مكتب

Quick Reference Notes 📖

Week 8: Multiple Paths with `if/else` and `elif`

7th Grade Computer Science

1 Introduction

1.1 This Week's Big Question

How can we make computers respond intelligently to many different situations? Last week, we learned to make simple yes/no decisions. This week, we'll create programs that can choose between multiple paths - like a GPS app that suggests different routes, or a game that responds differently to various player choices.

Prerequisites

Before starting this week, you should be comfortable with:

- Using comparison operators and logical operators (Week 5)
- Designing programs with simple decision-making (Week 6)
- Writing simple `if` statements (Week 7)

1.2 What You Already Know

You can write programs that check a condition and do something if it's True. You understand that Python uses indentation to group code together. You've practiced with simple decision-making programs like password checkers and age verifiers.

1.3 What You'll Be Able to Do

By the end of this week, you'll:

- Write programs that choose between two different actions using else
- Handle multiple possibilities with elif statements
- Create complex decision trees with nested conditions
- Combine multiple conditions using and/or logic
- Debug programs with multiple decision paths
- Build interactive programs like grade calculators and games

2 VIDEO 1: From if to if/else

2.1 Comparing Week 7 and Week 8

Last week, we learned the basic if statement - it was like a detour sign: “if road is blocked, take alternate route.” But what if the road isn’t blocked? We just continued normally. This week, we’re adding explicit instructions for both cases.

Week 7 style - only handles one case:

```python
age = 15
if age >= 18:
    print("You can vote!")
print("Have a nice day!")  # Always prints
```

Week 8 style - handles both cases explicitly:

```python
age = 15
if age >= 18:
    print("You can vote!")
else:
    print("You’ll be able to vote in", 18 - age, "years!")
print("Have a nice day!")  # Always prints
```

2.2 Why Use else?

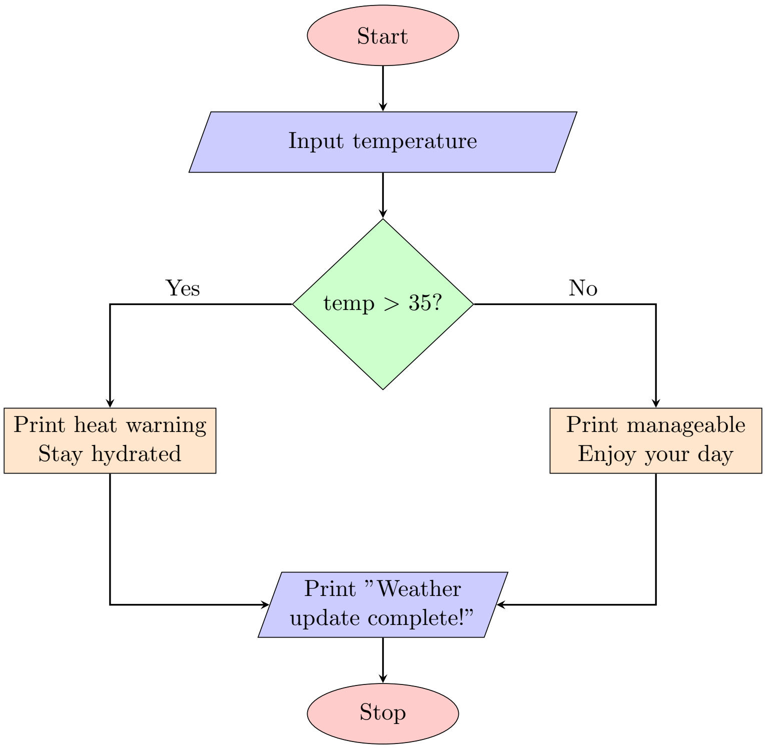
The `else` block ensures we always respond to the user - no silent failures! It makes our programs more complete and user-friendly.

2.3 Real-World Example: Smart Temperature Advisor

```python
temperature = int(input("What’s the temperature? "))

if temperature > 35:
    print("Heat warning! Stay hydrated.")
    print("Avoid outdoor activities 12-3 PM.")
else:
    print("Temperature is manageable.")
    print("Enjoy your day!")

print("Weather update complete!")  # Always runs
```



Rule

if/else Pattern:

```
if condition:
    # Code for when condition is True
    # Can have multiple lines
else:
    # Code for when condition is False
    # Also can have multiple lines
# Code here runs regardless
```

Key points:

- `else` must align with `if` (same indentation)
- `else` also needs a colon (:)
- You can only have one `else` per `if`

Quick Check

Convert this Week 7 code to use `if/else`. Add an `else` to tell user when password is wrong:

```
password = input("Enter password: ")
if password == "secret123":
    print("Access granted!")
```

3 VIDEO 2: Multiple Choices with elif

3.1 When Two Paths Aren't Enough

Real decisions often have more than two options. Think about grades: A, B, C, D, or F. That's five different paths! The elif statement (short for "else if") handles these multiple-choice situations.

