PROGRAMMING FUNDAMENTALS

CHAPTER 6
LISTS AND TUPLES

THE BIG IDEA
A list or tuple lets one variable name represent many related values. This chapter teaches you how to create, access, slice, search, compare and organise sequences, and how to decide when a list or tuple is the right tool.

Source coverage includes: compound types; lists; indexing; slicing; list mutation and methods; list comprehensions; tuples; tuple immutability; multiple assignment; multiple return values; sequence operations; membership; relational comparison; deletion; operator precedence; and introductory sorting/searching concepts.

Programming Principles • Trimester 2, 2026

Lei Wang

The Story

Week	Core idea	What you learned
1	CALCULATE	Values, types, operators and expressions
2	INTERACT	Variables, input, output and conversion
3	DECIDE	Boolean expressions and selection
4	REPEAT	for, while, range, break and continue
5	ORGANISE & REUSE	Functions, parameters, return and scope
6	COLLECT & STRUCTURE	Lists, tuples, sequences and collection operations

Learning Outcomes

- Explain sequences and identify lists and tuples as Python sequence types.
- Create empty and non-empty lists and tuples, including mixed data types.
- Use zero-based indexing, negative indexing and slicing correctly.
- Use concatenation, repetition, membership testing and iteration.
- Use len(), max(), min(), index() and count().
- Modify lists with indexing, slicing, del and common list methods.
- Explain mutability and the difference between lists and tuples.
- Write and read list comprehensions, including filtering.
- Use tuple packing, unpacking and multiple assignment.
- Swap variables using multiple assignment.
- Understand how a function can effectively return multiple values using a tuple.
- Compare sequences lexicographically.
- Trace and debug list/tuple programs systematically.

HOW TO STUDY

Read the explanation → predict the example → run it → explain why the result occurs.

For tracing questions, write the collection and its indices instead of guessing.

Quick Model

Question	Ask yourself
What is stored?	What are the elements?
Where is it?	What is its index?
Can it change?	List = yes; tuple = no.
What operation?	Read, slice, search, modify or create?

1. Python Compound Types

Python has built-in compound types that act as containers for other values. Here we concentrate on lists and tuples.

Type	Example	Main idea
list	[1, 2, 3]	Ordered, mutable collection
tuple	(1, 2, 3)	Ordered, immutable collection
dict	{'a': 1, 'b': 2}	Key-value mapping
set	{1, 2, 3}	Unique values; not an indexed sequence

Sequences

Strings, lists and tuples are sequences. They preserve order and share common operations such as indexing, slicing, concatenation, repetition, membership and iteration.

2. Arrays

A traditional array normally stores values of one type in a fixed-size structure, whereas a Python list can contain mixed types and can grow or shrink.

Feature	Traditional array	Python list
Type	Usually one element type	Can contain different types
Size	Typically fixed	Can grow/shrink
Access	Indexed	Indexed
Python form	Not the normal Python list model	[...]

COURSE FOCUS
Do not get stuck on the word 'array'.

For this course, the practical distinction is: list = mutable sequence; tuple = immutable sequence.

3. Lists: Ordered and Mutable

Lists are written with square brackets. Elements are separated by commas, and elements can have different types.

```
numbers = [10, 20, 30, 40]
names = ["Ada", "Lin", "Sam"]
mixed = [10, "hello", 3.14, [1, 2, 3]]
empty = []
```

- Lists preserve order.
- Lists may be empty.
- Lists can contain mixed types.
- Lists can be modified after creation.

Creating Lists

```
a = []
b = list()
c = [1, 2, 3]
d = list(range(5))

print(d)
# [0, 1, 2, 3, 4]
```

4. Indexing

Indexing selects one element by position. Python starts counting at zero.

Index	0	1	2	3	4
Value	2	3	5	7	11
Negative	-5	-4	-3	-2	-1

```
L = [2, 3, 5, 7, 11]

print(L[0])    # 2
print(L[1])    # 3
print(L[-1])   # 11
print(L[-2])   # 7
```

RULE
Positive indices start at 0 from the left. Negative indices start at -1 from the right.

