Here are coding questions that focus specifically on if-else and if-else-if logic, categorized by difficulty.

Level 1: The Basics (Simple Conditions)

Goal: Get comfortable with basic comparison operators (>, <, ==, !=).

1. **Positive, Negative, or Zero:** Write a program that takes a number as input and prints whether it is positive, negative, or zero.
2. **Odd or Even:** Take an integer input and determine if it is odd or even.
3. **Voting Eligibility:** Ask the user for their age. If they are 18 or older, print "Eligible to vote"; otherwise, print "Not eligible."
4. **Pass or Fail:** Input a student's score (out of 100). If the score is 40 or above, print "Pass"; otherwise, print "Fail."
5. **Maximum of Two Numbers:** Take two numbers as input and print the larger one.

Level 2: Intermediate (Logical Operators)

Goal: Use AND (&&), OR (||), and NOT (!) to combine conditions.

6. Write a program to check if a given year is a leap year.

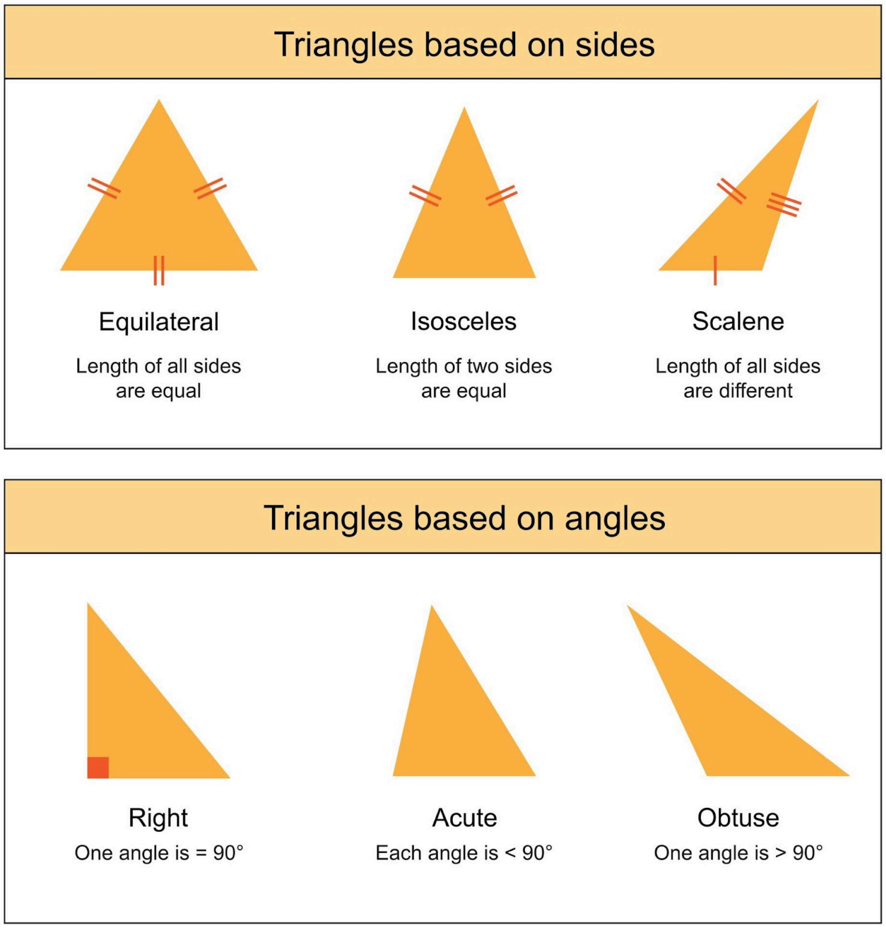
* Hint: A year is a leap year if it is divisible by 4 but not by 100, OR if it is divisible by 400.
7. Grade Calculator: Input a student's marks and assign a grade:
* 90-100: A
* 80-89: B
* 70-79: C
* 60-69: D
* Below 60: F
8. Largest of Three Numbers: Input three numbers and print the largest one using logical operators.
9. Vowel or Consonant: Input a single character. Check if it is a vowel (a, e, i, o, u) or a consonant. Bonus: Handle uppercase and lowercase inputs.
10. Triangle Validity: Input three angles of a triangle. Check if the triangle is valid (sum of angles = 180).