3.2 Building a Grade Calculator

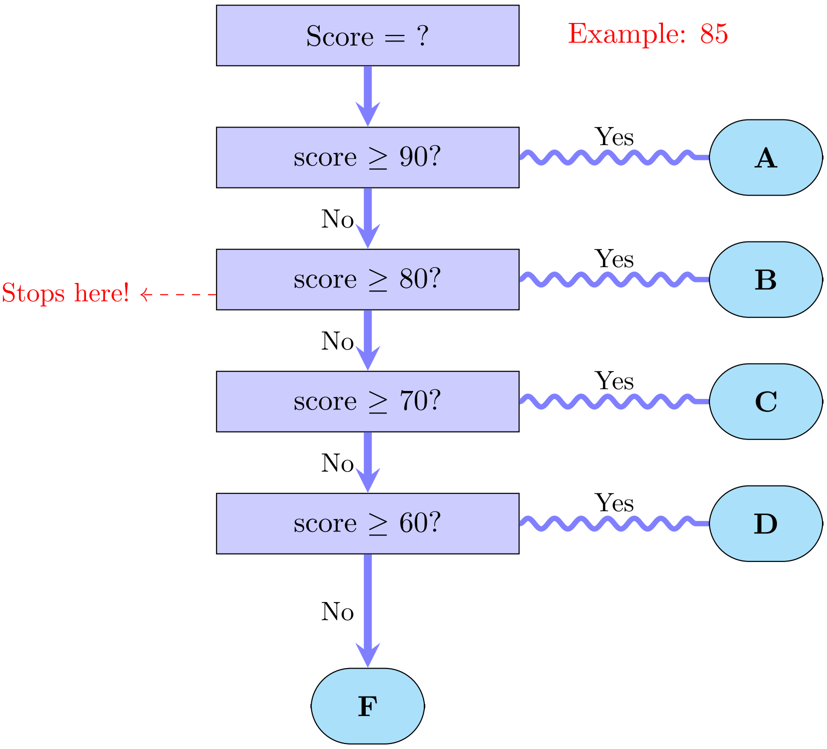
```python
score = int(input("Enter your test score: "))

if score >= 90:
    grade = "A"
    message = "Excellent work!"
elif score >= 80:
    grade = "B"
    message = "Good job!"
elif score >= 70:
    grade = "C"
    message = "Satisfactory."
elif score >= 60:
    grade = "D"
    message = "Needs improvement."
else:
    grade = "F"
    message = "Please see the teacher."
print(f"Grade: {grade}")
print(message)
```

3.3 How `elif` Works - The Waterfall Model

Think of `elif` chains like a waterfall with multiple levels:

1. Water (your program) starts at the top
2. At each level (condition), it checks: "Can I stop here?"
3. First True condition = water stops at that pool
4. If no conditions are True, water reaches the bottom (else)



3.4 The Key Rule: First True Wins!

Important: Once Python finds a `True` condition in an `if/elif` chain, it:

1. Executes that block of code
2. Skips all the remaining `elif` and `else` blocks
3. Continues with code after the entire `if/elif/else` structure

```python
# Demonstration: Only one block runs
score = 85

if score >= 90:          # score is 85, so False - skip this
    print("A")
elif score >= 80:        # score is 85, so True - Run this!
    print("B")
elif score >= 70:        # Never Checked - we already found a True
    print("C")
elif score >= 60:        # Never Checked
    print("D")
else:                    # Never Checked
    print("F")

print("Done grading")  # This always runs (outside the if/elif)
```

Let's trace through with score = 85:

- Check: 85 >= 90? **False** → Keep going
- Check: 85 >= 80? **True** → Print "B" and STOP CHECKING!
- The remaining conditions (>= 70, >= 60) are NEVER evaluated
- Python jumps directly to "Done grading"