Index Errors

```
L = [10, 20, 30]
print(L[2])    # 30
# print(L[3])  # IndexError
```

5. Slicing

Slicing selects several elements. The general form is:

```
sequence[start : stop : step]
```

Start is included; stop is excluded. Step controls the movement between selected indices.

```
L = [2, 3, 5, 7, 11]

print(L[0:3])    # [2, 3, 5]
print(L[:3])     # [2, 3, 5]
print(L[2:])     # [5, 7, 11]
print(L[-3:])    # [5, 7, 11]
print(L[:])      # [2, 3, 5, 7, 11]
print(L[::2])    # [2, 5, 11]
print(L[::-1])   # [11, 7, 5, 3, 2]
```

IMPORTANT SLICE RULE
start is included; stop is excluded. This is the same boundary idea used by range().

Reverse and Stepped Slices

```
s = "abcdefghijklmnopqrstuvwxyz"

print(s[3:7])      # defg
print(s[3:-3:2])   # dfhjlnprtv
print(s[::-1])     # reversed string
```

6. Common Sequence Operations

Operation	Example	Meaning
Concatenation	+	Joins sequences
Repetition	*	Repeats a sequence
Membership	in	Tests whether a value occurs
Non-membership	not in	Tests whether a value does not occur
Iteration	for x in sequence	Processes elements one at a time

```
print([1, 2] + [3, 4])
print(["Hi"] * 3)
print(3 in [1, 2, 3])
print(4 not in [1, 2, 3])

for x in [10, 20, 30]:
    print(x)
```

7. Useful Functions and Methods

Operation	Example	Purpose
len()	len(L)	Number of elements
max()	max(L)	Largest value
min()	min(L)	Smallest value
index()	L.index(x)	First position of x
count()	L.count(x)	Number of occurrences of x

```
L = [4, 7, 4, 9, 4]

print(len(L))       # 5
print(max(L))       # 9
print(min(L))       # 4
print(L.index(4))   # 0
print(L.count(4))   # 3
```

COMMON TRAP

index(x) finds the first matching position.

count(x) tells you how many times the value occurs.

If index(x) cannot find x, Python raises ValueError.

8. Changing a List

Lists are mutable, so an existing list can be changed by assigning to an index or a slice.

```
L = [2, 3, 5, 7, 11]

L[0] = 100
print(L)
# [100, 3, 5, 7, 11]

L[1:3] = [55, 56]
print(L)
# [100, 55, 56, 7, 11]
```

Index Assignment vs Slice Assignment

Index assignment replaces one element. Slice assignment can replace several elements and can even change the length of the list.

```
L = [10, 20, 30, 40]
L[1:3] = [7, 8, 9]
print(L)
# [10, 7, 8, 9, 40]
```

9. List Methods

Method	What it does	Example
append(x)	Adds one item to the end	L.append(8)
extend(iterable)	Adds all items from an iterable	L.extend([8,9])
insert(i,x)	Inserts x before index i	L.insert(1,99)
remove(x)	Removes first occurrence of x	L.remove(99)
pop(i)	Removes and returns item at i	x=L.pop(2)
pop()	Removes and returns last item	x=L.pop()
clear()	Removes all items	L.clear()
reverse()	Reverses the list in place	L.reverse()
sort()	Sorts the list in place	L.sort()

append() vs extend()

```
L = [1, 2]

L.append([3, 4])
print(L)
# [1, 2, [3, 4]]

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

REMEMBER
append() adds one item. extend() adds the items from another iterable.

remove() vs pop()

```
L = ["A", "B", "C", "D"]

L.remove("C")
print(L)          # ['A', 'B', 'D']

item = L.pop(1)
print(item)       # B
print(L)          # ['A', 'D']
```

remove() is value-based. pop() is position-based and returns the removed value.

sort() and reverse()

```
L = [5, 2, 9, 1]
L.sort()
print(L)          # [1, 2, 5, 9]

L.reverse()
print(L)          # [9, 5, 2, 1]
```

IN-PLACE WARNING
sort() and reverse() modify the existing list. Do not assume they return the new list. sorted(L) is different: it returns a new sorted list.

10. List Comprehensions

A list comprehension is a compact way to construct a list.

