مكتب

# Quick Reference Notes

Week 10: Nested Loops and 2D Patterns

7th Grade Computer Science

## 1 Introduction

### 1.1 This Week's Big Question

How can we make computers create two-dimensional patterns and grids? Just like weaving a beautiful carpet requires both horizontal and vertical threads working together, nested loops let us create patterns that span both rows and columns. This week, we'll discover how loops inside loops unlock a whole new dimension of programming power!

**Prerequisites**

Before starting this week, you should be comfortable with:

- Creating and using for loops with `range()` (Week 9)
- Understanding loop variables and how they change
- Tracing through single loops step by step
- Using proper indentation for loop bodies
- Basic loop patterns *(building, accumulating, processing)*

### 1.2 What You Already Know

You can use for loops to repeat code a specific number of times. You understand how the loop variable changes with each iteration and can trace through loops using memory diagrams. You've created patterns that go in one direction - like printing stars in a row or counting up numbers.

### 1.3 What You'll Be Able to Do

By the end of this week, you will be able to:

- Put loops inside other loops to create nested structures
- Visualize how nested loops work using tables and diagrams
- Build 2D patterns like rectangles, triangles, and diamonds
- Create grids for games and coordinate systems
- Debug nested loops by understanding the order of execution
- Recognize when nested loops make problems easier to solve

# 2 VIDEO 1: Nested Loops in Depth

## 2.1 What are Nested Loops?

A nested loop is simply a loop inside another loop. Think of it like a clock - the minute hand completes a full circle for each hour the hour hand moves. The inner loop (minutes) runs completely for each iteration of the outer loop (hours).