Level 3: Advanced (Nested If-Else)

Goal: Handle complex logic where decisions depend on previous decisions.

11. **Electricity Bill Calculator:** Calculate the bill based on units consumed:
    - First 100 units: $1/unit
    - Next 100 units: $2/unit
    - Above 200 units: $5/unit

12. **Triangle Type:**



Shutterstock

Input three sides of a triangle. First, check if it is a valid triangle (sum of any two sides > third side). If valid, determine if it is:
* Equilateral: All sides equal.
* Isosceles: Two sides equal.
* Scalene: No sides equal.

13. Login System: Ask for a username and password.
* If the username is correct, check the password.
* If the password is correct, print "Login Successful."
* If the password is wrong, print "Incorrect Password."
* If the username is wrong, print "User not found."

14. Quadratic Equation Roots: Input coefficients a, b, and c for the equation $ax^2 + bx + c = 0$. Calculate the discriminant ($D = b^2 - 4ac$).
* If $D > 0$: Two distinct real roots.
* If $D == 0$: One real root.
* If $D < 0$: No real roots (complex).

Level 4: Real-World Scenarios

Goal: Apply logic to practical problems.

15. **ATM Withdrawal:** Input a balance and an amount to withdraw.
    - If amount > balance, print "Insufficient Funds."
    - If amount <= 0, print "Invalid Amount."
    - Otherwise, subtract the amount and print the new balance.
16. **BMI Calculator:** Input weight (kg) and height (m). Calculate BMI ($\text{weight} / \text{height}^2$) and categorize:
    - < 18.5: Underweight
    - 18.5 - 24.9: Normal
    - 25 - 29.9: Overweight
    - 30+: Obese
17. **Shop Discount:** Input the total cost of items.
    - If cost > 1000, give a 10% discount.
    - If cost > 500 but < 1000, give a 5% discount.
    - Otherwise, no discount.
    - Print the final amount to be paid.

PUZZLE QUESTIONS

- (Take three numbers and check all the three numbers are in ascending order)

Here are three number-based logic puzzles that focus purely on if-else conditions. I have included the "Second Largest Number" puzzle you asked for, plus two others that test your numerical logic.

1. The "Middle Child" (Second Largest Number)

The Challenge:

Take three distinct numbers as input: a, b, and c. Find the number that is not the largest and not the smallest.

The Logic:

You need to identify the "middle" value.

- If a is larger than b but smaller than c (or vice versa), a is the answer.
- We can simplify this by first finding the largest, then comparing the remaining two. Or, checking all permutations.

Solution:

```python
a = int(input("Enter A: "))
b = int(input("Enter B: "))
c = int(input("Enter C: "))

if (a > b and a < c) or (a < b and a > c):
    print(f"{a} is the second largest")
elif (b > a and b < c) or (b < a and b > c):
    print(f"{b} is the second largest")
else:
    print(f"{c} is the second largest")
```

---

2. The "Closest to Target" Puzzle

The Challenge:

Input two numbers, num1 and num2. Print whichever number is closer to 10.

- *Constraint:* Do not use the abs() function. You must handle the distance logic manually.
- *Edge Case:* If they are equally close (e.g., 8 and 12), print "Tie".

The Logic:

You need to calculate the "distance" from 10.

- Distance = Number - 10.
- If the distance is negative (e.g., $8 - 10 = -2$), you must flip it to positive using if.

Solution:

```python
```

```python
n1 = int(input("Number 1: "))
n2 = int(input("Number 2: "))
target = 10

# Calculate distance for n1
dist1 = n1 - target
if dist1 < 0:
    dist1 = -dist1  # Flip negative to positive manually

# Calculate distance for n2
dist2 = n2 - target
if dist2 < 0:
    dist2 = -dist2

# Compare distances
if dist1 < dist2:
    print(f"{n1} is closer to 10")
elif dist2 < dist1:
    print(f"{n2} is closer to 10")
else:
    print("Both are equally close (Tie)")
```

---

3. The "Strictly Increasing" Digits

The Challenge:

Input a 3-digit number (e.g., 159). Check if its digits are in strictly ascending order (left to right).

