DRY Principle

- stands Don't Repeat Yourself.
- it is a common practice that developers use while writing codes to make sure they don't have multiple pieces of code doing same thing.
- when developers follow DRY they will isolate the piece of logic and while code for it in one place where it will be accessable to whatever else needs the logic. This means that `any other place that needs the same logic can reuse the code without rewriting the code.`

- Example:

    - eg: when you have a menu instead of distributing an individual copy to everyone just put it up on the wall & let everyone order from there.

Statement:

- A principle of software development, aimed at reducing repetition of information of all kinds.

Code:

Example 1:

- Without Dry Principle:

```
#include <iostream>
using namespace std;

int main() {
    // Calculate area of a rectangle
    int length = 5, width = 10;
    cout << "Area of rectangle: " << length * width << endl;

    // Calculate area of a square
    int side = 7;
    cout << "Area of square: " << side * side << endl;

    // Calculate area of a circle
    double radius = 4.5;
    cout << "Area of circle: " << 3.14159 * radius * radius << endl;

    return 0;
}
```

```
Problem: The area calculation logic for each shape is repeated, making the code
harder to maintain and prone to errors.
```

- With DRY Principle:

```
#include <iostream>
#include <cmath> // For mathematical functions like pow
using namespace std;

// Function to calculate area of rectangle
int calculateRectangleArea(int length, int width) {
    return length * width;
}

// Function to calculate area of square
int calculateSquareArea(int side) {
    return side * side;
}

// Function to calculate area of circle
double calculateCircleArea(double radius) {
    return M_PI * radius * radius; // M_PI is the value of π from <cmath>
}

int main() {
    cout << "Area of rectangle: " << calculateRectangleArea(5, 10) << endl;
    cout << "Area of square: " << calculateSquareArea(7) << endl;
    cout << "Area of circle: " << calculateCircleArea(4.5) << endl;

    return 0;
}
```

- Benefits of DRY in this Example:

  1. Reusability:

  The area calculation logic for each shape is encapsulated in a reusable function.

  2. Maintainability:

  If a formula needs to change, you only need to update it in the respective function.

  3. Readability:

  The main function becomes cleaner and more readable.

Example 2:

- Without DRY Principle:

```
# Calculate total price with tax for multiple items
# Item 1
```

```
    price1 = 100
    tax_rate = 0.15
    total_price1 = price1 + (price1 * tax_rate)
    print(f"Total price for item 1: {total_price1}")

    # Item 2
    price2 = 200
    total_price2 = price2 + (price2 * tax_rate)
    print(f"Total price for item 2: {total_price2}")

    # Item 3
    price3 = 50
    total_price3 = price3 + (price3 * tax_rate)
    print(f"Total price for item 3: {total_price3}")
```

```
 Problem: The logic for calculating the total price is repeated for each item,
making the code repetitive and harder to maintain.
```

- With DRY Principle:

```
    # Function to calculate total price with tax
    def calculate_total_price(price, tax_rate):
        return price + (price * tax_rate)

    # Items and tax rate
    tax_rate = 0.15
    prices = [100, 200, 50]

    # Initialize item counter
    item_number = 1

    # Calculate and display total price for each item
    for price in prices:
        total_price = calculate_total_price(price, tax_rate)
        print(f"Total price for item {item_number}: {total_price}")
        item_number += 1
```

Why Write DRY Code:

1. Maintainability:

- DRY code is easier to maintain because changes or updates only need to be made in one place, reducing the risk of inconsistencies.

2. Readability:

- DRY promotes cleaner and more readable code by eliminating unnecessary repetition, making it easier for developers to understand the logic.

3. Efficiency:

- Reusing existing code instead of duplicating it saves development time and effort, resulting in more efficient and productive workflows.

4. Consistency:

- DRY ensures consistent behavior throughout the codebase, as a specific piece of logic exists in only one place.

5. Reduced Bugs:

- DRY reduces the chances of introducing bugs or errors caused by inconsistent changes in duplicated code.

Advantage of DRY Code:

1. Code Reusable
2. Easier to maintain and update existing code
3. Make code more readable
4. Consistency
5. Reduced development time and effort
6. Avoidance of errors