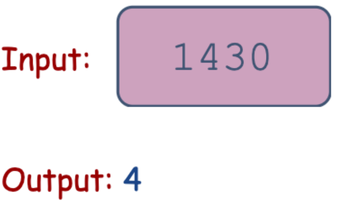
The key to transforming iterative code to recursive code is to find the specific lines of code that get transformed between the two implementations.

Count Number of Digits in a Number

When given a number, the code must print how many digits are present in that number. The two implementations of the same code are shown below. First, look through the two codes and spot the differences. Then read the explanation to better understand this transformation process.

The illustration below explains this concept.



The code below shows the two types of code followed by an explanation.

The Code

Iterative | Recursive

```
class IterativeClass {
    public static int countDigits(int num) {
      int count = 0;
      while (num > 0) {
        num = num / 10;
        count++;
      }
      return count;
    }

    public static void main( String args[] ) {

        int input = 1435043;
        int numDigits = countDigits(input);
        System.out.println("Number of digits in " + input + " = " + numDigits);
    }
}
```

Run | Save | Reset

Transforming the Code

First Step

The first step is to identify the **loop** in the iterative code. This loop has a special feature: it **modifies** an integer variable, which will eventually be returned by the method.

From **line 4 to line 8**, this encompasses the **single** while-loop- in our code, and the `return` statement, which will return our output. Note that the **modification** happens in the loop from **line 4 to line 7**. This is the loop that we will be dealing with.

Second Step

The second step is that the **loop condition** in this loop then becomes the **base case** in the *recursive code* of the method defined.

Look at **line 3** of the *recursive code*. This defines the base case of our code where it says that the method should return 1 when the `num` is less than or equal to 1. Alternatively, in the case of the *iterative code*, the while loop keeps running as long as `num` is greater than 0 where the variable `num` is modified inside the loop. Both these conditions serve as a termination of the method, eventually returning the output.

On **line 5 and line 6** in the *iterative code*, `num` is divided by 10, and the `count` variable is incremented by 1. Dividing the `num` by 10 essentially gets rid of the last digit of `num` and then increments the `count` by 1 in order to count the single digit that we just got rid of. Similarly, in the *recursive code* on **line 7**, this part of code returns 1 and then recursively calls the `countDigits` method with the argument `n/10` so that all digits are counted until the base condition is true.