- `159` $\rightarrow$ Yes ($1 < 5 < 9$)
- `121` $\rightarrow$ No ($1 < 2$, but $2 > 1$)
- `345` $\rightarrow$ Yes

The Logic:

You cannot iterate over the number. You must use math to extract the digits.

1. **Last digit (Units):** `num % 10`
2. **Middle digit (Tens):** `(num // 10) % 10`
3. **First digit (Hundreds):** `num // 100`

Solution:

Python

```python
num = int(input("Enter a 3-digit number: "))

# Extract Digits
# Example: 456
d3 = num % 10        # 6
d2 = (num // 10) % 10  # 5
d1 = num // 100       # 4

if d1 < d2 and d2 < d3:
    print("Digits are strictly increasing!")
else:
    print("Digits are NOT strictly increasing.")
```

Here are three more logic-heavy numerical puzzles that rely on careful use of `if-else` and basic math.

4. The "Interval Overlap" Puzzle

The Challenge:

Input two ranges of numbers (intervals).

- Interval 1: Starts at `a`, ends at `b`
- Interval 2: Starts at `c`, ends at `d`

Determine if these two intervals **overlap**.

- Example: (1, 5) and (4, 10) $\rightarrow$ Overlap (at 4 and 5).
- Example: (1, 5) and (6, 10) $\rightarrow$ No Overlap.

The Logic:

Instead of checking all the ways they do overlap, it is much easier to check the two ways they do NOT overlap.

1. Interval 1 ends before Interval 2 starts ($b < c$).
2. Interval 2 ends before Interval 1 starts ($d < a$).
   If neither of these is true, they must overlap.

Solution:

Python

```python
print("Enter Interval 1:")
a = int(input("Start: "))
b = int(input("End:   "))

print("Enter Interval 2:")
c = int(input("Start: "))
d = int(input("End:   "))

# Ensure start is less than end for both (Basic validation)
if a > b or c > d:
    print("Invalid intervals (Start must be < End)")
else:
    # Check for NO overlap
    if b < c or d < a:
        print("They do NOT overlap.")
    else:
        print("They Overlap.")
```

---

5. Rounding Without round()

The Challenge:

Input a floating-point number (e.g., 4.3, 4.7, 4.5). Round it to the nearest integer using only if-else and casting to int.

- **Rule:** If the decimal part is ≥ 0.5, round up. Otherwise, round down.

The Logic:

You can get the integer part of a number by casting it to an int (e.g., int(4.7) is 4).

Then, compare the original number with this integer part to find the decimal difference.

Solution:

Python

```python
num = float(input("Enter a float number: "))

integer_part = int(num)  # e.g., 4.7 becomes 4
decimal_part = num - integer_part  # e.g., 4.7 - 4 = 0.7

```

```python
if decimal_part >= 0.5:
    result = integer_part + 1
else:
    result = integer_part

print(f"Rounded Number: {result}")
```

---

6. The "Currency Denomination" Break-down

The Challenge:

Input an amount of money (e.g., $185). Calculate the minimum number of notes required to make that amount using standard denominations: $100, $50, $20, $10, $5, $1.

Why it's tricky:

You have to use the largest notes first to keep the count minimum. This is usually done with a loop, but doing it with if statements helps you understand the flow of remainder logic.

Solution:

Python

```python
amount = int(input("Enter amount: "))

# Initialize note counters
n100 = 0
n50 = 0
n20 = 0
n10 = 0
n5 = 0
n1 = 0

if amount >= 100:
    n100 = amount // 100  # How many 100s?
    amount = amount % 100 # What is left?

if amount >= 50:
    n50 = amount // 50
    amount = amount % 50

if amount >= 20:
```

```python
    n20 = amount // 20
    amount = amount % 20

if amount >= 10:
    n10 = amount // 10
    amount = amount % 10

if amount >= 5:
    n5 = amount // 5
    amount = amount % 5

if amount >= 1:
    n1 = amount  # The rest are $1 notes

print("Notes Breakdown:")
print(f"$100: {n100}")
print(f"$50:  {n50}")
print(f"$20:  {n20}")
print(f"$10:  {n10}")
print(f"$5:   {n5}")
print(f"$1:   {n1}")
```

Here are three more logic puzzles involving numbers and validity checks, solvable using only if-else.

