



The Islamic University of Gaza
Faculty of Engineering
Computer Engineering Department



Introduction to Computers

LAB #9

TUPLES, SETS, AND DICTIONARIES

Eng. Hashem M. Hejazy

Eng. Mohammed O. AlHabbash

Fall 2026

Introduction

Python provides several built-in data structures to store collections of data. Each has its own characteristics and is suited for specific tasks:

- Tuples are immutable sequences – once created, their elements cannot be changed. They are used when the collection should remain constant throughout the program (e.g., days of the week, coordinates).
- Sets store unordered collections of unique elements. They are extremely efficient for membership testing (`in/not in`) and for mathematical set operations (union, intersection, etc.).
- Dictionaries store key-value pairs. They provide fast lookup of a value given a unique key (like a real-world dictionary). Keys must be hashable (e.g., numbers, strings).

Creating Tuples

A tuple is created by enclosing elements in parentheses `()`. Even a single element must be followed by a comma to distinguish it from a parenthesised expression.

```
t1 = ()                # empty tuple
t2 = (1, 3, 5)          # tuple of three integers
t3 = ("green", "red", "blue") # tuple of strings
t4 = (1, "hello", 3.14) # mixed types
t5 = (10,)              # single-element tuple - note
                        # the comma
```

You can also create a tuple from any sequence (list, string, range) using the `tuple()` constructor:

```
list1 = [2, 4, 6]
t6 = tuple(list1)      # (2, 4, 6)
t7 = tuple("abc")      # ('a', 'b', 'c')
```

Basic Tuple Operations

Because a tuple is a sequence, it supports all common sequence operations:

Assume `t2 = (1, 3, 5)`

Operation	Example	Result
Indexing	<code>t2[0]</code>	1
Negative index	<code>t2[-1]</code>	5
Slicing	<code>t2[1:3]</code>	(3, 5)
Concatenation	<code>(1,2) + (3,4)</code>	(1,2,3,4)
Repetition	<code>(1,2) * 3</code>	(1,2,1,2,1,2)
Membership	<code>3 in (1,2,3)</code>	True
Length	<code>len((1,2,3))</code>	3
Min / Max / Sum	<code>min((5,2,8)),</code> <code>sum((1,2,3))</code>	2, 6

Ex:

```
1 def main(): 1 usage
2     tuple1 = ("green", "red", "blue")
3     tuple2 = (7, 1, 2, 23, 4, 5)
4     print("tuple1:", tuple1)
5     print("tuple2:", tuple2)
6     print("length of tuple2:", len(tuple2))
7     print("max of tuple2:", max(tuple2))
8     print("max of tuple1:", max(tuple1))
9     print("min of tuple2:", min(tuple2))
10    print("min of tuple1:", min(tuple1))
11    print("sum of tuple2:", sum(tuple2))
12    print("first element of tuple2:", tuple2[0])
13    print("last element of tuple2:", tuple2[-1])
14    print("*"*60)
15    tuple3 = tuple1 + tuple2
16    print("tuple1 + tuple2:", tuple3)
17    print("*"*60)
18    tuple4 = 2 * tuple1
19    print("2 * tuple1:", tuple4)
20    print("*"*60)
```

```

21     print("slice tuple2[2:4]:", tuple2[2:4])
22     print("2 in tuple2:", 2 in tuple2)
23     print("*"*60)
24     for element in tuple1:
25         print(element, end=" ")
26     print()
27     print("*"*60)
28
29     # Convert tuple to list, sort, then back to tuple
30     lst = list(tuple2)
31     lst.sort()
32     tuple5 = tuple(lst)
33     print("sorted tuple2:", tuple5)
34
35     # Compare tuples
36     tuple6 = tuple(lst)
37     print("tuple5 == tuple6:", tuple5 == tuple6)
38     main()

