

Python Data Structures: Lists, Tuples & Dictionaries

Class XI Computer Science Notes

Table of Contents

1. Lists
 2. Tuples
 3. Dictionaries
 4. Quick Reference
-

LISTS

What is a List?

A **list** is a standard data type in Python that can store a sequence of values of any type. Lists are **mutable** (changeable) - you can modify elements after creation.

Creating Lists

```
python

[]          # Empty list
[1, 2, 3]    # List of integers
['a', 1, 'b', 2] # List of mixed values
```

List Indexing

Lists use **two-way indexing**:

```
List: ['P', 'Y', 'T', 'H', 'O', 'N']
Index: 0  1  2  3  4  5   (positive)
Index: -6 -5 -4 -3 -2 -1  (negative)
```

List Operators

1. Joining Lists (+)

```
python
```

```
l1 = [1, 2, 3]
l2 = [4, 5]
result = l1 + l2  # [1, 2, 3, 4, 5]
```

2. List Replication (*)

```
python

l1 = [1, 2]
result = l1 * 2  # [1, 2, 1, 2]
```

3. Membership Operators (in, not in)

```
python

my_list = [1, 2, 3, 4]
print(3 in my_list)    # True
print(5 not in my_list) # True
```

4. Comparison Operators

All relational operators work: `>`, `<`, `>=`, `<=`, `==`, `!=`

5. List Slicing

```
python

my_list = [10, 12, 14, 20, 22, 24, 30, 32, 34]
subset = my_list[3:-3]  # [20, 22, 24]
```

Essential List Methods

append() - Add single item to end

```
python

my_list = [1, 2, 3]
my_list.append(4)  # [1, 2, 3, 4]
```

extend() - Add multiple elements

```
python

my_list = [1, 2]
my_list.extend([3, 4])  # [1, 2, 3, 4]
```

insert() - Insert at specific position

```
python  
  
my_list = [1, 3, 4]  
my_list.insert(1, 2)    # [1, 2, 3, 4]
```

index() - Find index of item

```
python  
  
my_list = [1, 2, 3]  
position = my_list.index(2) # Returns 1
```

pop() - Remove item by index

```
python  
  
my_list = [1, 2, 3]  
removed = my_list.pop(1)    # Removes and returns 2
```

remove() - Remove first occurrence of value

```
python  
  
my_list = [1, 2, 3, 2]  
my_list.remove(2)           # [1, 3, 2]
```

clear() - Remove all items

```
python  
  
my_list = [1, 2, 3]  
my_list.clear()             # []
```

count() - Count occurrences

```
python  
  
my_list = [1, 2, 2, 3]  
count = my_list.count(2)    # Returns 2
```

reverse() - Reverse the list

```
python
```

```
my_list = [1, 2, 3]
my_list.reverse()      # [3, 2, 1]
```

sort() - Sort in ascending order

```
python

my_list = [3, 1, 2]
my_list.sort()        # [1, 2, 3]
```

List Manipulation

Updating Elements

```
python

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

Deleting Elements

```
python

my_list = [10, 12, 13, 14]
del my_list[2]         # [10, 12, 14]
```

TUPLES

What is a Tuple?

A **tuple** is an ordered collection of items that is **immutable** (unchangeable). Tuples are created using round brackets `()`.

Creating Tuples

```
python

()          # Empty tuple
(3,)        # Single element tuple (note the comma!)
(1, 2, 3)    # Tuple of integers
(1, 2.5, 'a') # Mixed data types
```

Tuple Methods

len() - Get length

```
python
```

```
my_tuple = (1, 2, 3)  
length = len(my_tuple)    # Returns 3
```

max() - Find maximum value

```
python
```

```
my_tuple = (10, 5, 78)  
maximum = max(my_tuple)   # Returns 78
```

min() - Find minimum value

```
python
```

```
my_tuple = (9, 3, 1)  
minimum = min(my_tuple)   # Returns 1
```

index() - Find index of element

```
python
```

```
my_tuple = (3, 4, 5, 6)  
position = my_tuple.index(5) # Returns 2
```

count() - Count occurrences

```
python
```

```
my_tuple = (2, 3, 4, 5, 6, 2, 7, 8, 2)  
count = my_tuple.count(2)  # Returns 3
```

tuple() - Create tuple from other types

```
python
```

```
my_tuple = tuple("abc")    # ('a', 'b', 'c')
```

Tuple Operations

Traversing a Tuple

