Python Nested Loops

1. What Are Nested Loops?

A **nested loop** is a loop placed inside another loop.

In simple words:

> **One loop runs inside another loop.**

For example:

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

Here:

- The outer `for` loop controls `i`.
- The inner `for` loop controls `j`.
- For **every one iteration** of the outer loop, the inner loop runs completely.

Output:

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
```

```
2 1
2 2
```

---

2. Why Do We Need Nested Loops?

A normal loop is useful when you need to process one sequence.

But many real problems have **multiple levels of data**.

For example:

```
Students
   
Each student's subjects
```

or:

```
Products
   
Each product's variants
```

or:

```
Rows
   
Columns
```

A nested loop lets you work through these multiple levels.

Common uses

Nested loops are useful for:

- tables
- grids
- matrices
- patterns
- 2D lists
- comparing items
- combinations
- schedules
- game boards
- image pixels
- multi-level data
- algorithms

---

3. The Mental Model

Suppose the outer loop runs 3 times and the inner loop runs 2 times.

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

Think of it like:

```
Outer iteration 1
    Inner iteration 1
    Inner iteration 2

Outer iteration 2
```

```
  Inner iteration 1
  Inner iteration 2

Outer iteration 3
  Inner iteration 1
  Inner iteration 2
```

So the inner loop runs **completely for each outer iteration**.

---

4. Basic Syntax

The general structure is:

```
for outer_item in outer_sequence:
  for inner_item in inner_sequence:
    # code
```

The indentation is important.

Example:

```
for i in range(3):
  for j in range(3):
    print(i, j)
```

The second `for` is indented because it belongs to the outer loop.

---

5. How Nested Loops Execute

Consider:

```
for i in range(2):
    for j in range(3):
        print(i, j)
```

Let's trace it.

First outer iteration

`i = 0`

The inner loop runs:

```
j = 0
j = 1
j = 2
```

Output:

```
0 0
0 1
0 2
```

Second outer iteration

`i = 1`

The inner loop starts again:

```
j = 0
j = 1
j = 2
```

Output:

```
1 0
1 1
1 2
```

Final output:

```
0 0
0 1
0 2
1 0
1 1
1 2
```

---

6. Nested `for` Loops

The most common type of nested loop is a `for` loop inside another `for` loop.

```
for i in range(3):
    for j in range(2):
        print("i =", i, "j =", j)
```

Output:

```
i = 0 j = 0
i = 0 j = 1
i = 1 j = 0
i = 1 j = 1
i = 2 j = 0
i = 2 j = 1
```

---

7. Nested `while` Loops

You can also nest `while` loops.

```
i = 1

while i <= 3:
    j = 1

    while j <= 2:
        print(i, j)
        j += 1

    i += 1
```

Output:

```
1 1
1 2
2 1
2 2
3 1
3 2
```

Important

With nested `while` loops, make sure **both loop variables are updated**.

Otherwise, you can accidentally create an infinite loop.

---

8. Mixing `for` and `while`

Nested loops don't have to use the same type.

You can put a `while` loop inside a `for` loop:

```
for i in range(3):
    j = 0

    while j < 2:
        print(i, j)
        j += 1
```

Or a `for` loop inside a `while` loop:

```
i = 0

while i < 3:
    for j in range(2):
        print(i, j)

    i += 1
```

---

9. Nested Loops With `range()`

`range()` is frequently used with nested loops.

```
for row in range(3):
    for column in range(4):
        print(row, column)
```

There are:

- 3 outer iterations
- 4 inner iterations for each outer iteration

Total:

`3 × 4 = 12`

Output contains 12 pairs.

---

10. Printing Rows and Columns

Nested loops are excellent for working with rows and columns.

```
for row in range(3):
    for column in range(4):
        print("*", end=" ")

    print()
```

Output:

```
* * * *
* * * *
* * * *
```

Here:

- Outer loop controls rows.
- Inner loop controls columns.

Mental model

Outer loop How many rows?
Inner loop How many items per row?

---

11. Creating a Square Pattern

```
for row in range(5):
    for column in range(5):
        print("*", end=" ")

    print()
```

Output:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

The outer loop creates 5 rows.

The inner loop creates 5 stars in every row.

---

12. Creating a Triangle Pattern

Nested loops can create patterns.

```
for row in range(1, 6):
    for column in range(row):
        print("*", end=" ")

    print()
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Notice that the number of inner-loop iterations changes according to the current row.

---

13. Number Pattern

```
for row in range(1, 5):
    for number in range(1, row + 1):
        print(number, end=" ")

    print()
```

Output:

```
1
1 2
1 2 3
1 2 3 4
```

This demonstrates how the outer loop can control the behavior of the inner loop.

