]
```

Example 2: Repetition

```
pattern = [0, 1]
repeated = pattern * 3
print(repeated)
```

```
Output: [0, 1, 0, 1, 0, 1]
```

Example 3: Membership in

```
fruits = ['apple', 'banana', 'cherry']
print('banana' in fruits)
```

```
Output: True
```

Example 4: Membership not in

```
numbers = [1, 2, 3, 4, 5]
print(10 not in numbers)
```

```
Output: True
```

Example 5: Comparison ==

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 == list2)
```

```
Output: True
```

Example 6: Comparison <

```
list1 = [1, 2]
list2 = [1, 3]
print(list1 < list2)
```

```
Output: True
```

Example 7: Iterate For Loop

```
colors = ['red', 'green', 'blue']
for color in colors:
    print(color, end=' ')
```

```
Output: red green blue
```

Example 8: Enumerate Loop

```
fruits = ['apple', 'banana']
for i, fruit in enumerate(fruits):
    print(f'{i}: {fruit}')
```

```
Output: 0: apple
1: banana
```

Example 9: While Loop

```
nums = [1, 2, 3]
i = 0
while i < len(nums):
    print(nums[i], end=' ')
    i += 1
```

```
Output: 1 2 3
```

Example 10: List Comprehension Loop

```
numbers = [1, 2, 3, 4]
squares = [x**2 for x in numbers]
print(squares)
```

```
Output: [1, 4, 9, 16]
```

10. List Functions

Built-in Functions and Methods

Function	Description	Example
len()	Length of list	len([1, 2, 3])
min()	Minimum value	min([1, 2, 3])
max()	Maximum value	max([1, 2, 3])
sum()	Sum of elements	sum([1, 2, 3])
sorted()	Return sorted list	sorted([3, 1, 2])
count()	Count occurrences	list.count(2)
index()	Find index	list.index(3)
reverse()	Reverse in place	list.reverse()
sort()	Sort in place	list.sort()
copy()	Shallow copy	list.copy()

Table 6: List Functions

Code Examples

Example 1: len() Function

```
numbers = [1, 2, 3, 4, 5]
print(f'Length: {len(numbers)}')
```

```
Output: Length: 5
```

Example 2: min() Function

```
temps = [23, 18, 30, 15, 25]
print(f'Min: {min(temps)}')
```

```
Output: Min: 15
```

Example 3: max() Function

```
scores = [85, 92, 78, 95, 88]
print(f'Max: {max(scores)}')
```

```
Output: Max: 95
```

Example 4: sum() Function

```
prices = [10.50, 25.75, 15.00]
print(f'Total: {sum(prices)}')
```

```
Output: Total: 51.25
```

Example 5: sorted() Function

```
nums = [3, 1, 4, 1, 5, 9, 2]
print(sorted(nums))
```

```
Output: [1, 1, 2, 3, 4, 5, 9]
```

Example 6: count() Method

```
lst = [1, 2, 2, 3, 2, 4]
print(f'Count of 2: {lst.count(2)}')
```

```
Output: Count of 2: 3
```

Example 7: index() Method

```
fruits = ['apple', 'banana', 'cherry']
print(f'Index: {fruits.index("banana")}')
```

```
Output: Index: 1
```

Example 8: reverse() Method

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

```
Output: [5, 4, 3, 2, 1]
```

Example 9: sort() Method

```
names = ['Charlie', 'Alice', 'Bob']
names.sort()
print(names)
```

```
Output: ['Alice', 'Bob', 'Charlie']
```

Example 10: copy() Method

```
original = [1, 2, 3]
copy = original.copy()
copy.append(4)
print(f'Original: {original}')
```

```
Output: Original: [1, 2, 3]
```

11. List Comprehension

Concise Way to Create Lists

Syntax: [expression for item in iterable if condition]

Code Examples

Example 1: Basic Comprehension

```
squares = [x**2 for x in range(1, 6)]
print(squares)
```

```
Output: [1, 4, 9, 16, 25]
```