3.5 Order Matters! A Common Mistake

Wrong - This gives everyone a D or F!

```python
score = 85
if score >= 60:
    print("D")  # 85 is >= 60, so this runs!
elif score >= 70:
    print("C")  # Never reached
elif score >= 80:
    print("B")  # Never reached
elif score >= 90:
    print("A")  # Never reached
```

Correct - Check from highest to lowest

```python
if score >= 90:
    print("A")
elif score >= 80:
    print("B")  # 85 lands here
...
```

⚠ Common Error

elif Order Rule: Always order your conditions from most specific to least specific!
- Number ranges: Usually highest → lowest
- String lengths: Usually longest → shortest
- Special cases: Check before general cases

☛ Quick Check

Arrange these lines in the correct order for a time-of-day greeting program:

```python
elif hour >= 17:
    print("Good evening!")
else:
    print("Good night!")
if hour >= 5 and hour < 12:
    print("Good morning!")
elif hour >= 12:
    print("Good afternoon!")
```

4 VIDEO 3: Decisions Within Decisions

4.1 Starting Simple: Can You Play Outside?

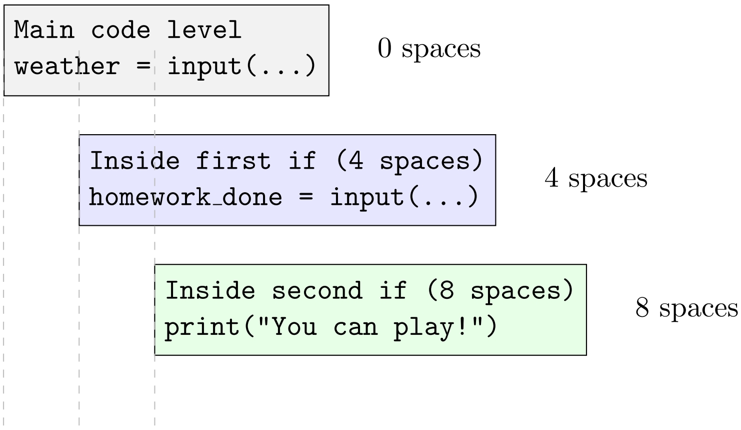
Sometimes one question leads to another. Let's say you want to play outside. First, you check if it's raining. If it's not raining, then you check if you finished your homework. That's nesting!

```python
# Simple nested decision
weather = input("Is it raining? (yes/no): ")

if weather == "no":
    # It's not raining, so check homework
    homework_done = input("Did you finish homework? (yes/no): ")
    if homework_done == "yes":
        print("You can play outside!")
    else:
        print("Finish homework first!")
else:
    print("It's raining - stay inside!")
```

4.2 Understanding Nesting with Indentation

See how the second if is inside the first if? The extra indentation (8 spaces total) shows it only happens when the first condition is True.



4.3 A School Cafeteria Example

Let's make it slightly more complex - checking if you can buy a special lunch:

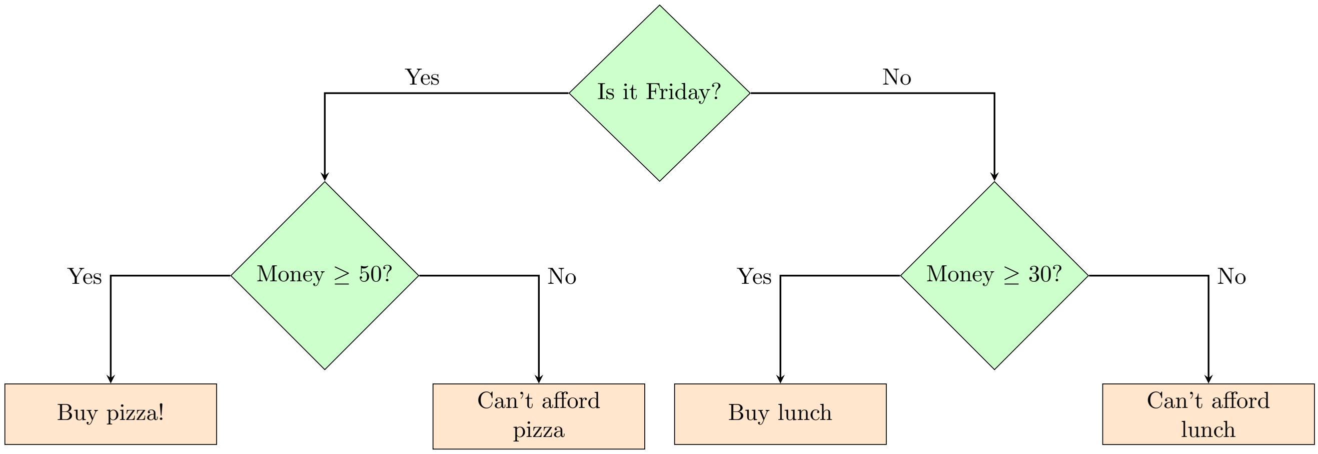
```python
# Cafeteria special lunch checker
day = input("What day is it? ")
money = int(input("How much money do you have? "))

if day == "Friday":
    print("It's Friday - pizza day!")
    if money >= 50:
        print("You can buy the pizza special!")
        print("Enjoy your lunch!")
    else:
        print("Sorry, pizza costs 50 rupees.")
        print("You have", money, "rupees.")
else:
    print("Regular menu today.")
    if money >= 30:
        print("You can buy lunch.")
    else:
        print("You need at least 30 rupees.")
```