7. Time Validity Checker

The Challenge:
Input three integers: hours, minutes, and seconds. Check if they represent a valid time in a digital 24-hour clock format (HH:MM:SS).

6. **Hours** must be between 0 and 23.
7. **Minutes** must be between 0 and 59.
8. **Seconds** must be between 0 and 59.

Why it's useful:
This teaches you input validation. In real-world software, you never trust user input. You must "sanitize" it before using it.
Solution:

Python

```python
h = int(input("Enter Hours: "))
m = int(input("Enter Minutes: "))
s = int(input("Enter Seconds: "))

# Check validity
if h >= 0 and h <= 23:
    if m >= 0 and m <= 59:
        if s >= 0 and s <= 59:
            print("Valid Time")
        else:
            print("Invalid Seconds")
    else:
        print("Invalid Minutes")
else:
    print("Invalid Hours")
```

---

8. Point in Quadrant (Geometry Logic)

The Challenge:
Take two numbers as input: x and y (coordinates of a point). Determine which quadrant the point lies in.Getty Images
The Logic:
You have to handle the "Standard" quadrants (1, 2, 3, 4) plus the "Edge Cases" (Axes and Origin).

7. **Origin:** (0, 0)
8. **X-Axis:** y is 0, x is not.
9. **Y-Axis:** x is 0, y is not.
10. **Q1:** (+, +), **Q2:** (-, +), **Q3:** (-, -), **Q4:** (+, -)

Solution:

Python

```python
x = int(input("Enter x coordinate: "))
y = int(input("Enter y coordinate: "))

if x == 0 and y == 0:
```

```python
    print("Point is at the Origin")
elif x == 0:
    print("Point is on the Y-Axis")
elif y == 0:
    print("Point is on the X-Axis")
elif x > 0 and y > 0:
    print("Quadrant I")
elif x < 0 and y > 0:
    print("Quadrant II")
elif x < 0 and y < 0:
    print("Quadrant III")
else:
    print("Quadrant IV")
```

---

9. The 3-Digit Palindrome

The Challenge:
Input a 3-digit number. Check if it is a palindrome (reads the same forwards and backwards, e.g., 121, 353, 909).

13. **Constraint:** You cannot treat the input as a string. You must use math logic.

Why it's tricky:
You have to extract the first and last digits using math operations and compare them. For a 3-digit number, the middle digit doesn't actually affect whether it's a palindrome (only the first and last matter).
Solution:

Python

```python
num = int(input("Enter a 3-digit number: "))

# Validation to ensure it's 3 digits
if num < 100 or num > 999:
    print("Please enter a valid 3-digit number.")
else:
    # Extract first digit (Hundreds place)
    first_digit = num // 100

    # Extract last digit (Units place)
```

```
    last_digit = num % 10

    if first_digit == last_digit:
        print(f"{num} is a Palindrome")
    else:
        print(f"{num} is NOT a Palindrome")
```

Here are two more tricky logic questions. The first is the Progressive Tax Calculator, which is a classic "real-world" logic problem. The second is a game logic simulator.

10. The Progressive Tax Calculator

The Challenge:

Calculate the income tax based on the following rules (this is how real taxes work—it is "progressive," not flat):

- **First $10,000 of income: 0% Tax**
- **Next $20,000 (income between $10,001 - $30,000): 10% Tax**
- **Any income above $30,000: 20% Tax**

Why it's tricky:

If someone earns $45,000, you cannot simply calculate $45,000 \times 20\%$.

- **You must tax the first chunks of money at their specific lower rates, and only the *remaining* amount at the higher rate.**

The Logic (Example for $45,000):

1. **First $10k $\rightarrow$ 0 tax.**
2. **Next $20k $\rightarrow$ 10% of $20,000 = \$2,000$.**
3. **Remaining $15k ($45k - 30k$) $\rightarrow$ 20% of $15,000 = \$3,000$.**
4. **Total Tax: $\$5,000$.**

Solution:

Python

```
income = float(input("Enter your annual income: "))
```

```python
tax = 0

if income <= 10000:
    tax = 0
elif income <= 30000:
    # Tax on income above 10,000
    tax = (income - 10000) * 0.10
else:
    # Tax on the chunk between 10k and 30k (which is 20k total)
    # 20,000 * 0.10 = 2,000
    # PLUS tax on the remaining amount above 30k
    tax = 2000 + (income - 30000) * 0.20

print(f"Total Tax Payable: ${tax}")
```

---

11. Rock, Paper, Scissors (The Logic Tree)

The Challenge:

Simulate a game of Rock, Paper, Scissors between two players.

