Chapter 4: Data Structures in Python

Chapter Objectives

At the end of this chapter, the student will be able to:

- Understand and manipulate lists, tuples, dictionaries, and sets.
- Use indexing and slicing.
- Apply sorting, filtering, and list comprehensions.
- Choose the appropriate data structure depending on the context.

4.1 Lists and Tuples

A list is an ordered, mutable (modifiable), and indexed data structure.

4.1.1 Lists

Creating Lists

```
# List of numbers
numbers = [1, 2, 3, 4, 5]
print(numbers)  # [1 2 3 4 5]
# List of strings
fruits = ["apple", "banana", "orange"]

# Mixed list
mixed = [10, "Python", 3.14, True]
```

Indexing

Indexes start at 0.

```
fruits = ["apple", "banana", "orange"]

print(fruits[0]) # apple
print(fruits[1]) # banana
print(fruits[-1]) # orange (last element)
```

Modifying Lists

```
fruits[1] = 'kiwi'
print(fruits) # ['apple', 'kiwi', 'orange']
```

List Length

```
print(len(numbers))
```

Slicing (Extracting Sub-lists)

Syntax: list[start:end:step]

```
numbers = [0,1,2,3,4,5,6]

print(numbers[1:4])    # [1,2,3]
print(numbers[:3])     # [0,1,2]
print(numbers[::2])    # [0,2,4,6]
print(numbers[::-1])   # reverse
```

List Operations

Concatenation:

```
a = [1,2]
b = [3,4]
print(a + b)  # [1,2,3,4]
```

Repetition

```
print([1,2] * 3)  # [1,2,1,2,1,2]
```

Important List Methods

```
list = [3,1,4]
list.append(5) # add at the end
list.insert(1,10) # insert at position 1
list.remove(3) # remove value 3
list.pop() # remove last element
list.sort() # sorting
list.reverse() # reverse order
```

Important Properties of Lists

1. Lists are ordered

They preserve the insertion order.

```
fruits = ["apple", "banana", "orange"]
print(fruits)
```

2. Access by index

Each element has a position (index).

Indexes start at **0**.

```
print(fruits[0])   # first element
print(fruits[-1])  # last element
```

3. They can contain any data type

```
mixed = [10, 3.14, "Python", True]
```

Even other lists:

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

4. They are mutable

You can:

- Modify
- Add
- Delete elements

```
grades[0] = 14
grades.append(20)
grades.remove(9)
```

Example: Program that calculates the average:

```
grades = [10, 14, 12, 16]
average = sum(grades) / len(grades)
print("Average =", average)
```

4.1.2 Tuples

A tuple is an ordered but immutable data structure.

Tuple Example

Creation

```
coord = (10, 20)
colors = ("red", "green", "blue")
```

Accessing Elements

```
print(coord[0])     # 10
print(colors[-1])   # blue
```

Modification Not Allowed

```
coord[0] = 5   # Error
```

Tuple Use Cases

Coordinates (x,y)

Multiple return values from functions

Constant data

Function Returning Tuple

```
def calculate(a,b):
    return a+b, a*b

sum_value, product = calculate(3,4)
```

4.2 Dictionaries and Sets

4.2.1 Dictionaries

A dictionary stores data as key:value pairs.

Creation

```
student = {"name": "Ali","age": 22,"major": "Computer Science"}
```

Accessing Values

```
print(student["name"])
# Ali

print(student.get("age"))
```

Modify Dictionary

```
student['age'] = 23

student['email'] = 'ali@mail.com'
```

Deleting Values

```
del student['email']

    student.pop('age')
```

Dictionary Iteration

```
for key,value in student.items():

  print(key,':',value)
```

Example: Counting Letters

```
text = 'python'

counter = {}

for letter in text:

  counter[letter] = counter.get(letter,0) + 1

print(counter)  #{'p': 1, 'y': 1, 't': 1, 'h': 1, 'o': 1, 'n': 1}
```

4.2.2 Sets

A set is an unordered collection without duplicates.

A **set** is a collection that is:

- Unordered
- Without duplicates

Creation

```
s = {1,2,3,4}
print(s)
```

Set Operations

Union

A = {1,2,3}
B = {3,4,5}

print(A | B)# A = {1,2,3,4,5}

Intersection

print(A & B)

Result: {3}

4.3 Operations on Data Structures

4.3.1 Sorting

grades = [12,15,10,18]
grades.sort()
print(grades)

4.3.2 Filtering

Example: **even numbers**

```
list = [1,2,3,4,5,6]

even = []

for x in list:
    if x % 2 == 0:
        even.append(x)

print(even)
```

Result: [2,4,6]

4.3.3 List Comprehension

Syntax:

newlist =[expression for element in list]

Example:

```
list = [1,2,3,4]
squares = [x**2 for x in list]
print(squares)  #   [1,4,9,16]
```

List Comprehension with Condition

newlist = [expression for element in list if condition == True]

Example with condition:

```
list = [1,2,3,4]
evens = [x for x in list if x%2==0]

print(evens)
```

Result: [2,4]

Example

Accept only numbers lower than 5:

```
newlist = [x for x in range(10) if x < 5]
```

Exercises

Exercise 1

Create a list containing **5 numbers** and display the **maximum value**.

Solution

```
lst = [10,5,8,12,3]

print(max(lst))
```

Exercise 2

Create a dictionary containing:

- name
- age
- city

Then display each element.

Solution

```
person = {
"name":"Ali",
"age":20,
"city":"Oran"
}

for k,v in person.items():
    print(k,v)
```

Explanation:

- for → loop to iterate through elements
- person.items() → returns keys and values
- k → key
- v → value

Exercise 3

Create a list of the **squares of numbers from 1 to 10**.

Solution

```
squares = [x**2 for x in range(1,11)]

print(squares)
```