Example 2: With Condition

```
evens = [x for x in range(1, 11) if x % 2 == 0]
print(evens)
```

```
Output: [2, 4, 6, 8, 10]
```

Example 3: String Operations

```
names = ['alice', 'bob', 'charlie']
upper = [name.upper() for name in names]
print(upper)
```

```
Output: ['ALICE', 'BOB', 'CHARLIE']
```

Example 4: Mathematical

```
nums = [1, 2, 3, 4]
doubled = [x * 2 for x in nums]
print(doubled)
```

```
Output: [2, 4, 6, 8]
```

Example 5: Filtering

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
odd_squares = [x**2 for x in numbers if x % 2 != 0]
print(odd_squares)
```

```
Output: [1, 9, 25, 49]
```

Example 6: Nested Loop

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

```
Output: [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

Example 7: If-Else

```
nums = [1, 2, 3, 4, 5]
result = ['even' if x % 2 == 0 else 'odd' for x in nums]
print(result)
```

```
Output: ['odd', 'even', 'odd', 'even', 'odd']
```

Example 8: String Filtering

```
words = ['apple', 'banana', 'cherry', 'date']
long_words = [w for w in words if len(w) > 5]
print(long_words)
```

```
Output: ['banana', 'cherry']
```

Example 9: Flattening

```
nested = [[1, 2], [3, 4], [5, 6]]
flat = [item for sublist in nested for item in sublist]
print(flat)
```

```
Output: [1, 2, 3, 4, 5, 6]
```

Example 10: With Functions

```
def square(x):
    return x ** 2
nums = [1, 2, 3, 4]
squares = [square(x) for x in nums]
print(squares)
```

```
Output: [1, 4, 9, 16]
```

12. Ways to Reverse a List

Item-Wise and Index-Wise Reversal

Code Examples

Example 1: reverse() Method

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

```
Output: [5, 4, 3, 2, 1]
```

Example 2: Slicing [::-1]

```
lst = [1, 2, 3, 4, 5]
reversed_lst = lst[::-1]
print(reversed_lst)
```

```
Output: [5, 4, 3, 2, 1]
```

Example 3: reversed() Function

```
nums = [1, 2, 3, 4]
reversed_nums = list(reversed(nums))
print(reversed_nums)
```

```
Output: [4, 3, 2, 1]
```

Example 4: Manual Loop

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

```
Output: [4, 3, 2, 1]
```

Example 5: Using Stack Pop

```
nums = [1, 2, 3, 4, 5]
reversed_nums = []
while nums:
    reversed_nums.append(nums.pop())
print(reversed_nums)
```

```
Output: [5, 4, 3, 2, 1]
```

Example 6: Two Pointer Swap

```
lst = [1, 2, 3, 4, 5]
left, right = 0, len(lst) - 1
while left < right:
    lst[left], lst[right] = lst[right], lst[left]
    left += 1
    right -= 1
print(lst)
```

```
Output: [5, 4, 3, 2, 1]
```

Example 7: Comprehension

```
nums = [1, 2, 3, 4]
rev = [nums[i] for i in range(len(nums)-1, -1, -1)]
print(rev)
```

```
Output: [4, 3, 2, 1]
```

Example 8: Recursive

```
def reverse_recursive(lst):
    if len(lst) == 0:
        return []
    return [lst[-1]] + reverse_recursive(lst[:-1])

print(reverse_recursive([1, 2, 3]))
```

```
Output: [3, 2, 1]
```

Example 9: Negative Indexing

```
lst = [10, 20, 30, 40]
rev = [lst[-i] for i in range(1, len(lst)+1)]
print(rev)
```

```
Output: [40, 30, 20, 10]
```

Example 10: Using Deque

```
from collections import deque
lst = [1, 2, 3, 4]
d = deque(lst)
d.reverse()
print(list(d))
```

```
Output: [4, 3, 2, 1]
```

13. Zip Function

Combining Multiple Lists

zip() function combines elements from multiple iterables into tuples.