Notice how we check the day first, then decide what to do based on the money available. If it's Friday, only then we check whether we can afford pizza or not. One condition has to be True before we check the next one.

4.4 Visualizing the Decision Flow

Think of nested ifs like a tree where each branch can split again:



4.5 When to Use Nested If

Use nested if when the second decision only makes sense after the first one:

- Check weather → then check if you have an umbrella
- Check age → then check height for a ride
- Check day → then check time for special events

Rule

Nested If Guidelines:

```python
if first_condition:
    # This runs if first is True
    # 4 spaces from the left!
    if second_condition:
        # This needs both to be True
        # 8 spaces from the left!
    else:
        # First True, second False
else:
    # First condition was False
    # Can have another if here too!
```

Remember: Each level adds 4 more spaces!

Tip

Keep It Simple!

- Don't go more than 2 levels deep
- If it gets confusing, draw it out first
- Test each path separately
- Add print statements to see which path your code takes

Quick Check

What will this print if has_permission is True and age is 9?

```python
if has_permission == True:
    if age >= 10:
        print("You can go!")
    else:
        print("Too young!")
else:
    print("Need permission first!")
```

5 VIDEO 4: Complex Logic with and/or

5.1 Combining Multiple Conditions

Real-world decisions often depend on multiple factors. Can you go to the park? Depends on: Is it daytime AND is the weather good AND do you have permission?

5.2 The ‘and’ Operator - All Must Be True

```python
# Club membership requirements
age = int(input("Age: "))
grade = int(input("Grade (0-100): "))
permission = input("Parent permission? (yes/no): ")

if age >= 13 and grade >= 80 and permission == "yes":
    print("Welcome to the Science Club!")
else:
    print("Sorry, you don’t meet all requirements.")
    # Let’s be helpful and explain why
    if age < 13:
        print("- Must be 13 or older")
    if grade < 80:
        print("- Need 80% average")
    if permission != "yes":
        print("- Need parent permission")
```

Sample Output:

```
Age: 12
Grade (0-100): 50
Parent permission? (yes/no): no
Sorry, you don’t meet all requirements.
- Must be 13 or older
- Need 80% average
- Need parent permission
```

5.3 The ‘or’ Operator - Any One Is Enough

```python
# Weekend detection - multiple ways to check
day = input("What day is it? ")

# Accept different formats
```

```python
if day == "Saturday" or day == "Sunday" or day == "Sat" or day == "Sun":
    print("It's the weekend!")
    print("No school today!")
else:
    print("It's a weekday.")
    print("Time for school!")
```

Sample Output:

```
What day is it? Saturday
It's the weekend!
No school today!
```

5.4 Truth Tables - See How They Work

Let's trace through some examples of how `and` and `or` work with different conditions. We'll use a simple table format to visualize the results:

Expression	age = 14	grade = 85	Result
age >= 13	True	–	True
grade >= 80	–	True	True
age >= 13 **and** grade >= 80	True	True	True
age >= 15 **and** grade >= 80	False	True	False
age >= 15 **or** grade >= 80	False	True	True

5.5 Common Mistakes with and/or

Wrong - Common English confusion:

```python
name = input("Enter name: ")
if name == "Ali" or "Ahmed":  # Always True!
    print("Welcome!")
```

Correct - Must repeat the comparison, check both variables:

```python
if name == "Ali" or name == "Ahmed":
    print("Welcome!")
```

Wrong - Impossible condition:

```python
age = 15
if age < 13 and age > 18:       # Can't be both!
    print("This never runs")
```

Correct - Did you mean or?

```python
if age < 13 or age > 18:
    print("Not a teenager")
```

Tip

Why is the condition in the line above:
`name == "Ali" or "Ahmed"`
always True?