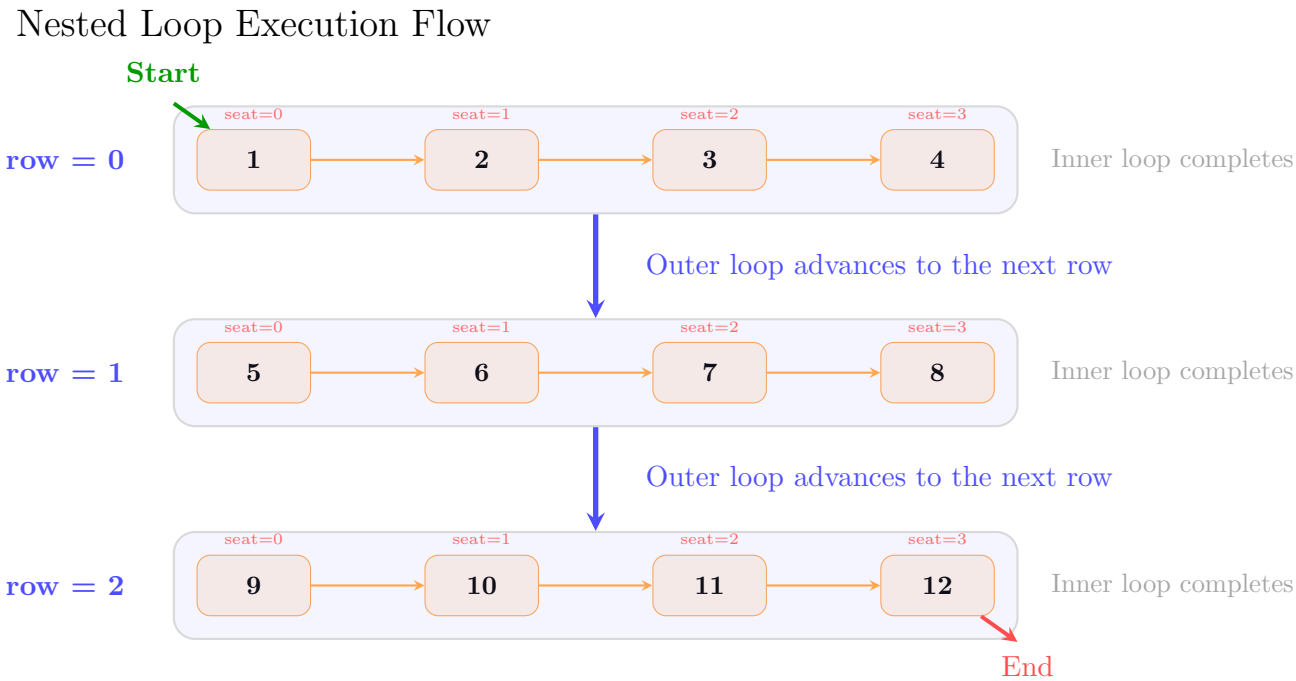
## 2.2 Let's See It In Action

A simple nested loop showing a $3 \times 4$ classroom seating chart

```python
for row in range(3):
    print(f"Row {row + 1}:", end=" ")
    for seat in range(4):
        print(f"Seat{seat + 1}", end=" ")
    print()  # Important: New line after each row

# Output:
# Row 1: Seat1 Seat2 Seat3 Seat4
# Row 2: Seat1 Seat2 Seat3 Seat4
# Row 3: Seat1 Seat2 Seat3 Seat4
```

## 2.3 Breaking It Down




*Each gray box represents one iteration of the outer loop, containing all 4 inner loop iterations*

The numbers in each of the smaller orange boxes shows the current step number.

Let's understand the flow:

1. **Outer loop starts**: `row = 0`
2. Print "`Row 1:`"

3. **Inner loop runs completely**: prints all 4 seats
4. Move to next line with `print()`
5. **Outer loop continues**: `row = 1`
6. Process repeats...

### 2.4 The Key Insight

For nested loops with `range(m)` and `range(n)`, the inner code runs $m \times n$ times total. In our example: 3 rows $\times$ 4 seats $=$ 12 total iterations.

**Rule**

**Nested Loop Pattern:**

```
for outer_variable in range(outer_count):
    # This runs outer_count times
    for inner_variable in range(inner_count):
        # This runs outer_count * inner_count times
    # Back to outer loop after inner completes
```

**Remember:** The inner loop must complete ALL its iterations before the outer loop moves to its next value.

**Quick Check**

How many times will `"Hello"` print?

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

## 3 VIDEO 2: Loop Visualization Techniques

### 3.1 Understanding with Tables

When tracing nested loops, tables help us see exactly what's happening. Let's trace a simple example:

```
for x in range(2):
    for y in range(3):
        print(f"({x},{y})", end=" ")
    print()
```

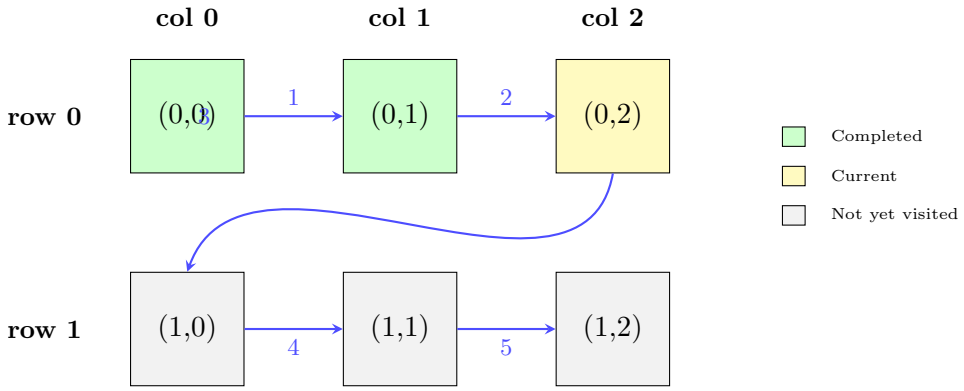
### 3.2 Execution Table

Here's how the variables change:

| Step | x | y | Output |
|---|---|---|---|
| 1 | 0 | 0 | (0,0) |
| 2 | 0 | 1 | (0,1) |
| 3 | 0 | 2 | (0,2) |
| 4 | | | *[new line]* |
| 5 | 1 | 0 | (1,0) |
| 6 | 1 | 1 | (1,1) |
| 7 | 1 | 2 | (1,2) |
| 8 | | | *[new line]* |

**Color coding:** The light gray rows represent the inner loop iterations.

### 3.3 The 2D Grid Mental Model




*Inner loop fills columns, outer loop moves to next row*

Think of nested loops as filling a grid. Fill each row completely before moving to the next one.