```

Output:

```

tuple1: ('green', 'red', 'blue')
tuple2: (7, 1, 2, 23, 4, 5)
length of tuple2: 6
max of tuple2: 23
max of tuple1: red
min of tuple2: 1
min of tuple1: blue
sum of tuple2: 42
first element of tuple2: 7
last element of tuple2: 5
*****
tuple1 + tuple2: ('green', 'red', 'blue', 7, 1, 2, 23, 4, 5)
*****
2 * tuple1: ('green', 'red', 'blue', 'green', 'red', 'blue')
*****
slice tuple2[2:4]: (2, 23)
2 in tuple2: True
green red blue
*****
sorted tuple2: (1, 2, 4, 5, 7, 23)
tuple5 == tuple6: True

```

Notes:

- A tuple is immutable – you cannot add, remove, or replace elements. However, if a tuple contains a mutable object (like a list or a custom object), that object itself can be changed.
- Forgetting the comma in a single-element tuple: `t = (10)` creates an integer, not a tuple. Correct: `t = (10,)`.

Sets

Creating Sets

A set is created by enclosing elements in curly braces `{}` or by using the `set()` constructor.

```
s1 = {1, 3, 5}           # set with three integers
s2 = set()               # empty set - note: {} creates an empty
                        # dictionary!
s3 = set([2, 4, 6, 2])   # from a list - duplicates are removed → {2,
                        # 4, 6}
s4 = set("abracadabra")  # from a string → {'a', 'b', 'c', 'd', 'r'}
s5 = {1, "hello", 3.14}  # mixed types
```

Operation	Method / Operator	Description
Add element	<code>s.add(x)</code>	Inserts <code>x</code> if not already present.
Remove element	<code>s.remove(x)</code>	Removes <code>x</code> ; raises <code>KeyError</code> if missing.
Discard element	<code>s.discard(x)</code>	Removes <code>x</code> if present; no error if missing.
Size	<code>len(s)</code>	Returns number of elements.
Membership	<code>x in s</code>	Fast membership test.
Min / Max / Sum	<code>min(s)</code> , <code>max(s)</code> , <code>sum(s)</code>	Work as expected for numeric sets.
Copy	<code>s.copy()</code>	Returns a shallow copy.
Clear	<code>s.clear()</code>	Removes all elements.

EX:

```
1 s = {1, 2, 4}
2 print("Original set:", s)
3 s.add(6)
4 print("After add(6):", s)
5 s.remove(2)
6 print("After remove(2):", s)
7 print("Length:", len(s))
8 print("Is 4 in s?", 4 in s)
9 print("Max:", max(s))
10 print("Min:", min(s))
11 print("Sum:", sum(s))
12
13 # Traverse
14 for element in s:
15     print(element, end=" ")
16 print()
```

```
"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe"
Original set: {1, 2, 4}
After add(6): {1, 2, 4, 6}
After remove(2): {1, 4, 6}
Length: 3
Is 4 in s? True
Max: 6
Min: 1
Sum: 11
1 4 6

Process finished with exit code 0
```

Subset and Superset

- `s1.issubset(s2)` – True if every element of `s1` is in `s2`.
- `s1.issuperset(s2)` – True if every element of `s2` is in `s1`.

Comparison operators:

- `s1 < s2` – proper subset ($s1 \subsetneq s2$ and $s1 \neq s2$)
- `s1 <= s2` – subset ($s1 \subseteq s2$)
- `s1 > s2` – proper superset
- `s1 >= s2` – superset

Set Operations – Union, Intersection, Difference, Symmetric Difference

Operation	Method	Operator	Description
Union	<code>s1.union(s2)</code>	<code>s1 s2</code>	Elements in either set.
Intersection	<code>s1.intersection(s2)</code>	<code>s1 & s2</code>	Elements in both sets.
Difference	<code>s1.difference(s2)</code>	<code>s1 - s2</code>	Elements in <code>s1</code> but not in <code>s2</code> .
Symmetric difference	<code>s1.symmetric_difference(s2)</code>	<code>s1 ^ s2</code>	Elements in exactly one of the sets (union minus intersection).

```

1  set1 = {"green", "red", "blue"}
2  set2 = {"green", "yellow"}
3
4  print("set1:", set1)
5  print("set2:", set2)
6  print("Union (|) :", set1 | set2)
7  print("Union (union) :", set1.union(set2))
8  print("Intersection (&):", set1 & set2)
9  print("Difference (-) :", set1 - set2)
10 print("Symmetric diff ^:", set1 ^ set2)