```
python
```

```
my_tuple = ('p', 'u', 'r', 'e')
for element in my_tuple:
    print(element)
```

Joining Tuples

```
python

t1 = (1, 2, 3)
t2 = (4, 5)
result = t1 + t2      # (1, 2, 3, 4, 5)
```

Replicating Tuples

```
python

my_tuple = (1, 2)
result = my_tuple * 2    # (1, 2, 1, 2)
```

Slicing Tuples

```
python

my_tuple = (10, 12, 14, 20, 22, 24, 30, 32, 34)
subset = my_tuple[3:-3]    # (20, 22, 24)
```

DICTIONARIES

What is a Dictionary?

A **dictionary** is a collection of **key-value pairs**. Dictionaries are **mutable** and **unordered** collections where each key is unique.

Creating Dictionaries

```
python

# Syntax: {key: value, key: value}
emp = {'salary': 10000, 'age': 24, 'name': 'john'}
```

Dictionary Methods

len() - Get number of key-value pairs

python

```
emp = {'salary': 10000, 'age': 24, 'name': 'john'}  
length = len(emp)      # Returns 3
```

clear() - Remove all items

python

```
emp = {'salary': 10000, 'age': 24}  
emp.clear()      # {}
```

get() - Get value by key

python

```
emp = {'salary': 10000, 'age': 24, 'name': 'john'}  
age = emp.get('age')    # Returns 24
```

keys() - Get all keys

python

```
emp = {'salary': 10000, 'age': 24, 'name': 'john'}  
keys = emp.keys()      # ['salary', 'age', 'name']
```

values() - Get all values

python

```
emp = {'salary': 10000, 'age': 24, 'name': 'john'}  
values = emp.values()   # [10000, 24, 'john']
```

items() - Get all key-value pairs

python

```
emp = {'salary': 10000, 'age': 24, 'name': 'john'}  
items = emp.items()  
for key, value in items:  
    print(f"{key}: {value}")
```

update() - Merge dictionaries

python

```
emp1 = {'salary': 10000, 'age': 24, 'name': 'john'}
emp2 = {'salary': 20000, 'age': 25, 'name': 'riya'}
emp1.update(emp2)      # emp1 becomes emp2's values
```

Dictionary Operations

Traversing a Dictionary

```
python

emp = {'salary': 10000, 'age': 24, 'name': 'john'}
for key in emp:
    print(f"{key}: {emp[key]}")
```

Adding Elements

```
python

emp = {'salary': 10000, 'age': 24, 'name': 'john'}
emp['dept'] = 'sales'    # Adds new key-value pair
```

Deleting Elements

Method 1: Using del

```
python

emp = {'salary': 10000, 'age': 24, 'name': 'john'}
del emp['age']           # Removes 'age' key-value pair
```

Method 2: Using pop()

```
python

emp = {'salary': 10000, 'age': 24, 'name': 'john'}
removed_value = emp.pop('age') # Removes and returns 24
```









Checking Key Existence

```
python

emp = {'salary': 10000, 'name': 'john'}
print('age' in emp)      # False
print('salary' not in emp) # False
```


Quick Reference

When to Use Each Data Structure

Feature	List	Tuple	Dictionary
Mutability	Mutable 	Immutable 	Mutable 
Ordered	Yes 	Yes 	No 
Indexed	Numeric	Numeric	Key-based
Duplicates	Allowed 	Allowed 	Keys: No, Values: Yes
Use Case	Dynamic collections	Fixed collections	Key-value mapping

Memory Map Summary

List Functions: `index()`, `append()`, `extend()`, `insert()`, `pop()`, `remove()`, `clear()`, `count()`, `reverse()`, `sort()`

Tuple Functions: `len()`, `max()`, `min()`, `index()`, `count()`, `tuple()`

Dictionary Functions: `len()`, `clear()`, `keys()`, `get()`, `values()`, `items()`, `update()`

Key Points to Remember

1. **Lists** are mutable and use square brackets `[]`
2. **Tuples** are immutable and use round brackets `()`
3. **Dictionaries** are mutable key-value collections using curly brackets `{}`
4. All three support membership operators `in`, `not in`
5. Lists and tuples support slicing and indexing
6. Only dictionaries use keys instead of numeric indices

These notes cover all essential concepts for Class XI Computer Science curriculum on Python data structures.