- Input Player 1's choice ("R", "P", "S").
- Input Player 2's choice ("R", "P", "S").
- Print the winner or if it's a draw.

Why it's tricky:

You have to handle every winning condition without writing messy code. You must check for a Tie first to save time, then check specific winning scenarios.

Solution:

```python
p1 = input("Player 1 (R/P/S): ")

p2 = input("Player 2 (R/P/S): ")


if p1 == p2:
    print("It's a Draw!")
elif p1 == "R":
    if p2 == "S":
        print("Player 1 Wins (Rock smashes Scissors)")
    else: # p2 is P
        print("Player 2 Wins (Paper covers Rock)")
elif p1 == "P":
    if p2 == "R":
        print("Player 1 Wins (Paper covers Rock)")
    else: # p2 is S
        print("Player 2 Wins (Scissors cuts Paper)")
elif p1 == "S":
    if p2 == "P":
        print("Player 1 Wins (Scissors cuts Paper)")
    else: # p2 is R
        print("Player 2 Wins (Rock smashes Scissors)")
else:
    print("Invalid Input! Use only R, P, or S.")
```

12. The "Shop Inventory" Restock Alarm

The Challenge:

A shopkeeper needs an automated message for restocking items.

Input: current_stock (int) and is_holiday_season (boolean - enter 1 for Yes, 0 for No).

Rules:

1. **If it is Holiday Season: Restock if stock is below 50.**
2. **If it is Not Holiday Season: Restock only if stock is below 20.**
3. **If stock is 0, print "Out of Stock! Urgent!" regardless of the season.**

Solution:

```python
stock = int(input("Current Stock: "))

season_input = int(input("Is it holiday season? (1 for Yes, 0 for No): "))


# Convert integer input to boolean

if season_input == 1:

  is_holiday = True

else:

  is_holiday = False


if stock == 0:

  print("Out of Stock! Urgent!")

elif is_holiday:

  if stock < 50:

    print("Stock is low (Holiday rule). Order more.")

  else:
```

```
        print("Stock is sufficient.")
else:
    if stock < 20:
        print("Stock is low (Standard rule). Order more.")
    else:
        print("Stock is sufficient.")
```

Here are two more logic-heavy coding questions using `if-else`. The first one combines multiple criteria (performance + experience), and the second involves "soft" criteria like special quotas.

13. The Salary Bonus Calculator

The Challenge:

A company calculates year-end bonuses based on Years of Experience and Performance Rating (1 to 5).

Input: salary, experience, and rating.

The Rules:

1. Top Performers (Rating $\ge$ 4):
   - If experience > 10 years: 20% Bonus
   - If experience ≤ 10 years: 10% Bonus
2. Average Performers (Rating = 3):
   - If experience > 10 years: 10% Bonus
   - If experience ≤ 10 years: 5% Bonus
3. Low Performers (Rating < 3):
   - No Bonus.

Why it's tricky:

You have a "matrix" of decisions. You must first filter by rating (the most critical factor), and then nest the experience check inside.

Solution:

```python
salary = float(input("Enter Salary: "))

years = int(input("Years of Experience: "))

rating = int(input("Performance Rating (1-5): "))


bonus_percentage = 0


if rating >= 4:

    if years > 10:

        bonus_percentage = 20

    else:

        bonus_percentage = 10

elif rating == 3:

    if years > 10:

        bonus_percentage = 10

    else:

        bonus_percentage = 5

else:

    # Rating is 1 or 2

    bonus_percentage = 0
```

```
bonus_amount = (salary * bonus_percentage) / 100

print(f"Bonus: {bonus_percentage}%")

print(f"Bonus Amount: ${bonus_amount}")

print(f"Total Salary: ${salary + bonus_amount}")
```

---

14. The University Admission System (Sports Quota)

The Challenge:

Determine if a student gets admitted to a prestigious university.

Input: math_score, science_score, and has_sports_quota (Enter 'Y' or 'N').