---

14. Multiplication Tables

Nested loops are useful for generating multiplication tables.

```
for number in range(1, 4):
    for multiplier in range(1, 6):
        print(number * multiplier, end=" ")

    print()
```

Output:

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
```

The outer loop selects the number.

The inner loop calculates its multiples.

---

15. Nested Loops With Lists

Suppose you have:

```
students = [
    ["Ali", "Sara"],
    ["Hafsa", "Ahmed"],
    ["Zain", "Ayesha"]
]
```

You can use nested loops:

```
for group in students:
    for student in group:
        print(student)
```

Output:

```
Ali
Sara
Hafsa
Ahmed
Zain
Ayesha
```

The outer loop gets each inner list.

The inner loop gets each student.

---

16. Working With a 2D List

A two-dimensional list can be thought of as a table.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

Use nested loops:

```
for row in matrix:
    for value in row:
        print(value, end=" ")

    print()
```

Output:

```
1 2 3
4 5 6
7 8 9
```

Mental model

```
matrix
 
rows
 
values inside each row
```

---

17. Accessing Rows and Columns

You can also use indexes.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for row in range(len(matrix)):
    for column in range(len(matrix[row])):
```

```
        print(matrix[row][column])
```

This gives access to both the row and column indexes.

---

18. Nested Loops and `enumerate()`

When you need indexes, `enumerate()` can make the code clearer.

```
students = [
    ["Ali", "Sara"],
    ["Hafsa", "Ahmed"]
]

for row_index, row in enumerate(students):
    for column_index, student in enumerate(row):
        print(row_index, column_index, student)
```

Output:

```
0 0 Ali
0 1 Sara
1 0 Hafsa
1 1 Ahmed
```

This is often easier to understand than manually managing indexes.

---

19. Nested Loops for Searching

Suppose you want to search through groups of users.

```
groups = [
    ["Ali", "Sara"],
    ["Hafsa", "Ahmed"],
    ["Zain", "Ayesha"]
]

target = "Hafsa"

for group in groups:
    for user in group:
        if user == target:
            print("User found")
```

Output:

```
User found
```

Nested loops allow you to search through multiple levels of data.

---

20. Using `break` in Nested Loops

`break` inside nested loops stops the **innermost loop**.

```
for i in range(3):
    for j in range(3):
        if j == 1:
            break

        print(i, j)
```

Output:

```
0 0
1 0
2 0
```

When `j == 1`, the inner loop stops.

The outer loop continues.

Important rule

`break`
□
Stops the nearest/current loop

It does not automatically stop all nested loops.

---

21. Using `continue` in Nested Loops

`continue` skips the current iteration of the loop where it appears.

```
for i in range(3):
    for j in range(3):
        if j == 1:
            continue

        print(i, j)
```

Output:

```
0 0
0 2
1 0
```

```
1 2
2 0
2 2
```

Only the inner-loop iteration where `j == 1` is skipped.

---

22. Using `pass` in Nested Loops

`pass` does nothing.

```
for i in range(2):
    for j in range(2):
        if j == 1:
            pass

        print(i, j)
```

Output:

```
0 0
0 1
1 0
1 1
```

This is different from `continue`.

---

23. `break`, `continue`, and `pass` Together

Consider:

```
for row in range(3):
    for column in range(3):

        if column == 0:
            pass

        if column == 1:
            continue

        if column == 2:
            break

        print(row, column)
```

Understanding which statement controls which part of the loop is essential when working with nested loops.

---

24. Comparing Every Item With Every Other Item

Nested loops are commonly used when every item needs to be compared with other items.

Example:

```
numbers = [1, 2, 3]

for first in numbers:
    for second in numbers:
        print(first, second)
```

Output:

```
1 1
```

```
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

There are:

```
3 × 3 = 9
```

comparisons/pairs.

This concept appears in many algorithms.

---

25. Generating Combinations

Nested loops can generate combinations.

```
colors = ["Red", "Blue"]
sizes = ["S", "M", "L"]

for color in colors:
    for size in sizes:
        print(color, size)
```

Output:

```
Red S
```

```
Red M
Red L
Blue S
Blue M
Blue L
```

This can represent product variants:

```
Color × Size
```

---

26. Nested Loops in Real Applications

Nested loops appear in many real-world situations.

E-commerce

```
Products
   
Variants
```

Education

```
Students
   
Subjects
```

Scheduling

```
Days
   
Time slots
```

Games

Rows
  □
Columns

Data processing

Records
  □
Fields

Web/API data

Responses
  □
Items

---

27. Real haas.dev Example

Imagine haas.dev has resources grouped by category:

```
resources = {
    "Python": [
        "Variables",
        "Conditions",
        "Loops"
    ],
    "JavaScript": [
        "Variables",
        "Functions",
        "Arrays"
    ],
    "DSA": [
        "Arrays",
```

