Python Data Structures: Lists, Tuples, Sets, Dictionaries

Martin Wafula
Multimedia University of Kenya

Outline

Introduction to Python Data Structures

Core Python Data Types

- **Numbers**: int, float, complex
- **Strings**: Immutable sequences of characters
- **Lists**: Mutable ordered sequences
- **Tuples**: Immutable ordered sequences
- **Sets**: Unordered collections of unique elements
- **Dictionaries**: Key-value mappings
- **Files**: For input/output operations

Lists

List Basics

- Ordered, mutable sequences
- Can contain mixed data types
- Created with square brackets `[]`

Example

```
numbers = [1, 2, 3, 4, 5, 6]
names = ["Jordan", "Ray", "Maisie"]
mixed = [1, "cat", 7.8]
```

List Indexing and Slicing

Similar to strings

```
names = ["Jordan", "Ray", "Maisie"]
names[1]    # "Ray"
names[-2]   # "Ray"
names[:2]   # ["Jordan", "Ray"]
names[1:]   # ["Ray", "Maisie"]
```

Negative indices

```
c = [-45, 6, 0, 72, 1543]
c[-1]  # 1543 (last element)
c[-5]  # -45 (first element)
```

List Mutability

- Lists can be modified after creation
- Contrast with immutable strings

Example

```
animals = ['cat', 'dog', 'pig']
animals[2] = 'rabbit'  # Valid
print(animals)  # ['cat', 'dog', 'rabbit']

# Strings are immutable
animal = 'bat'
animal[1] = 'd'  # TypeError
```

Common List Methods

Modification

- `append(item)`
- `insert(index, item)`
- `remove(item)`
- `pop([index])`
- `extend(iterable)`

Information

- `count(item)`
- `index(item)`
- `sort()`
- `reverse()`

Example

```
nums = [3, 1, 4, 1, 5]
nums.sort()  # [1, 1, 3, 4, 5]
nums.count(1)  # 2
nums.extend([9, 2])  # [1, 1, 3, 4, 5, 9, 2]
```

Important!

Simple assignment creates a reference, not a copy

Example

```
animals = ['cat', 'dog', 'pig']
animals2 = animals  # Reference
animals.remove('dog')
print(animals2)  # ['cat', 'pig']

# Proper copy methods
backup = animals.copy()  # Method 1
backup = animals[:]      # Method 2
backup = list(animals)  # Method 3
```

Multi-dimensional Lists

- Lists can contain other lists
- Useful for matrices and tabular data

Example

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

# Heterogeneous and irregular
data = [[1, 'a'], [2, 'b', True], ['x']]

# Creating a 3x4 2D list initialized to 0
rows, cols = 3, 4
matrix = [[0 for j in range(cols)] for i in range(rows)]
```

List Comprehensions

- Concise way to create and transform lists
- More readable and often faster than loops

Example

```
# Squares of numbers 0-9
squares = [x**2 for x in range(10)]

# Even numbers only
evens = [x for x in range(10) if x % 2 == 0]

# Convert strings to uppercase
colors = ['red', 'orange', 'yellow']
upper_colors = [color.upper() for color in colors]
```

Stack Simulation with Lists

- Lists can simulate stacks (LIFO)
- Use `append()` for push
- Use `pop()` for pop

Example

```
stack = []
stack.append('red')    # push
stack.append('green')  # push
print(stack)  # ['red', 'green']
top = stack.pop()      # pop
print(top)    # 'green'
print(stack)  # ['red']
```

Tuples

Tuple Basics

- Immutable ordered sequences
- Created with parentheses `()` or just commas
- Often used for fixed collections of items

Example

```
my_tuple = (1, 2, 3)
my_tuple[1]  # 2
my_tuple[1:3]  # (2, 3)

# Single-element tuple needs comma
single = (1,)
not_a_tuple = (1)  # Just an integer
```

Tuple Immutability

- Cannot modify elements after creation
- But can contain mutable objects

Example

```
my_tuple = ([1, 2], [3, 4])
my_tuple[0] = [5, 6]  # TypeError
my_tuple[0][1] = 7  # Valid
print(my_tuple)  # ([1, 7], [3, 4])

# Empty tuple
empty_tuple = ()
```

Multiple assignment

```
# Packing
t = 1, 2, 3  # (1, 2, 3)

# Unpacking
x, y, z = t
print(y)  # 2

# Swapping values
a, b = 10, 20
a, b = b, a  # Swap
```

Function return values

```
def min_max(nums):
    return min(nums), max(nums)
```

- From `collections` module
- Tuple with named fields

Example

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
print(p.x, p.y)  # 11 22
print(p[0], p[1])  # 11 22 (still works)
```

Sets

- Unordered collections of unique elements
- Created with `set()` or curly braces `{}`
- Optimized for membership testing

Example

```
basket = ['apple', 'orange', 'apple']
fruit = set(basket)  # {'orange', 'apple'}
'orange' in fruit  # True (fast lookup)

# Empty set
empty_set = set()  # Not {}, which is dict
```

Methods

- `add(item)`
- `remove(item)`
- `discard(item)`
- `pop()`
- `clear()`

Example

```
a = {1, 2, 3}
b = {2, 3, 4}
a | b  # {1, 2, 3, 4}
a & b  # {2, 3}
a - b  # {1}
```

Mathematical Operations

- Union (`|` or `union()`)
- Intersection (`&` or `intersection()`)
- Difference (`-` or `difference()`)
- Symmetric diff (^ or `symmetric_difference()`)