- Outer loop = rows (vertical)
- Inner loop = columns (horizontal)

**Quick Check**

In a nested loop where outer range is `range(4)` and inner range is `range(2)`, which of the following is true?

(a) Inner loop variable goes 0,1,2,3

(b) We get 4 rows and 2 columns

(c) Total iterations = 6

(d) Outer loop runs 8 times

## 4 VIDEO 3: Building 2D Patterns

### 4.1 From Lines to Shapes

Single loops create lines. Nested loops create shapes! Let's build some fundamental patterns that form the basis for more complex designs.

### 4.2 Pattern 1: Rectangle

A solid rectangle of hashtags

```
width = 5
height = 3

for row in range(height):
    for col in range(width):
        print("#", end="")
    print()  # New line after each row

# Output:

# #####
# #####
# #####
```

### 4.3 Pattern 2: Right Triangle

Triangle that grows each row

```
size = 4

for row in range(size):
    for col in range(row + 1):
        print("*", end="")
    print()

# Output:

# *
# **
# ***
# ****
```

Notice how the inner loop's range depends on the outer loop variable!

### 4.4 Pattern 3: Hollow Rectangle

Rectangle with hollow center

```
width = 6
height = 4

for row in range(height):
    for col in range(width):
        if row == 0 or row == height-1 or col == 0 or col == width-1:
            print("o", end="")
        else:
            print(" ", end="")
    print()

# Output:

# oooooo
# o    o
# o    o
# oooooo
```

### Common Nested Loop Patterns

**All positions filled**

```
#####
#####
#####
```

*5×3 solid rectangle*

**Columns depend on row number**

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

*Right triangle*

**Border only - condition checks edges**

```
oooooo
o    o
o    o
oooooo
```

*6×4 hollow rectangle*

**Common Error**

**Pattern Building Pitfalls:**

- Forgetting `print()` after inner loop → Everything on one line
- Using `print()` instead of `print(end="")` → Each character on new line
- Off-by-one errors in conditions → Wrong pattern size
- Not considering 0-indexing → Patterns shifted or incomplete

**Quick Check**

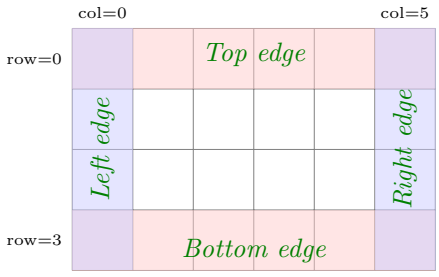
What pattern does this create?

```
for i in range(3):
    for j in range(5-i):
        print("+", end="")
    print()
```

Trace it before running!

## 4.5 Understanding the Hollow Rectangle Example

Let's break down how we decide where to place the `"o"` characters:




The magic happens in this if statement:

```
if row == 0 or row == height-1 or col == 0 or col == width-1:
    print("o", end="")
```

Think of it as asking: "Am I on any edge?" We print `"o"` if we're on:

- The **top** (`row = 0`) OR
- The **bottom** (`row = 3`) OR
- The **left** (`col = 0`) OR
- The **right** (`col = 5`)

**Example positions:**

Are we on any edge?