```
[expression for variable in iterable]

squares = [n ** 2 for n in range(10)]
print(squares)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Filtering

```
values = [n for n in range(20) if n % 3 > 0]
print(values)
# [1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19]
```

The if at the end is a filter: only values satisfying the condition are included.

Conditional Expressions

```
result = [n if n % 2 else -n
          for n in range(20)
          if n % 3]
print(result)
```

Here the final if filters items, while the earlier 'if ... else ...' chooses the value placed into the new list.

Nested Comprehensions

```
pairs = [(i, j) for i in range(2) for j in range(3)]
print(pairs)
# [(0,0), (0,1), (0,2), (1,0), (1,1), (1,2)]
```

USE WITH CAUTION
A comprehension is useful when it makes the idea clearer. If it becomes difficult to read, use an ordinary for loop.

11. Tuples

Tuples are ordered sequences like lists, but they are immutable. Parentheses are commonly used, although commas are what define the tuple.

```
t = (1, 2, 3)
u = 1, 2, 3

print(t)
print(u)
print(type(t))
```

Property	List	Tuple
Ordered	Yes	Yes
Indexing	Yes	Yes
Slicing	Yes	Yes
Mixed types	Yes	Yes
Mutable	Yes	No
Notation	[...]	(...)
append()	Available	Not available

Tuple Immutability

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

# t[1] = 4
# TypeError

# t.append(4)
# AttributeError
```

CENTRAL DISTINCTION
Use a list when the collection needs to change.

Use a tuple when the grouped values should stay fixed.

12. Tuple Operations and Methods

Tuples support indexing, slicing, concatenation, repetition, membership, iteration, count() and index(). They do not support list-changing methods.

```
t = ("red", "blue", "red", "green")
```

```
print(t[1])          # blue
print(t[-1])         # green
print(t[:2])         # ('red', 'blue')
print(t.count("red"))
print(t.index("green"))
```

13. Multiple Assignment and Unpacking

Python can assign several values in one statement. The number of values must match the number of target variables.

```
x, y = (4, "fred")
print(x)   # 4
print(y)   # fred

a, b = 99, 98
print(a)   # 99
print(b)   # 98
```

Swapping Values

```
a = 10
b = 20

a, b = b, a

print(a)   # 20
print(b)   # 10
```

WHY THIS WORKS
Python evaluates the right-hand side first, then assigns the resulting values to the variables on the left.

14. Multiple Return Values

A function returns one object. If that object is a tuple, the function can effectively return several related values.

```
from math import sin, cos

def polar_to_rect(r, theta):
    return (r * cos(theta), r * sin(theta))

x, y = polar_to_rect(5, 0.5)
```

The tuple returned by the function is unpacked into x and y. This is a direct connection between Week 5 functions and Week 6 tuples.

15. Comparing Sequences

Lists and tuples can be compared with relational operators. Python compares sequences lexicographically: from left to right, stopping at the first unequal pair.

```
print((0, 1, 2) < (5, 1, 2))       # True
print((0, 1, 2000000) < (0, 3, 4)) # True
```

```
print(("Jones", "Sally") < ("Jones", "Sam")) # True
print((1, 9) < (1, 8, 100))        # False
```

LEXICOGRAPHIC ORDER

Think dictionary-style: compare the first elements; if they are equal, move to the next pair.

Sequence length is not the first thing Python compares.

16. Deleting Parts of a List

```
L = [10, 20, 30, 40, 50]

del L[0]
print(L)
# [20, 30, 40, 50]

del L[1:3]
print(L)
# [20, 50]
```

Deletion using del is another operation available because lists are mutable.

17. Practical Programming Patterns

Pattern A — Build a list

```
values = []

for i in range(5):
    values.append(i * 10)

print(values)
# [0, 10, 20, 30, 40]
```

Pattern B — Filter a list

```
numbers = [4, 7, 10, 13, 16, 21]
multiples = []

for n in numbers:
    if n % 3 == 0:
        multiples.append(n)

print(multiples)   # [21]
```

Pattern C — Filter with a comprehension

```
multiples = [n for n in numbers if n % 3 == 0]
print(multiples)   # [21]
```

Pattern D — Search safely

```
names = ["Ada", "Lin", "Sam", "Ada"]
target = "Sam"
```

```
if target in names:
    print("Found at index", names.index(target))
else:
    print("Not found")
```

18. Trace a List Program

For quiz and exam questions, track the list after every statement.