```

```

"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe"
set1: {'green', 'blue', 'red'}
set2: {'green', 'yellow'}
Union (|) : {'blue', 'red', 'green', 'yellow'}
Union (union) : {'blue', 'red', 'green', 'yellow'}
Intersection (&): {'green'}
Difference (-) : {'blue', 'red'}
Symmetric diff ^: {'blue', 'red', 'yellow'}

Process finished with exit code 0

```

Dictionaries

Creating Dictionaries

A dictionary stores key-value pairs. Each key must be unique and hashable (immutable). Values can be any object.

```

students = {"111-34-3434": "John", "132-56-6290": "Peter"}
empty_dict = {} # empty dictionary
grades = {"John": 85, "Jane": 92} # keys are strings,
# values integers
mixed = {1: "one", 2: "two"} # keys are integers
d = dict([(1, "one"), (2, "two")]) # from list of tuples
d = dict(one=1, two=2) # using keyword
# arguments (keys become strings)

```

Basic Dictionary Operations

Operation	Syntax / Method	Description
Access value	<code>d[key]</code>	Returns value for key; raises <code>KeyError</code> if missing.
Add / modify item	<code>d[key] = value</code>	If key exists, value is updated; otherwise new item added.
Delete item	<code>del d[key]</code>	Removes the item with that key; raises <code>KeyError</code> if missing.
Number of items	<code>len(d)</code>	Returns number of key-value pairs.

Membership (keys only)	key in d	Tests whether key exists.
Get value safely	d.get(key, default)	Returns value if key exists, else default (or None if no default).
Remove and return value	d.pop(key, default)	Removes key and returns its value; if missing, returns default or raises KeyError.
Remove arbitrary item	d.popitem()	Removes and returns a random (key, value) pair.
Get all keys	d.keys()	Returns a view of keys.
Get all values	d.values()	Returns a view of values.
Get all items	d.items()	Returns a view of (key, value) tuples.
Clear dictionary	d.clear()	Removes all items.

```

1 def main(): 1usage
2     students = {'111-34-3434': 'John', '132-56-6290': 'Peter'}
3     print("Initial dictionary:", students)
4     # Add a new item
5     students["234-56-9010"] = "Susan"
6     print("After adding Susan:", students)
7     # Modify an existing item
8     students["111-34-3434"] = "John Smith"
9     print("After modifying John:", students)
10    # Retrieve a value
11    print("Value for key '111-34-3434':", students["111-34-3434"])
12    # Safe retrieval using get
13    print("get('999-99-9999'):", students.get("999-99-9999"))
14    print("get('999-99-9999', 'Not found'):", students.get("999-99-9999", "Not found"))
15    # Delete an item
16    del students["132-56-6290"]
17    print("After deleting Peter:", students)
18    # Keys, values, items
19    print("Keys:", list(students.keys()))
20    print("Values:", list(students.values()))
21    print("Items:", list(students.items()))
22    # Length
23    print("Number of students:", len(students))
24    # Membership test
25    print("Is '111-34-3434' a key?", "111-34-3434" in students)
26    # Loop through keys
27    for key in students:
28        print(key, "->", students[key])
29    main()

```

```

"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe" "C:\Users\
Initial dictionary: {'111-34-3434': 'John', '132-56-6290': 'Peter'}
After adding Susan: {'111-34-3434': 'John', '132-56-6290': 'Peter', '234-56-90
After modifying John: {'111-34-3434': 'John Smith', '132-56-6290': 'Peter', '2
Value for key '111-34-3434': John Smith
get('999-99-9999'): None
get('999-99-9999', 'Not found'): Not found
After deleting Peter: {'111-34-3434': 'John Smith', '234-56-9010': 'Susan'}
Keys: ['111-34-3434', '234-56-9010']
Values: ['John Smith', 'Susan']
Items: [('111-34-3434', 'John Smith'), ('234-56-9010', 'Susan')]
Number of students: 2
Is '111-34-3434' a key? True
111-34-3434 -> John Smith
234-56-9010 -> Susan

Process finished with exit code 0

