1 Genetic Algorithm (GA) - Mutation and Recombination

Mutation: Consider a string of text instead of a strain of DNA. When that DNA is selected from generation to generation, it has—for the purpose of this exercise—a 1/100 chance to have a mutated gene. Examine the following algorithm and write a corresponding Python code that follows the logic.

Algorithm 1: Mutation of a String with 1/100 Chance

```
1 Function mutateString($DNA$)
2     Get a random ASCII character or symbol $\alpha$
3     Create a random integer $\gamma$ between 0 and the length of $DNA$
4     Create a random integer $\delta$ between 1 and 100
5     if $\delta == 1$ then
6         Replace the character of $DNA$ at position $\gamma$ with $\alpha$
7     Return $DNA$
```

In the above code, the *whole DNA has a 1% chance* to change *one gene.* Line 5 of the code above essentially does this by checking for the same number each turn. This is set to 1 for readability, but can be any number between 1 and 100, as long as it stays fixed.

How can you change the code so *each gene has a 1% chance* of mutating instead of once for the whole DNA? Think of this problem as a gene-by-gene alteration with introduction of random elements.

Recombination: DNA recombination (or crossover) can occur in nature, e.g., with the help of reproduction. We mimic this strategy by scrambling two DNA strains at a random position. The following few lines describe the mechanism. Write a function that takes two strings as arguments for the function and returns two strings with recombined sections.

Algorithm 2: Recombination of Two Strings

```
Function recombination($DNA1, DNA2$)
    Create a random integer $\gamma$ between 0 and the length of $DNA1$ (or $DNA2$)
    Assign to $newDNA1$ a new string; DNA1[0:$\gamma$] + DNA2[$\gamma$:end]
    Assign to $newDNA2$ a new string; DNA2[0:$\gamma$] + DNA1[$\gamma$:end]
    Return $(newDNA1, newDNA2)$
```