- Position `(0,2)`: `row=0` → YES, print "o" *(top edge)*
- Position `(1,1)`: Not on any edge → print space
  *All of the conditions in the if statement fail*
- Position `(2,0)`: `col=0` → YES, print `"o"` *(left edge)*

# 5 VIDEO 4: Creating Grids

## 5.1 Grids Are Everywhere

Grids organize information in rows and columns - like a chessboard, tic-tac-toe board, or even your class seating arrangement. Let's learn to create and work with grid patterns.

## 5.2 Numbered Grid

Create a 3x3 numbered grid (like phone keypad)

```
rows = 3
cols = 3
number = 1

for row in range(rows):
    for col in range(cols):
        print(f"{number}  ", end=" ")
        number = number + 1
    print()

# Output:
#  1  2  3
#  4  5  6
#  7  8  9
```

## 5.3 Coordinate Grid

Show `(row,col)` coordinates - useful for games!

```
for row in range(3):
    for col in range(4):
        print(f"({row},{col})", end=" ")
    print()

# Output:
# (0,0) (0,1) (0,2) (0,3)
# (1,0) (1,1) (1,2) (1,3)
# (2,0) (2,1) (2,2) (2,3)
```

### 5.4 Checkerboard Pattern

Alternating pattern like a chess board

```python
size = 5

for row in range(size):
    for col in range(size):
        if (row + col) % 2 == 0:
            print("*", end=" ")
        else:
            print("o", end=" ")
    print()

# Output:
# * o * o *
# o * o * o
# * o * o *
# o * o * o
# * o * o *
```

**Rule**

**Grid Patterns - Key Techniques:**

- **Sequential numbering**: Use a counter variable that increments
- **Coordinates**: Print `(row, col)` directly
- **Alternating**: Use `(row + col) % 2` for checkerboard
- **Borders**: Check if `row == 0` or `col == 0` etc.
- **Diagonals**: Check if `row == col` (main diagonal)

**Quick Check**

In a 4x4 grid, which cell is at the center? Remember: we start counting from 0!

(a) (2,2)

(b) (1,1) and (2,2)

(c) There's no single center

(d) (1.5, 1.5)

## 6 VIDEO 5: Summary

### 6.1 The Power of Nested Loops

This week, we've unlocked a new dimension in programming! With nested loops, you can now create complex 2D patterns, grids, and structures that would be impossible with single loops alone.

### 6.2 Key Concepts We Covered

| Concept | Key Points | Example Use |
|---|---|---|
| Nested loop basics | Inner loop completes for each outer | Seating charts |
| Visualization | Tables and grids help trace execution | Debug complex loops |
| 2D patterns | Rectangles, triangles, hollow shapes | ASCII art, designs |
| Grid systems | Coordinates, numbering, alternating | Game boards |
| Loop relationships | Inner range can depend on outer | Triangular patterns |

### 6.3 Common Patterns to Remember

- **Rectangle**: Both loops use fixed ranges
- **Triangle**: Inner range depends on outer variable
- **Hollow shapes**: Check if on border with conditions
- **Checkerboard**: Use `(row + col) % 2`
- **Coordinates**: Print `(row, col)` directly

**Landmark Moment**

You've mastered nested loops - one of programming's most powerful pattern-creating tools! You can now think in two dimensions, creating everything from simple rectangles to complex grids. This skill opens doors to game development, data visualization, and mathematical modeling. Next week, you'll combine these nested structures with conditions for even more sophisticated programs!

## 7 Reflection

Think about your learning journey this week:

1. When you first saw a loop inside another loop, what was your initial reaction? How did visualization help you understand?
2. The concept of "inner loop completes for each outer" is crucial. What analogy helps you remember this? (Clock? Calendar? Something else?)
3. Look at a pattern you created this week. Can you explain why each line of code is necessary? What happens if you remove the `print()` after the inner loop?
4. Think about the checkerboard pattern. Why does `(row + col) % 2` create alternating colors? Can you think of another way?
5. If you had to explain nested loops to a friend using a real-world example (not programming), what would you choose?

Next week, we'll combine loops with conditions to create even more sophisticated programs that can filter data and make smart decisions while iterating!