The Rules:

1. **Direct Merit: If both Math and Science are ≥ 90, they qualify.**
2. **Total Cutoff: If the sum of Math + Science ≥ 160, they qualify.**
3. **Sports Quota Relaxation: If they have a sports quota, they only need a sum of ≥ 120 to qualify.**
4. **Rejection: If none of these are met, they are rejected.**

Why it's tricky:

You have to check the "Strict" rules first. If those fail, you check the "Relaxed" (Quota) rule. The order ensures that a high-scoring student who also has a sports quota isn't accidentally processed as just a "Quota" student (though the result is the same, the logic path is cleaner).

Solution:

```python
math = int(input("Math Score: "))

science = int(input("Science Score: "))

quota = input("Do you have Sports Quota? (Y/N): ")
```

```python
total = math + science

if math >= 90 and science >= 90:
   print("Admitted (Merit: High Subject Scores)")
elif total >= 160:
   print("Admitted (Merit: High Total)")
elif quota == 'Y':
   if total >= 120:
      print("Admitted (Sports Quota Relaxation)")
   else:
      print("Rejected (Score too low even for Quota)")
else:
   print("Rejected")
```

Summary of what you have practiced so far:

- Basic Logic: Odds/Evens, Positive/Negative.
- Logical Operators: Leap years, Grades.
- Nested Logic: Tax brackets, Electricity bills.
- Mathematical Logic: Palindromes, overlapping intervals, finding the middle number.
- Complex Rules: Bonuses and Quotas.

Next Step:

Would you like to try a "Traffic Light Controller" simulation? (e.g., If light is Green, Go. If Yellow, check distance...)?

Here are two more real-world logic scenarios. The first involves safety decisions (Traffic Light), and the second involves financial risk assessment (Loan Approval).

15. The Smart Traffic Light Controller

The Challenge:

Simulate a self-driving car's decision at a traffic light.

Input: light_color ("Green", "Yellow", "Red") and distance_to_intersection (in meters).

The Rules:

1. **Green: Always "Go".**
2. **Red: Always "Stop".**
3. **Yellow: This is the tricky part (The "Dilemma Zone").**
   - **If the car is less than or equal to 10 meters away, it cannot stop safely in time $\rightarrow$ "Cross CAREFULLY."**
   - **If the car is more than 10 meters away $\rightarrow$ "Slow Down and STOP."**

Why it's tricky:

You have to handle the specific edge case of the Yellow light. A simple "Stop if Yellow" is dangerous in real life because of inertia.

Solution:

```python
color = input("Light Color (Green/Yellow/Red): ")

distance = float(input("Distance to intersection (meters): "))


if color == "Green":

    print("Action: GO")

elif color == "Red":

    print("Action: STOP")

elif color == "Yellow":

    if distance <= 10:
```

```
        print("Action: CROSS CAREFULLY (Too close to stop)")
    else:
        print("Action: SLOW DOWN AND STOP")
else:
    print("Invalid Light Color")
```

---

16. Bank Loan Approval System (Debt-to-Income Ratio)

The Challenge:

Determine if a person qualifies for a loan based on their financial health.

Input: monthly_income, monthly_debt, and credit_score.

The Rules:

1. **Credit Score Check: If credit score is below 650, reject immediately.**
2. **Debt-to-Income (DTI) Ratio:**
   - **Calculate DTI: `(monthly_debt / monthly_income) * 100`.**
   - **If DTI is above 40%, reject (Too much existing debt).**
3. **Approval: If Score ≥ 650 AND DTI $\le 40\%$, approve the loan.**

Why it's tricky:

You must calculate a derived value (DTI) inside the logic flow. Also, checking the credit score first is more efficient (if the score is bad, why bother doing the math?).

Solution:

```python
income = float(input("Monthly Income: "))

debt = float(input("Monthly Debt Payment: "))

score = int(input("Credit Score: "))


# Check Credit Score First
```

```python
if score < 650:
    print("Loan Rejected: Credit Score too low.")
else:
    # Calculate DTI Ratio
    dti = (debt / income) * 100

    print(f"Your Debt-to-Income Ratio is: {dti}%")

    if dti > 40:
        print("Loan Rejected: High Debt-to-Income Ratio.")
    else:
        print("Loan Approved!")
```