Python Lists and Tuples - Complete Revision Notes

Table of Contents

Lists

Definition

A **List** is a built-in data type in Python that stores a collection of items in a specific order. Lists are:

- **Mutable** (can be changed after creation)
- **Ordered** (items have a defined order)
- **Allow duplicates**
- **Indexed** (items can be accessed by index)

Syntax

```
# Empty list
empty_list = []
empty_list = list()

# List with elements
my_list = [1, 2, 3, 'hello', True, 3.14]
mixed_list = [1, 'string', [1, 2, 3], {'key': 'value'}]
```

Tuples

Definition

A **Tuple** is a built-in data type in Python that stores a collection of items in a specific order. Tuples are:

- **Immutable** (cannot be changed after creation)
- **Ordered** (items have a defined order)
- **Allow duplicates**
- **Indexed** (items can be accessed by index)

Syntax

```
# Empty tuple
empty_tuple = ()
empty_tuple = tuple()

# Tuple with elements
my_tuple = (1, 2, 3, 'hello', True, 3.14)
single_element = (5,)  # Note: comma is required for single element
mixed_tuple = (1, 'string', [1, 2, 3], {'key': 'value'})
```

List Operations and Methods

1. Creating Lists

```
# Different ways to create lists
list1 = [1, 2, 3, 4, 5]
list2 = list(range(1, 6))  # [1, 2, 3, 4, 5]
list3 = [x for x in range(1, 6)]  # List comprehension
list4 = [0] * 5  # [0, 0, 0, 0, 0]
```

2. Accessing Elements

```
my_list = ['a', 'b', 'c', 'd', 'e']

# Positive indexing
print(my_list[0])    # 'a'
print(my_list[2])    # 'c'

# Negative indexing
print(my_list[-1])   # 'e'
print(my_list[-2])   # 'd'

# Slicing
print(my_list[1:4])  # ['b', 'c', 'd']
print(my_list[:3])   # ['a', 'b', 'c']
print(my_list[2:])   # ['c', 'd', 'e']
print(my_list[::2])  # ['a', 'c', 'e'] (step=2)
```

3. Modifying Lists

Adding Elements

```
my_list = [1, 2, 3]

# append() - adds single element at the end
my_list.append(4)          # [1, 2, 3, 4]

# insert() - adds element at specific position
```

```
my_list.insert(1, 'new')   # [1, 'new', 2, 3, 4]

# extend() - adds multiple elements
my_list.extend([5, 6])     # [1, 'new', 2, 3, 4, 5, 6]

# Using + operator
my_list = my_list + [7, 8] # [1, 'new', 2, 3, 4, 5, 6, 7, 8]

# Using += operator
my_list += [9, 10]         # [1, 'new', 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Removing Elements

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

# remove() - removes first occurrence
my_list.remove(2)     # [1, 3, 2, 4, 5]

# pop() - removes and returns element (default: last)
last_item = my_list.pop()      # Returns 5, list: [1, 3, 2, 4]
second_item = my_list.pop(1)   # Returns 3, list: [1, 2, 4]

# del statement
del my_list[0]         # [2, 4]
del my_list[0:2]       # [] (deletes slice)

# clear() - removes all elements
my_list.clear()        # []
```

Modifying Elements

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

# Change single element
my_list[0] = 'first'  # ['first', 2, 3, 4, 5]

# Change multiple elements
my_list[1:3] = ['a', 'b', 'c']  # ['first', 'a', 'b', 'c', 4, 5]
```

4. List Methods and Functions

Search and Count

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

# index() - returns first index of element
print(my_list.index(2))     # 1
print(my_list.index(2, 2))  # 3 (search from index 2)

# count() - counts occurrences
```

```
print(my_list.count(2))     # 3

# in operator - check membership
print(2 in my_list)         # True
print(6 in my_list)         # False
```

Sorting and Reversing

```
my_list = [3, 1, 4, 1, 5, 9, 2, 6]

# sort() - sorts in place
my_list.sort()              # [1, 1, 2, 3, 4, 5, 6, 9]
my_list.sort(reverse=True)  # [9, 6, 5, 4, 3, 2, 1, 1]

# sorted() - returns new sorted list
original = [3, 1, 4, 1, 5]
new_sorted = sorted(original)  # original unchanged

# reverse() - reverses in place
my_list.reverse()           # [1, 1, 2, 3, 4, 5, 6, 9]

# Custom sorting
words = ['apple', 'pie', 'washington', 'book']
words.sort(key=len)         # ['pie', 'book', 'apple', 'washington']
```