Python evaluates this as: `(name == "Ali") or ("Ahmed")`

The key insight: In Python, any non-empty string is “truthy” (counts as True).

- `"Ahmed"` by itself → True (it’s a non-empty string)
- `"Hello"` by itself → True
- `""` (empty string) → False

So the expression becomes:

- If name is “Ali”: `True or True` → True
- If name is “Sara”: `False or True` → True
- If name is anything: `False or True` → True

Remember: Always repeat the comparison for each value. Check each variable separately!

Tip

Testing Complex Conditions: When debugging and/or:

- Test each part separately first
- Print the parts: `print(age >= 13, grade >= 80)`
- Use parentheses to make order clear
- Remember: `‘and’` needs **all** comparisons to be `True`, `‘or’` needs **any** `True`

Quick Check

What prints when `temp = 38` and `humid = 85`?

```python
if temp > 35 and humid > 80:
    print("Extreme weather alert!")
elif temp > 35 or humid > 80:
    print("Uncomfortable conditions")
else:
    print("Pleasant weather")
```

6 VIDEO 5: Summary and Debugging Strategies

6.1 Our Complete Decision-Making Toolkit

This week we expanded from simple if to a full toolkit:

Structure	When to Use	Example
if only	One action needed	Print warning if hot
if/else	Two alternatives	Login success or failure
if/elif/else	Multiple exclusive options	Grade calculation
Nested if	Decisions depend on other conditions	Movie ticket pricing
and	All conditions must be True	Club requirements
or	Any condition can be True	Weekend detection

6.2 Debugging Complex Conditions

When your conditions don't work as expected:

```python
# Strategy 1: Print the condition parts
age = 15
grade = 78
print(f"age >= 13: {age >= 13}")      # True
print(f"grade >= 80: {grade >= 80}")  # False
print(f"Both: {age >= 13 and grade >= 80}")  # False

# Strategy 2: Simplify complex conditions
# Instead of:
if (age >= 13 and grade >= 80) or (age >= 16):
    # Hard to debug!

# Try:
teen_with_grades = age >= 13 and grade >= 80
old_enough = age >= 16
if teen_with_grades or old_enough:
    # Much clearer!
```

6.3 Common Patterns Reference

```python
# Range checking
if low <= value <= high:  # Python special!
if value >= low and value <= high:  # Same thing

# Opposite conditions
if not (age < 18):  # Same as age >= 18
if age >= 18:       # Clearer!
```

Landmark Moment

You've mastered the art of decision-making in Python! From simple if/else to complex nested conditions with and/or logic, you can now write programs that respond intelligently to any situation. These tools are the foundation of all interactive programs!

7 Quick Reference

Quick Reference

Week 8 Decision Structures:

```python
# if/else - two paths
if condition:
    # True path
else:
    # False path

# elif - multiple exclusive paths
if condition1:
    # First case
elif condition2:
    # Second case
elif condition3:
    # Third case
else:
    # Default case

# Nested - decisions within decisions
if outer:
    if inner:
        # Both true

# Complex conditions
if cond1 and cond2:    # Both must be True
if cond1 or cond2:     # Either can be True
if (A and B) or C:     # Use parentheses to group

# Debugging helpers
print(f"Condition result: {condition}")
simplified = complex_part1 and complex_part2
if simplified:
    # Easier to debug
```

8 Reflection

Think about your learning journey this week:

1. When you first saw elif, what made sense immediately? What took more time to understand? How did you work through the confusion?

2. Think of a time this week when your code didn't work as expected. What was your debugging process? What helped you figure it out?

3. The order of elif conditions tripped up many students. When did this concept "click" for you? What example or explanation made it clear?

4. How do you feel about nested if statements? Do they make sense to you, or do you find them confusing? What strategies help you read nested code?

5. Looking at and/or operators - do you think in "English" first and then translate to Python, or has Python logic become natural to you?

6. What real-life decision would you like to turn into a Python program now that you know if/elif/else? Why would this be useful to automate?

7. Compare how you feel about making decisions in code now versus last week. What's changed in your confidence level? What do you want more practice with?

Next week, we'll learn about for loops - a powerful way to repeat code without copying and pasting!