```
values = [3, 1, 4]
values.append(2)
values.sort()
values.pop(1)
print(values)
```

Step	Statement	List
1	values = [3, 1, 4]	[3, 1, 4]
2	append(2)	[3, 1, 4, 2]
3	sort()	[1, 2, 3, 4]
4	pop(1)	[1, 3, 4]
5	print(values)	[1, 3, 4]

TRACE ROUTINE
Write the collection → write its indices → apply one statement → update the collection → repeat. For pop(), record the removed value if it is assigned.

19. Common Mistakes

Mistake	Correct idea
First item is L[1]	First item is L[0].
L[1:3] includes index 3	Stop is excluded.
append([3,4]) adds 3 and 4 separately	It adds one item: the list [3,4].
remove(2) removes index 2	remove() removes the first occurrence of value 2.
pop(2) searches for value 2	pop(2) removes the item at index 2.
Tuple can be changed with t[0]=...	Tuples are immutable.
sort() returns the sorted list	sort() modifies the list in place.
index() tells frequency	count() tells frequency.
Negative -1 is second last	-1 is the last element.
Sequence comparisons use length first	They compare lexicographically.
Every comprehension is better than a loop	Use the clearest form.

20. Choosing List or Tuple

Situation	List	Tuple
Values will change	✓	
Need append/remove/sort	✓	
Fixed group of related values		✓
Need indexing/slicing	✓	✓

Need multiple return values	Possible	Often natural

A practical beginner rule: mutable collection → list; fixed ordered group → tuple.

21. Problem-Solving Recipe

1. Identify the values that belong together.
2. Choose list or tuple based on whether the collection should change.
3. Create the collection.
4. Write down indices if you need position-based access.
5. Use a loop when processing every element.
6. Use in/not in before searching when appropriate.
7. Use built-in functions and methods for standard operations.
8. Trace the collection after each mutation.
9. Test empty, one-element, duplicate and boundary cases.

22. Quiz and Exam Strategy

Indexing

1. Write the sequence exactly.
2. Number positions from 0.
3. For negative indices, count from -1 at the right.
4. Select the requested element.

Slicing

1. Identify start, stop and step.
2. Include start.
3. Exclude stop.
4. Move according to step.

Methods

1. Ask whether the method changes the list.
2. For append/extend/insert, write the new list.
3. For remove/pop, identify exactly what is removed.
4. For sort/reverse, update the list before continuing.

Comprehensions

1. Find the iterable.
2. Find any filter condition.
3. For each selected item, evaluate the expression.
4. Write results in order.

23. Predict Before You Run

Challenge 1 — Indexing

```
L = ["A", "B", "C", "D"]
print(L[0])
print(L[-1])
print(L[1:3])
```

Challenge 2 — Mutation

```
L = [5, 2, 8]
L[1] = 9
L.append(3)
print(L)
```

Challenge 3 — Methods

```
L = [4, 1, 4, 2]
x = L.pop(1)
print(x)
print(L)
print(L.count(4))
```

Challenge 4 — Comprehension

```
result = [n * 2 for n in range(6) if n % 2 == 1]
print(result)
```

Challenge 5 — Unpacking

```
a, b = 7, 3
a, b = b, a
print(a, b)
```

Challenge 6 — Comparison

```
print((1, 9) < (2, 0))
print((1, 9) < (1, 8, 100))
```

Challenge 7 — Nested comprehension

```
pairs = [(i, j) for i in range(2) for j in range(2)]
print(pairs)
```

Answers

```
Challenge 1
A
D
['B', 'C']

Challenge 2
[5, 9, 8, 3]
```

```
Challenge 3
1
[4, 4, 2]
2

Challenge 4
[2, 6, 10]