Other Useful Functions

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

# len() - length of list
print(len(my_list))    # 5

# sum() - sum of numeric elements
print(sum(my_list))    # 15

# min() and max()
print(min(my_list))    # 1
print(max(my_list))    # 5

# all() and any()
print(all([True, True, False]))   # False
print(any([True, False, False]))  # True

# enumerate() - get index and value
for i, value in enumerate(my_list):
    print(f"Index {i}: {value}")
```

5. List Comprehensions

```
# Basic syntax: [expression for item in iterable]
squares = [x**2 for x in range(1, 6)]  # [1, 4, 9, 16, 25]

# With condition
even_squares = [x**2 for x in range(1, 11) if x % 2 == 0]  # [4, 16, 36, 64, 100]

# Nested list comprehension
matrix = [[i*j for j in range(1, 4)] for i in range(1, 4)]
# [[1, 2, 3], [2, 4, 6], [3, 6, 9]]

# With if-else
result = [x if x > 0 else 0 for x in [-1, 2, -3, 4]]  # [0, 2, 0, 4]
```

Tuple Operations

1. Creating Tuples

```
# Different ways to create tuples
tuple1 = (1, 2, 3, 4, 5)
tuple2 = tuple([1, 2, 3, 4, 5])
tuple3 = 1, 2, 3, 4, 5  # Parentheses optional
single = (5,)  # Single element tuple
empty = ()     # Empty tuple
```

2. Accessing Elements

```
my_tuple = ('a', 'b', 'c', 'd', 'e')

# Same as lists - indexing and slicing
print(my_tuple[0])     # 'a'
print(my_tuple[-1])    # 'e'
print(my_tuple[1:4])   # ('b', 'c', 'd')
```

3. Tuple Methods

```
my_tuple = (1, 2, 3, 2, 4, 2, 5)

# count() - counts occurrences
print(my_tuple.count(2))  # 3

# index() - returns first index
print(my_tuple.index(2))  # 1

# len() - length
print(len(my_tuple))      # 7
```

```python
# in operator - membership
print(2 in my_tuple)      # True
```

4. Tuple Unpacking

```python
# Basic unpacking
point = (3, 4)
x, y = point
print(x, y)  # 3 4

# Multiple assignment
a, b, c = (1, 2, 3)

# Swapping variables
a, b = b, a

# Extended unpacking (Python 3+)
first, *middle, last = (1, 2, 3, 4, 5)
print(first)   # 1
print(middle)  # [2, 3, 4]
print(last)    # 5
```

Comparison: Lists vs Tuples

Feature	Lists	Tuples
Mutability	Mutable	Immutable
Syntax	`[1, 2, 3]`	`(1, 2, 3)`
Performance	Slower	Faster
Memory Usage	More	Less
Use Case	Dynamic data	Fixed data
Methods	Many methods	Few methods
Hashable	No	Yes (if contains hashable elements)

Performance Comparison

```python
import timeit

# Creation time
list_time = timeit.timeit(lambda: [1, 2, 3, 4, 5], number=1000000)
tuple_time = timeit.timeit(lambda: (1, 2, 3, 4, 5), number=1000000)

print(f"List creation: {list_time}")
print(f"Tuple creation: {tuple_time}")
# Tuples are generally faster to create
```

Advanced Concepts

1. Nested Lists and Tuples

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

# Nested tuple
nested_tuple = ((1, 2), (3, 4), (5, 6))
print(nested_tuple[0][1])  # 2

# Mixed nesting
mixed = [(1, 2), [3, 4], (5, [6, 7])]
```

2. Copying Lists

```
original = [1, 2, [3, 4]]

# Shallow copy
copy1 = original.copy()
copy2 = original[:]
copy3 = list(original)

# Deep copy
import copy
deep_copy = copy.deepcopy(original)

# Demonstrate difference
original[2][0] = 'changed'
print(copy1)     # [1, 2, ['changed', 4]] - shallow copy affected
print(deep_copy) # [1, 2, [3, 4]] - deep copy unaffected
```