Set Comparisons

- Subset and superset relationships
- Equality and inequality

Example

```
a = {1, 2, 3}
b = {1, 2}
c = {1, 2, 3}

b < a   # True (proper subset)
b <= a  # True (subset)
a == c  # True
a != b  # True
a.isdisjoint({4, 5})  # True
```

Set Comprehensions

- Similar to list comprehensions
- Creates a set instead of a list

Example

```
# Unique letters not in 'abc'
letters = {x for x in 'abracadabra'
           if x not in 'abc'}
# {'r', 'd'}

# Squares of numbers 1-5
squares = {x**2 for x in range(1, 6)}
# {1, 4, 9, 16, 25}
```

Frozen Sets

- Immutable version of sets
- Can be used as dictionary keys

Example

```
frozen = frozenset([1, 2, 3])
# frozen.add(4)  # AttributeError

# Valid dictionary key
d = {frozen: "value"}
```

Dictionaries

- Key-value mappings (associative arrays)
- Created with curly braces `{}`
- Keys must be immutable (strings, numbers, tuples)

Example

```
contacts = {
    "Suzy": "413-286-3712",
    "Alison": "972-272-2782"
}
print(contacts["Suzy"])  # "413-286-3712"

# Empty dictionary
empty_dict = {}
```

Dictionary Operations

Modification

- `d[key] = value`
- `update(other_dict)`
- `pop(key)`
- `popitem()`
- `clear()`

Information

- `keys()`
- `values()`
- `items()`
- `get(key, default)`
- `setdefault(key, default)`

Dictionary Access Methods

- Direct access vs `get()` method
- Handling missing keys

Example

```
grades = {'Alice': 85, 'Bob': 92}

# Direct access - raises KeyError if missing
alice_grade = grades['Alice']

# get() method - returns None or default if missing
carol_grade = grades.get('Carol')  # None
carol_grade = grades.get('Carol', 0)  # 0
```

Common patterns

```
# Iterate keys
for name in contacts:
    print(name)

# Iterate key-value pairs
for name, number in contacts.items():
    print(f"{name}: {number}")

# Iterate values
for number in contacts.values():
    print(number)
```

From lists using zip

```
names = ["Suzy", "Alison"]
numbers = ["413-286-3712", "972-272-2782"]
contacts = dict(zip(names, numbers))
```

Dictionary comprehension

```
squares = {x: x*x for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Filtering with comprehension
even_squares = {k: v for k, v in squares.items()
                if k % 2 == 0}
```

Default Dictionaries

- From `collections` module
- Provides default values for missing keys

Example

```
from collections import defaultdict

# Default to 0 for missing keys
word_counts = defaultdict(int)
for word in text.split():
    word_counts[word] += 1

# Default to empty list
graph = defaultdict(list)
graph['A'].append('B')  # No need to initialize
```

Counter Objects

- Specialized dictionary for counting
- From `collections` module

Example

```
from collections import Counter

words = "to be or not to be that is the question"
word_counts = Counter(words.split())
print(word_counts.most_common(2))
# [('to', 2), ('be', 2)]
```

Comparison

When to Use Each Data Structure

	List	Tuple	Dictionary
Mutable	Yes	No	Yes
Ordered	Yes	Yes	No (Python 3.7+ p
Indexed	By position	By position	By key
Use Case	Dynamic collections	Fixed data	Key-value pairs

- **Sets**: When you need uniqueness and fast membership testing
- **Tuples**: For data integrity and fixed collections
- **Dictionaries**: For labeled data and fast lookups

Performance Characteristics

Operation	List	Dictionary/Set
Indexing	O(1)	O(1)
Insertion	O(n)	O(1)
Deletion	O(n)	O(1)
Search	O(n)	O(1)

Exercises

Dictionary-based solution

```
def word_frequency(text):
    word_counts = {}
    for word in text.lower().split():
        if word in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1
    return word_counts

text = "to be or not to be that is the question"
print(word_frequency(text))
```

Dictionary of lists

```
grade_book = {
    'Susan': [92, 85, 100],
    'Eduardo': [83, 95, 79],
    'Azizi': [91, 89, 82]
}

# Calculate and display averages
for name, grades in grade_book.items():
    avg = sum(grades) / len(grades)
    print(f"{name}'s average: {avg:.2f}")
```

Using list for Sieve of Eratosthenes

```
def sieve(limit):
    primes = [True] * (limit + 1)
    primes[0] = primes[1] = False
    for num in range(2, int(limit**0.5) + 1):
        if primes[num]:
            primes[num*num::num] = [False]*len(primes[num*r
    return [i for i, is_prime in enumerate(primes) if is_pr

print(sieve(100))
```

Dictionary with product information

```
inventory = {
    1001: {'name': 'Pen', 'price': 1.99, 'quantity': 50},
    1002: {'name': 'Notebook', 'price': 4.99, 'quantity': 3
}

# Update inventory
def add_stock(product_id, amount):
    inventory[product_id]['quantity'] += amount

# Process sale
def sell(product_id, amount):
    if inventory[product_id]['quantity'] >= amount:
        inventory[product_id]['quantity'] -= amount
```

Summary

- **Lists**: Mutable, ordered sequences - most flexible
- **Tuples**: Immutable sequences - data integrity
- **Sets**: Unique element collections - fast membership
- **Dictionaries**: Key-value mappings - labeled data
- Choose the right structure for your specific needs

Further Learning

- Explore the `collections` module (Counter, defaultdict, etc.)
- Learn about generators and iterators
- Study more advanced comprehensions
- Practice with real-world datasets

Recommended Exercises

- Implement a queue using lists
- Create a dynamic die-rolling visualization
- Build a word anagram finder using dictionaries
- Implement a simple database using nested dictionaries

Questions

Any questions about Python data structures?