**PRACTICAL: 4**

**Aim:** Implement different crossover and Mutation Methods of GA in Python.

**Code: Single Point Crossover**

def single\_point\_crossover(parent1, parent2):

    point = random.randint(1, len(parent1)-1)

    return parent1[:point] + parent2[point:], parent2[:point] + parent1[point:]

p1 = [1,1,0,0]

p2 = [0,1,0,1]

print("String 1:", p1)

print("String 2:", p2)

print("After single-point crossover:",single\_point\_crossover(p1, p2))

**Output:**

**A close up of a text

AI-generated content may be incorrect.**

**Code: Two-Point Crossover**

def two\_point\_crossover(parent1, parent2):

    p1, p2 = sorted(random.sample(range(1, len(parent1)-1), 2))

    return (parent1[:p1] + parent2[p1:p2] + parent1[p2:],

            parent2[:p1] + parent1[p1:p2] + parent2[p2:])

x1 = [1,1,0,0,1,1]

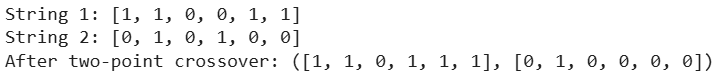
x2 = [0,1,0,1,0,0]

print("String 1:", x1)

print("String 2:", x2)

print("After two-point crossover:",two\_point\_crossover(x1, x2))

**Output:**

****

**Code: Uniform Crossover**

def uniform\_crossover(parent1, parent2, prob=0.5):

    child1, child2 = [], []

    for i in range(len(parent1)):

        if random.random() < prob:

            child1.append(parent1[i])

            child2.append(parent2[i])

        else:

            child1.append(parent2[i])

            child2.append(parent1[i])

    return child1, child2

s1 = [1,1,0,0,1,1]

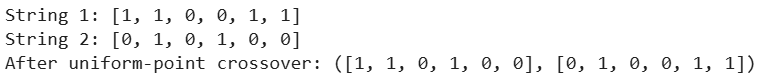
s2 = [0,1,0,1,0,0]

print("String 1:", s1)

print("String 2:", s2)

print("After uniform-point crossover:",uniform\_crossover(s1,s2))

**Output:**

****

**Code: Bit Flip Mutation**

def bit\_flip\_mutation\_k(individual, k=2):

    mutated = individual[:]

    # Choose k unique positions

    positions = random.sample(range(len(individual)), k)

    for i in positions:

        mutated[i] = 1 - mutated[i]  # flip bit

    return mutated

chromosome = [0, 1, 0, 1, 1, 0, 0]

print("Original:", chromosome)

print("Mutated :", bit\_flip\_mutation\_k(chromosome, k=3))

**Output:**

**A number with a punctuation mark

AI-generated content may be incorrect.**

**Code: Swap Mutation**

def swap\_mutation(individual):

    mutated = individual[:]

    # Select two distinct random positions

    i, j = random.sample(range(len(individual)), 2)

    # Swap the genes

    mutated[i], mutated[j] = mutated[j], mutated[i]

    return mutated

chromosome = [0, 1, 0, 1, 1, 0, 0]

print("Original:", chromosome)

print("Mutated :", swap\_mutation(chromosome))

**Output:**

**A number symbols on a white background

AI-generated content may be incorrect.**

**Code: Inverse Mutation**

def inverse\_mutation(individual):

    mutated = individual[:]

    i, j = sorted(random.sample(range(len(individual)), 2))

    # Reverse the subsequence

    mutated[i:j+1] = reversed(mutated[i:j+1])

    return mutated

chromosome = [1, 2, 3, 4, 5, 6, 7]

print("Original:", chromosome)

print("Mutated :", inverse\_mutation(chromosome))

**Output:**

**A number with black text

AI-generated content may be incorrect.**