```

Notes:

- Two dictionaries are equal (==) if they contain the same key-value pairs (order does not matter).
- Comparison operators (<, >, etc.) are not supported (they raise `TypeError`).
- Keys must be hashable. Lists, dictionaries, and sets cannot be used as keys because they are mutable and not hashable. Tuples and strings are fine.
- Keys must be hashable. Lists, dictionaries, and sets cannot be used as keys because they are mutable and not hashable. Tuples and strings are fine.
- The `in` operator checks keys only, not values.

```
1 d1 = {"red": 41, "blue": 3}
2 d2 = {"blue": 3, "red": 41}
3 print(d1 == d2)
4 print(d1 != d2)
```

Output: True, False

Process finished with exit code 1

Quick Comparison: List, Tuple, Set, and Dictionary

Feature	List	Tuple	Set	Dictionary
Syntax	[1, 2, 3]	(1, 2, 3)	{1, 2, 3}	{"a":1, "b":2}
Mutability	Mutable (can change)	Immutable (fixed)	Mutable	Mutable
Ordered	Yes (insertion order)	Yes	No (unordered)	Yes (since 3.7)
Duplicates	Allowed	Allowed	Not allowed	Keys: not allowed; Values: allowed
Indexing / Slicing	Yes	Yes	No	Via keys only
Typical Use	Ordered collection, may change	Fixed collection, e.g., coordinates	Unique elements, fast membership	Key-value mapping

What This Demonstrates

- List: preserves order, allows duplicates, and can be modified (append, insert, etc.).
- Set: automatically removes duplicates, provides fast membership testing, but does not maintain order.
- Tuple: fixed content – used here for course information that should not change.
- Dictionary: provides a mapping from unique keys (student IDs) to values (grades); keys must be unique and hashable.

Simple Case Study: Student Records

We want to manage student data for two courses, storing names, unique IDs, and grades. We'll use each of the four data structures where they fit best:

- Tuple: to represent fixed course information (course code, credits).
- Set: to ensure student IDs are unique.
- List: to maintain an ordered list of student names (e.g., by registration).
- Dictionary: to map student IDs to names, and to store grades per course.

The final output will be a table showing each student's ID, name, and grade in both courses.

```
1 course1 = ("CS101", 3) # (course code, credits)
2 course2 = ("MATH201", 4)
3 student_ids = {1001, 1002, 1003, 1004}
4 student_names_list = ["Ali", "Omar", "Rami", "Ahmed"]
5 id_to_name = dict(zip(student_ids, student_names_list))
6 grades_cs101 = {
7     1001: "A",
8     1002: "B+",
9     1004: "A-"
10 }
11 grades_math201 = {
12     1001: "B",
13     1002: "A",
14     1003: "C+"
15 }
16 print("Student Grade Report")
17 print("-" * 50)
18 print(f'{"ID":<10} {"Name":<12} {"course1[0]:<8} {"course2[0]:<10}"')
19 print("-" * 50)
20
21 for sid in (student_ids):
22     name = id_to_name.get(sid, "Unknown")
23     grade1 = grades_cs101.get(sid, "N/A")
24     grade2 = grades_math201.get(sid, "N/A")
25     print(f'{"ID":<10} {"Name":<12} {"grade1:<8} {"grade2:<10}"')
26
27 print("-" * 50)
```

"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe" "C:\Users\HP\PycharmProjects\First project\main.py"

ID	Name	CS101	MATH201
1001	Ali	A	B
1002	Omar	B+	A
1003	Rami	N/A	C+
1004	Ahmed	A-	N/A

Process finished with exit code 0