3. List as Stack and Queue

```
# Stack (LIFO) - use append() and pop()
stack = []
stack.append(1)
stack.append(2)
stack.append(3)
print(stack.pop())  # 3

# Queue (FIFO) - use collections.deque for efficiency
from collections import deque
queue = deque([1, 2, 3])
queue.append(4)
print(queue.popleft())  # 1
```

Tricky Interview Questions

1. What happens when you try to modify a tuple?

```python
# Question: What's the output?
t = (1, 2, [3, 4])
t[2].append(5)
print(t)

# Answer: (1, 2, [3, 4, 5])
# Explanation: Tuple is immutable, but it can contain mutable objects
```

2. List multiplication gotcha

```python
# Question: What's the output?
matrix = [[0] * 3] * 3
matrix[0][0] = 1
print(matrix)

# Answer: [[1, 0, 0], [1, 0, 0], [1, 0, 0]]
# Explanation: All rows reference the same list object
# Correct way: matrix = [[0] * 3 for _ in range(3)]
```

3. Default mutable arguments

```python
# Question: What's wrong with this function?
def add_item(item, target_list=[]):
    target_list.append(item)
    return target_list

print(add_item(1))  # [1]
print(add_item(2))  # [1, 2] - Unexpected!

# Correct way:
def add_item(item, target_list=None):
    if target_list is None:
        target_list = []
    target_list.append(item)
    return target_list
```

4. List comprehension variable scope

```python
# Question: What's the output?
x = 'global'
result = [x for x in range(3)]
print(x)

# Answer: 'global' (in Python 3)
# Explanation: List comprehensions have their own scope in Python 3
```

5. Late binding closures

```python
# Question: What's the output?
functions = []
for i in range(3):
    functions.append(lambda: i)

for f in functions:
    print(f())

# Answer: 2, 2, 2
# Explanation: All lambdas capture the same variable 'i'
# Solution: functions.append(lambda x=i: x)
```

6. Equality vs Identity

```python
# Question: Explain the difference
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a == b)   # True (equality)
print(a is b)   # False (different objects)
print(a is c)   # True (same object)
```

7. Tuple immutability exception

```python
# Question: Will this work?
t = ([1, 2], [3, 4])
t[0].append(3)
print(t)  # ([1, 2, 3], [3, 4]) - Yes, it works!

# But this won't:
# t[0] = [1, 2, 3]  # TypeError
```

8. List slicing edge cases

```python
# Question: What are the outputs?
lst = [1, 2, 3, 4, 5]
print(lst[10:])     # [] (no error)
print(lst[:-10])    # [] (no error)
print(lst[2:1])     # [] (empty slice)
print(lst[::-1])    # [5, 4, 3, 2, 1] (reverse)
```

9. Memory efficiency question

```python
# Question: Which is more memory efficient for storing coordinates?
# Option A: points = [(1, 2), (3, 4), (5, 6)]
# Option B: points = [[1, 2], [3, 4], [5, 6]]
```

```python
# Answer: Option A (tuples)
# Tuples are more memory efficient than lists
```

10. List methods return values

```python
# Question: What's the output?
lst = [3, 1, 4, 1, 5]
result = lst.sort()
print(result)  # None
print(lst)     # [1, 1, 3, 4, 5]

# Most list methods modify in-place and return None
# vs sorted() which returns a new list
```

Quick Reference Cheat Sheet

List Methods Summary

- `append(x)` - Add item to end
- `extend(iterable)` - Add all items from iterable
- `insert(i, x)` - Insert item at position
- `remove(x)` - Remove first occurrence
- `pop([i])` - Remove and return item
- `clear()` - Remove all items
- `index(x)` - Find index of first occurrence
- `count(x)` - Count occurrences
- `sort()` - Sort in place
- `reverse()` - Reverse in place
- `copy()` - Shallow copy

Common Patterns

```python
# Check if list is empty
if not my_list:  # Preferred over len(my_list) == 0

# Flatten nested list
flat = [item for sublist in nested_list for item in sublist]

# Remove duplicates (preserving order)
unique = list(dict.fromkeys(my_list))

# Find common elements
common = list(set(list1) & set(list2))

# List rotation
rotated = my_list[n:] + my_list[:n]
```

---

*These notes cover the essential concepts of Python Lists and Tuples. Practice these operations and review the tricky questions regularly for interview prepa