```
        "Searching",
        "Sorting"
    ]
}
```

You can process them with nested loops:

```
for category, items in resources.items():
    print(category)

    for item in items:
        print(" -", item)
```

Output:

```
Python
 - Variables
 - Conditions
 - Loops
JavaScript
 - Variables
 - Functions
 - Arrays
DSA
 - Arrays
 - Searching
 - Sorting
```

This is a realistic example of hierarchical data.

---

28. Nested Loops and Complexity

Nested loops are important in algorithm analysis.

Consider:

```
for i in range(n):
    for j in range(n):
        print(i, j)
```

The outer loop runs n times.

For every outer iteration, the inner loop runs n times.

Therefore:

$n \times n = n^2$

The time complexity is:

$O(n^2)$

For example:

$n = 10$
$10 \times 10 = 100$ iterations

$n = 100$
$100 \times 100 = 10{,}000$ iterations

$n = 1{,}000$
$1{,}000 \times 1{,}000 = 1{,}000{,}000$ iterations

This is why nested loops can become expensive when working with large datasets.

---

29. Nested Loops With Different Sizes

Not every nested loop is $O(n^2)$.

Consider:

```
for i in range(n):
    for j in range(m):
        print(i, j)
```

The outer loop runs `n` times.

The inner loop runs `m` times.

Total:

`n × m`

Time complexity:

O(n × m)

This distinction becomes important when analyzing algorithms.

---

30. Nested Loops Are Not Always Bad

Seeing a nested loop does not automatically mean the code is inefficient.

For a small grid:

```
for row in range(10):
    for column in range(10):
        print(row, column)
```

There are only:

`100 iterations`

The important question is:

> **How large can the input become?**

You should consider both the problem requirements and the size of the data.

---

31. Three Nested Loops

Python allows more than two levels.

```
for i in range(2):
    for j in range(2):
        for k in range(2):
            print(i, j, k)
```

Total iterations:

`2 × 2 × 2 = 8`

For `n` iterations at each level, three nested loops can result in:

$O(n^3)$

Three or more nested loops can be useful for certain problems, but they can also become expensive quickly.

---

32. Common Mistake: Wrong Indentation

Incorrect:

```
for i in range(3):
for j in range(3):
   print(i, j)
```

Python will raise an indentation error.

Correct:

```
for i in range(3):
   for j in range(3):
      print(i, j)
```

Indentation defines the relationship between the loops.

---

33. Common Mistake: Forgetting the Inner Loop Restarts

Consider:

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

Some beginners expect `j` to continue from where it stopped.

It doesn't.

For every new `i`, the inner loop starts again:

```
i = 0 → j = 0, 1
i = 1 → j = 0, 1
i = 2 → j = 0, 1
```

---

34. Common Mistake: Infinite `while` Loops

This is dangerous:

```
i = 0

while i < 3:
    j = 0

    while j < 3:
        print(i, j)
```

`j` never changes.

So:

```
j = 0
```

```
j = 0
j = 0
...
```

The inner loop never ends.

Correct:

```
i = 0

while i < 3:
    j = 0

    while j < 3:
        print(i, j)
        j += 1

    i += 1
```

---

35. Common Mistake: Using the Same Variable Carelessly

Avoid confusing code such as:

```
for i in range(3):
    for i in range(3):
        print(i)
```

The inner loop overwrites the outer i.

Prefer:

```
for row in range(3):
    for column in range(3):
        print(row, column)
```

Meaningful variable names make nested loops much easier to understand.

---

36. Common Mistake: Too Much Nesting

This:

```
for a in data:
    for b in a:
        for c in b:
            for d in c:
                # more logic
```

can become difficult to understand and maintain.

When nesting becomes complicated, consider:

- functions
- helper functions
- better data structures
- list comprehensions where appropriate
- built-in functions
- more efficient algorithms

Don't remove nesting simply for the sake of removing it. First understand what the algorithm actually needs.

---

37. Nested Loops and List Comprehensions

Some simple nested loops can be written using a list comprehension.

Normal version:

```
numbers = []

for x in range(3):
    for y in range(3):
        numbers.append((x, y))
```

Equivalent:

```
numbers = [
    (x, y)
    for x in range(3)
    for y in range(3)
]
```

The comprehension is shorter, but the normal nested loop may be easier for beginners to understand.

Learn the loop logic first.

---

38. Debugging Nested Loops

When a nested loop gives unexpected output, print the variables:

```
for row in range(3):
    for column in range(3):
```

```
    print("row:", row, "column:", column)
```

This helps you see exactly how the values change.

Debugging checklist

Ask:

1. How many times does the outer loop run?
2. How many times does the inner loop run for each outer iteration?
3. Which variable controls each loop?
4. Is the indentation correct?
5. Is `break` or `continue` affecting the intended loop?
6. Could a `while` loop be stuck?

---

39. Mini Quiz

Question 1

What is a nested loop?

A. A loop that never ends

B. A loop inside another loop

C. A loop with no condition

D. A loop with two variables

Answer: B

Question 2

How many times does the inner loop run?