Challenge 5
3 7

Challenge 6
True
False

Challenge 7
[(0, 0), (0, 1), (1, 0), (1, 1)]
```

24. Practice: Build Real Skill

Practice 1 — Basic List Operations

Create a list of five integers. Print its length, first element, last element, largest value, smallest value and reverse.

Practice 2 — List Modification

Start with [10, 20, 30]. Append 40, insert 15 at index 1, remove 30 and sort the list. Print each stage.

Practice 3 — Filtering

Given a list of integers, create a new list containing only values divisible by 4. Solve it with a for loop and a list comprehension.

Practice 4 — Search

Ask for a value and report whether it occurs. If it does, print its first index and its count.

Practice 5 — Tuple Unpacking

Create a tuple containing a name, age and city. Unpack it and print a sentence.

Practice 6 — Multiple Return Values

Write min_max(a,b) that returns the smaller and larger values as a tuple, then unpack them.

Practice 7 — Trace

Write a short program using append(), insert(), pop(), remove() and sort(). Predict the final list before running it.

25. Deeper Understanding

List = Mutable Object

A list variable refers to a list object. Methods such as append() change that object.

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

Tuple = Fixed Group

```
point = (3, 7)
x, y = point
print(x)
print(y)
```

This makes tuples natural for fixed records, coordinates and grouped return values.

26. Knowledge Map

Concept	Remember this
Sequence	Ordered collection of elements
List	Mutable sequence written with []
Tuple	Immutable sequence usually written with ()
Indexing	One element: sequence[i]
Negative indexing	-1 is last
Slicing	start included, stop excluded
Concatenation	sequence1 + sequence2
Repetition	sequence * n
Membership	x in sequence / x not in sequence
len()	Number of elements
max()/min()	Largest/smallest value
index()	First position
count()	Number of occurrences
append()	Add one item to end
extend()	Add items from an iterable
insert()	Insert before an index
remove()	Remove first matching value
pop()	Remove by index and return item
sort()	Sort list in place
reverse()	Reverse list in place
Comprehension	Compact list construction/filtering
Unpacking	Assign sequence elements to variables
Lexicographic comparison	Compare left to right until a difference

27. Checklist

☐ I can explain why collections are useful.

☐ I can distinguish a list from a tuple.

☐ I understand mutable vs immutable.

☐ I can create empty and non-empty lists and tuples.

☐ I can use positive and negative indices.

☐ I can trace slices with start, stop and step.

☐ I can use +, *, in, not in and for with sequences.

☐ I can use len(), max(), min(), index() and count().

☐ I can modify a list by index or slice.

☐ I can use append(), extend(), insert(), remove(), pop() and clear().

☐ I understand that sort() and reverse() modify a list.

☐ I can explain append() vs extend() and remove() vs pop().

☐ I can write simple list comprehensions.

☐ I can use filtering in a comprehension.

☐ I can explain tuple immutability.

☐ I can unpack a tuple and swap variables.

☐ I understand tuple-based multiple return values.

☐ I can compare sequences lexicographically.

☐ I can trace list/tuple programs.

☐ I can choose a list or tuple appropriately.

28. Key Takeaways

Idea	Memory rule
List	Ordered collection that can change.
Tuple	Ordered collection that cannot change.
Index	Positions start at 0.
Negative index	-1 is the last item.
Slice	Start included, stop excluded.
append	Adds one item.
extend	Adds all items from another iterable.
remove	Removes by value.
pop	Removes by index and returns the item.
sort	Sorts in place.
reverse	Reverses in place.
Comprehension	Builds a list compactly.
Unpacking	Assigns sequence elements to variables.
Comparison	Sequences are compared left to right.

FOUR QUESTIONS

1. What values are stored? 2. What are their indices? 3. Is the collection mutable? 4. Does this operation read, create or modify?

29. Reflection

Why are collections more useful than many separate variables?

_____

_____

What is the difference between L[2] and L[2:3]?

_____

_____

Why is L[-1] useful?

_____

_____

What does mutable mean?

_____

_____

What is the difference between append() and extend()?

_____

_____

What is the difference between remove() and pop()?

_____

_____

Why can a tuple be indexed even though it cannot be changed?

_____

_____

When would you choose a tuple instead of a list?

_____

_____

How does the stop index work in a slice?

_____

_____

Why can a function return several related values using a tuple?

_____

_____

Why is (1, 9) < (1, 8, 100) false?

_____

_____