1. Maximum of Three:

- **Problem:** Write a program that takes three integers as input and prints the largest of the three.
- **Example:**
  - Input: 5, 12, 3
  - Output: 12
- **Hint:** Use `if`, `elif`, and `else` to compare the numbers. You might need to use the `and` operator.

2. Triangle Type:

- **Problem:** Write a program that takes three side lengths of a triangle as input. Determine if the triangle is equilateral (all sides equal), isosceles (two sides equal), or scalene (no sides equal).
- **Example:**
  - Input: 5, 5, 5
  - Output: Equilateral
- **Hint:** Use `if`, `elif`, and `else` to compare the side lengths.

3. Quadrant Finder:

- **Problem:** Write a program that takes the x and y coordinates of a point as input. Print the quadrant in which the point lies (I, II, III, or IV). Assume the point will not lie on an axis.
- **Example:**
  - Input: x = 2, y = -3
  - Output: IV (fourth quadrant)
- **Hint:** Remember the signs of x and y in each quadrant:
  - I (first quadrant): x > 0, y > 0
  - II (second quadrant): x < 0, y > 0
  - III (third quadrant): x < 0, y < 0
  - IV (fourth quadrant): x > 0, y < 0

4. Change Calculator:

- **Problem:** Write a program that calculates the minimum number of coins needed to make change for a given amount in cents (under 100 cents). Use quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent).
- **Example:**
  - Input: 68 cents
  - Output: 2 quarters, 1 dime, 1 nickel, 3 pennies
- **Hint:** Use integer division (`//`) and the modulo operator (`%`) to find the number of each coin.

5. Weekday Calculator:

- **Problem:** Write a program that takes a number from 1 to 7 as input, representing the days of the week (1 for Monday, 2 for Tuesday, etc.). Print the corresponding day's name.
- **Example:**
  - Input: 4
  - Output: Thursday
- **Hint:** Use `if`, `elif`, and `else` to match the number to the day.

These exercises focus on applying conditional statements to basic mathematical concepts. They encourage problem-solving and logical thinking using Python.

----

4. Change Calculator:

- **Problem:** Write a program that calculates the minimum number of coins needed to make change for a given amount in cents (under 100 cents). Use quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent).
- **Example:**
  - Input: 68 cents
  - Output: 2 quarters, 1 dime, 1 nickel, 3 pennies
- **Hint:** Use integer division (`//`) and the modulo operator (%) to find the number of each coin.

Give the solution for problem number 4

```
amount = int(input("Enter the amount in cents (under 100): "))

quarters = amount // 25
amount = amount % 25

dimes = amount // 10
amount = amount % 10

nickels = amount // 5
amount = amount % 5

pennies = amount

print("Change:")
print("Quarters:", quarters)
print("Dimes:", dimes)
print("Nickels:", nickels)
print("Pennies:", pennies)
```

Explanation:

1. **Input:** The program first takes the amount in cents as input from the user and stores it in the `amount` variable.

2. **Calculating Quarters:**

    - `quarters = amount // 25`: This line uses integer division (`//`) to find the maximum number of quarters that can fit into the given amount. Integer division gives the whole number result of the division, discarding any remainder.
    - `amount = amount % 25`: This line updates the `amount` variable by calculating the remainder after dividing by 25 (using the modulo operator `%`). This remainder represents the amount left after using as many quarters as possible.

3. **Calculating Dimes, Nickels, and Pennies:** The program repeats the same process as above to calculate the number of dimes, nickels, and pennies, updating the `amount` each time to reflect the remaining amount.

4. **Output:** Finally, the program prints the calculated number of each coin.

Example Run:

If the user enters 68 cents, the output will be:

```
Change:
Quarters: 2
Dimes: 1
Nickels: 1
Pennies: 3
```

Does this logic always work or can it fail for some cases?

You're right to be curious about whether the logic always works! The good news is that yes, this logic will always work for any amount in cents under 100.

Here's why:

- **Greedy Approach:** The algorithm uses a "greedy" approach. This means it always tries to use the largest possible coin denomination first (quarters, then dimes, then nickels, then pennies).

- **US Currency System:** The denominations in the US currency system (25, 10, 5, 1) are designed in a way that this greedy approach guarantees the minimum number of coins. There's no situation where it would be better to use more smaller coins instead of fewer larger coins.

Let's test it with an edge case:

Imagine the amount is 41 cents.

- The algorithm would first use 1 quarter (25 cents), leaving 16 cents.
- Then it would use 1 dime (10 cents), leaving 6 cents.
- Then it would use 1 nickel (5 cents), leaving 1 cent.
- Finally, it would use 1 penny.

This results in 1 quarter, 1 dime, 1 nickel, and 1 penny – the minimum number of coins possible.

In Summary: The combination of the greedy approach and the structure of the US currency system ensures that this logic will always find the optimal solution for making change with the minimum number of coins.

Can you give an example with a different set of coin denominations from the US system, where this greedy approach fails

You're thinking critically about how this algorithm works, and that's excellent! You're right, the greedy approach doesn't *always* guarantee the optimal solution with different coin systems. Here's an example:

Coin System: Imagine a currency with these denominations:

- 12-cent coin
- 5-cent coin
- 1-cent coin

Scenario: Let's say you need to make change for 15 cents.

- **Greedy Approach:**
    - The greedy approach would first use one 12-cent coin, leaving 3 cents to be paid with 3 one-cent coins. This results in a total of **4 coins**.
- **Optimal Solution:**
    - The optimal solution would be to use three 5-cent coins, resulting in a total of **3 coins**.

Why the Greedy Approach Fails Here:

The greedy approach fails because it prioritizes the largest coin (12-cent coin) without considering the overall optimal combination. In this case, using the smaller 5-cent coins multiple times is more efficient.

Key Takeaway:

The success of the greedy approach depends on the specific set of coin denominations. It works for the US system due to the relationship between the denominations, but it's not a universally optimal solution for all coin systems.