Code Examples

Example 1: Basic Zip

```
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]
combined = list(zip(names, ages))
print(combined)
```

```
Output: [('Alice', 25), ('Bob', 30), ('Charlie', 35)]
```

Example 2: Three Lists

```
a = [1, 2, 3]
b = ['a', 'b', 'c']
c = [10, 20, 30]
result = list(zip(a, b, c))
print(result)
```

```
Output: [(1, 'a', 10), (2, 'b', 20), (3, 'c', 30)]
```

Example 3: Unequal Lengths

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

```
Output: [(1, 'a'), (2, 'b')]
```

Example 4: Unzip with zip(*)

```
pairs = [(1, 'a'), (2, 'b'), (3, 'c')]
nums, letters = zip(*pairs)
print(f'Nums: {list(nums)}')
print(f'Letters: {list(letters)}')
```

```
Output: Nums: [1, 2, 3]
Letters: ['a', 'b', 'c']
```

Example 5: Dictionary Creation

```
keys = ['name', 'age', 'city']
values = ['Alice', 25, 'NYC']
person = dict(zip(keys, values))
print(person)
```

```
Output: {'name': 'Alice', 'age': 25, 'city': 'NYC'}
```

Example 6: Loop Through Zip

```
fruits = ['apple', 'banana', 'cherry']
prices = [1.50, 0.75, 2.00]
for fruit, price in zip(fruits, prices):
    print(f'{fruit}: ${price}')
```

```
Output: apple: $1.5
banana: $0.75
cherry: $2.0
```

Example 7: Parallel Iteration

```
students = ['Alice', 'Bob', 'Charlie']
scores = [85, 90, 78]
for student, score in zip(students, scores):
```

```
    print(f'{student}: {score}')
```

```
Output: Alice: 85
Bob: 90
Charlie: 78
```

Example 8: Matrix Transpose

```
matrix = [[1, 2, 3], [4, 5, 6]]
transposed = list(zip(*matrix))
print(transposed)
```

```
Output: [(1, 4), (2, 5), (3, 6)]
```

Example 9: Zip with Enumerate

```
items = ['a', 'b', 'c']
values = [1, 2, 3]
for i, (item, value) in enumerate(zip(items, values)):
    print(f'{i}: {item}={value}')
```

```
Output: 0: a=1
1: b=2
2: c=3
```

Example 10: Finding Pairs

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
sums = [a + b for a, b in zip(list1, list2)]
print(sums)
```

```
Output: [6, 8, 10, 12]
```

Practice Exercises

Exercise 1: Create a list of 10 numbers and find the sum of even numbers only.

Exercise 2: Given two lists of equal length, create a dictionary using zip().

Exercise 3: Remove all duplicate elements from a list while maintaining order.

Exercise 4: Find the second largest number in a list without sorting.

Exercise 5: Create a nested list (3x3 matrix) and print the diagonal elements.

Exercise 6: Use list comprehension to create a list of squares of even numbers from 1-20.

Exercise 7: Reverse a list without using reverse(), reversed(), or slicing.

Exercise 8: Merge two sorted lists into one sorted list.

Exercise 9: Count the frequency of each element in a list and store in a dictionary.

Exercise 10: Implement a function to rotate a list by n positions.

Quick Reference

Operation	Syntax/Method	Example
Create	[]	[1, 2, 3]
Access	list[index]	lst[0]
Slice	[start:stop:step]	lst[1:5]
Append	append()	lst.append(4)
Extend	extend()	lst.extend([4, 5])
Insert	insert()	lst.insert(0, 10)
Remove	remove()	lst.remove(3)
Pop	pop()	lst.pop()
Delete	del	del lst[0]
Clear	clear()	lst.clear()
Length	len()	len(lst)
Sort	sort()	lst.sort()
Reverse	reverse()	lst.reverse()
Count	count()	lst.count(2)
Index	index()	lst.index(3)
Copy	copy()	lst.copy()
Comprehension	[expr for x in lst]	[x*2 for x in lst]
Zip	zip()	zip(lst1, lst2)

Table 7: Python Lists Quick Reference

End of Notes

Prepared by: Bikkad IT Institute

For: Python Beginner Level Students

Date: February 2026