

Python Collections and Advanced List Guide

1. Introduction to List in Python

Lists are one of the most commonly used data structures in Python. They are flexible, easy to use, and can hold multiple values of varying data types.

Properties of Lists

- **Ordered:** Elements are stored in a specific order.
- **Mutable:** Can be changed after creation.
- **Allows Duplicates:** Elements can repeat.
- **Allows Multiple Data Types:** Numbers, strings, objects, etc.
- **Indexable:** Access using index (starting from 0).
- **Dynamic Size:** Can grow or shrink.
- **Heterogeneous:** Can store different data types in one list.

List Features / Methods

- `append()` : Add item to end of list
 - `insert()` : Add item to specific index
 - `remove()` : Remove first occurrence of a value
 - `pop()` : Remove and return an element at a given index
 - `sort()` : Sort items of list
 - `reverse()` : Reverse the order of items
 - `extend()` : Add elements from another list
 - `index()` : Get index of a value
 - `count()` : Count occurrences of a value
-

2. Applications of Lists

- Storing collections of data like names, numbers.
 - Dynamic datasets like shopping carts.
 - Used to implement stacks or queues.
 - Matrix (2D list) or table data representation.
 - Good for prototyping.
-

3. Practice Examples

Easy Level (5 Examples)

```
# 1. Create a list and print it
fruits = ["apple", "banana", "cherry"]
print(fruits)

# 2. Add an item to the list
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)

# 3. Access the first and last element
letters = ['a', 'b', 'c', 'd']
print(letters[0], letters[-1])

# 4. Remove an element
colors = ['red', 'green', 'blue']
colors.remove('green')
print(colors)

# 5. Count occurrences of an element
votes = ['yes', 'no', 'yes', 'yes']
print(votes.count('yes'))
```

Moderate Level (5 Examples)

```
# 6. Insert item at specific index
animals = ['dog', 'cat', 'rabbit']
animals.insert(1, 'parrot')
print(animals)

# 7. Reverse the list
nums = [5, 4, 3, 2, 1]
nums.reverse()
print(nums)

# 8. Sort a list of numbers
grades = [88, 92, 75, 89, 94]
grades.sort()
print(grades)

# 9. Merge two lists
list1 = ['a', 'b']
```

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

# 10. Replace an element
languages = ['Java', 'Python', 'C++']
languages[2] = 'JavaScript'
print(languages)
```

Medium Level (5 Examples)

```
# 11. Create and flatten a nested list
nested_list = [[1, 2], [3, 4], [5, 6]]
flat_list = [num for sub in nested_list for num in sub]
print(flat_list)

# 12. Filter even numbers using list comprehension
nums = [1, 2, 3, 4, 5, 6]
evens = [n for n in nums if n % 2 == 0]
print(evens)

# 13. Find second-largest number in a list
numbers = [3, 7, 2, 9, 6]
numbers.remove(max(numbers)) # remove max
print(max(numbers))

# 14. Find duplicates in a list
items = [1, 2, 2, 3, 4, 4, 4]
duplicates = list(set([x for x in items if items.count(x) > 1]))
print(duplicates)

# 15. Swap first and last element in list
names = ['Alice', 'Bob', 'Charlie', 'David']
names[0], names[-1] = names[-1], names[0]
print(names)
```

4. Advanced List Concepts

Lists with Complex Data Structures

```
students = [
    {"name": "Alice", "age": 22},
    {"name": "Bob", "age": 25},
```

```

        {"name": "Charlie", "age": 23}
    ]
    names = [student['name'] for student in students]
    print(names)

```

map(), filter(), reduce() with Lists

```

from functools import reduce
numbers = [1, 2, 3, 4, 5]

# map: square each number
squares = list(map(lambda x: x**2, numbers))

# filter: get even numbers
evens = list(filter(lambda x: x % 2 == 0, numbers))

# reduce: calculate product
product = reduce(lambda x, y: x * y, numbers)

print(squares, evens, product)

```

Nested List / Matrix

```

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

# Transpose the matrix
transpose = [[row[i] for row in matrix] for i in range(len(matrix[0]))]
print(transpose)

```

5. Difference between List, Set, Tuple, and Dictionary

Feature	List	Set	Tuple	Dictionary
Ordered	✓ Yes	✗ No	✓ Yes	✓ Yes (Python 3.7+)
Mutable	✓ Yes	✓ Yes	✗ No	✓ Yes
Allows Duplicates	✓ Yes	✗ No	✓ Yes	Keys: ✗ No, Values: ✓ Yes

Feature	List	Set	Tuple	Dictionary
Syntax	[]	{ }	()	{ key: value }
Access via Index	✓ Yes	✗ No	✓ Yes	✗ No (accessed via keys)
Use Case	General-purpose	Unique items	Fixed collection	Mapping key-value pairs

6. Advanced List Challenges

Rotate a List by N Positions

```
def rotate(lst, n):
    return lst[n:] + lst[:n]

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

Remove Duplicate Elements but Keep Order

```
def unique_order(lst):
    seen = set()
    return [x for x in lst if not (x in seen or seen.add(x))]

print(unique_order([1, 2, 2, 3, 4, 4, 1]))
```

Intersection of Two Lists

```
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
intersection = [x for x in list1 if x in list2]
print(intersection)
```

Flatten a Deeply Nested List

```
def flatten(lst):
    flat_list = []
    for item in lst:
        if isinstance(item, list):
            flat_list.extend(flatten(item))
        else:
            flat_list.append(item)
```

```
        return flat_list

print(flatten([1, [2, [3, 4], [5, 6]], 7]))
```

Group List into Sublists of N Size

```
def group(lst, n):
    return [lst[i:i + n] for i in range(0, len(lst), n)]

print(group([1, 2, 3, 4, 5, 6, 7], 3))
```

Feel free to revisit and practice the examples regularly to strengthen your knowledge of Python lists and other collection types!