```
for i in range(3):
    for j in range(4):
        print(i, j)
```

A. 3

B. 4

C. 7

D. 12

Answer: D

Question 3

What does `break` do inside the inner loop?

```
for i in range(5):
    for j in range(5):
        if j == 2:
            break
```

Answer: It stops the inner loop. The outer loop can continue.

---

40. 5 Minute Challenge

Create a program that prints this pattern:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Requirements:

- Use nested `for` loops.
- Outer loop controls the rows.
- Inner loop controls the numbers.
- Use `range()`.
- Use `print(..., end=" ")` for values on the same line.
- Use `print()` after each row.

Then modify it to produce:

```
*
* *
* * *
* * * *
* * * * *
```

---

41. Exercises

Exercise 1: Rectangle

Print:

```
* * * *
* * * *
* * * *
```

Use nested loops.

Exercise 2: Number Triangle

Print:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Exercise 3: Multiplication Table

Use nested loops to generate multiplication tables from `1` to `5`.

Exercise 4: 2D List

Given:

```
matrix = [
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
]
```

Use nested loops to print every value.

Exercise 5: Search Nested Data

Given:

```
groups = [
    ["Ali", "Sara"],
    ["Hafsa", "Ahmed"],
    ["Zain", "Ayesha"]
]
```

Search for `"Ahmed"` using nested loops.

---

42. Mini Project: Student Subject Report

Create a program that stores students and their subjects:

```
students = {
    "Ali": ["Math", "English", "Physics"],
    "Sara": ["Math", "Computer", "English"],
    "Hafsa": ["Python", "DSA", "Database"]
}
```

Use nested loops to display:

```
Ali
Math
English
Physics
```

```
Sara
Math
Computer
English

Hafsa
Python
DSA
Database
```

Extra challenge

Add marks:

```
students = {
    "Ali": {
        "Math": 80,
        "English": 75
    },
    "Sara": {
        "Math": 90,
        "English": 85
    }
}
```

Then use nested loops to print each student's subjects and marks.

---

43. Interview Questions

Beginner

1. What is a nested loop?

A loop placed inside another loop.

2. How does a nested loop execute?

For every iteration of the outer loop, the inner loop completes all of its iterations.

3. Can `for` and `while` loops be nested?

Yes. Any combination of `for` and `while` loops can be nested.

Intermediate

4. What happens when `break` is used inside a nested loop?

It exits the innermost loop containing the `break`.

5. What happens when `continue` is used inside a nested loop?

It skips the current iteration of the innermost loop containing the `continue`.

6. Why are nested loops useful?

They are useful for processing multidimensional data, grids, tables, combinations, patterns, and problems involving multiple levels of data.

Advanced

7. What is the time complexity of two nested loops where each runs `n` times?

$O(n^2)$.

8. What is the complexity if the outer loop runs `n` times and the inner loop runs `m` times?

O(n × m).

9. Are nested loops always inefficient?

No. Their efficiency depends on the number of iterations and the size of the input.

---

44. Cheat Sheet

Basic nested loop

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

Rows and columns

```
for row in range(rows):
    for column in range(columns):
        print("*", end=" ")

    print()
```

2D list

```
for row in matrix:
    for value in row:
        print(value)
```

Search

```
for group in groups:
    for item in group:
        if item == target:
            print("Found")
```

Complexity

n iterations × n iterations
= n^2
= $O(n^2)$

Loop controls

break
 Exit current loop

continue
 Skip current iteration

pass
 Do nothing

---

45. Key Takeaways

- A nested loop is a loop inside another loop.
- The inner loop runs completely for every outer-loop iteration.
- Nested loops are useful for tables, grids, patterns, matrices, and hierarchical data.
- They are commonly used with 2D lists.
- `break` affects the innermost loop where it appears.
- `continue` skips an iteration of the innermost loop where it appears.

- Nested loops can be used with `for`, `while`, or both together.
- Two `n`-sized nested loops commonly result in $O(n^2)$ time complexity.
- Three `n`-sized nested loops can result in $O(n^3)$.
- Meaningful variable names such as `row` and `column` make nested loops easier to understand.
- Always be careful with nested `while` loops because incorrect updates can create infinite loops.

The core idea

```
Outer Loop
   
   Inner Loop
      
      Run completely
   
Next outer iteration
   
Inner Loop runs again
```

Visit haas.dev for more resources and guides.

haas.dev

https://dev-roast-app.vercel.app/resources