A Gentle Introduction to Genetic Algorithms with Python and DEAP

Ruopeng An, Ph.D., MPP, FACE, FAAHB
Associate Professor
Faculty Lead in Public Health Sciences
Faculty Fellow in AI Innovations for Education
Lead, Center of Diabetes Translation Research AI Core
Washington University in St. Louis

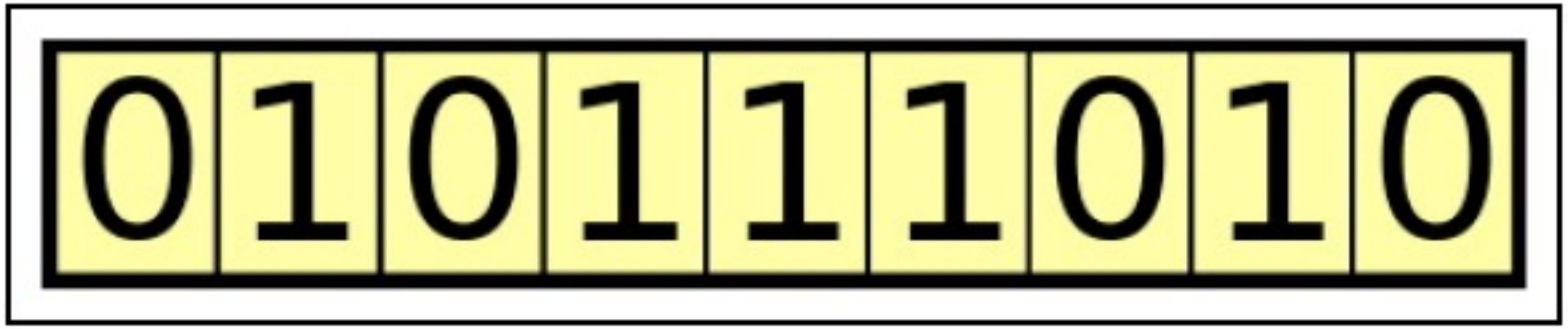
Darwinian Evolution

- Variation
- Inheritance
- Selection

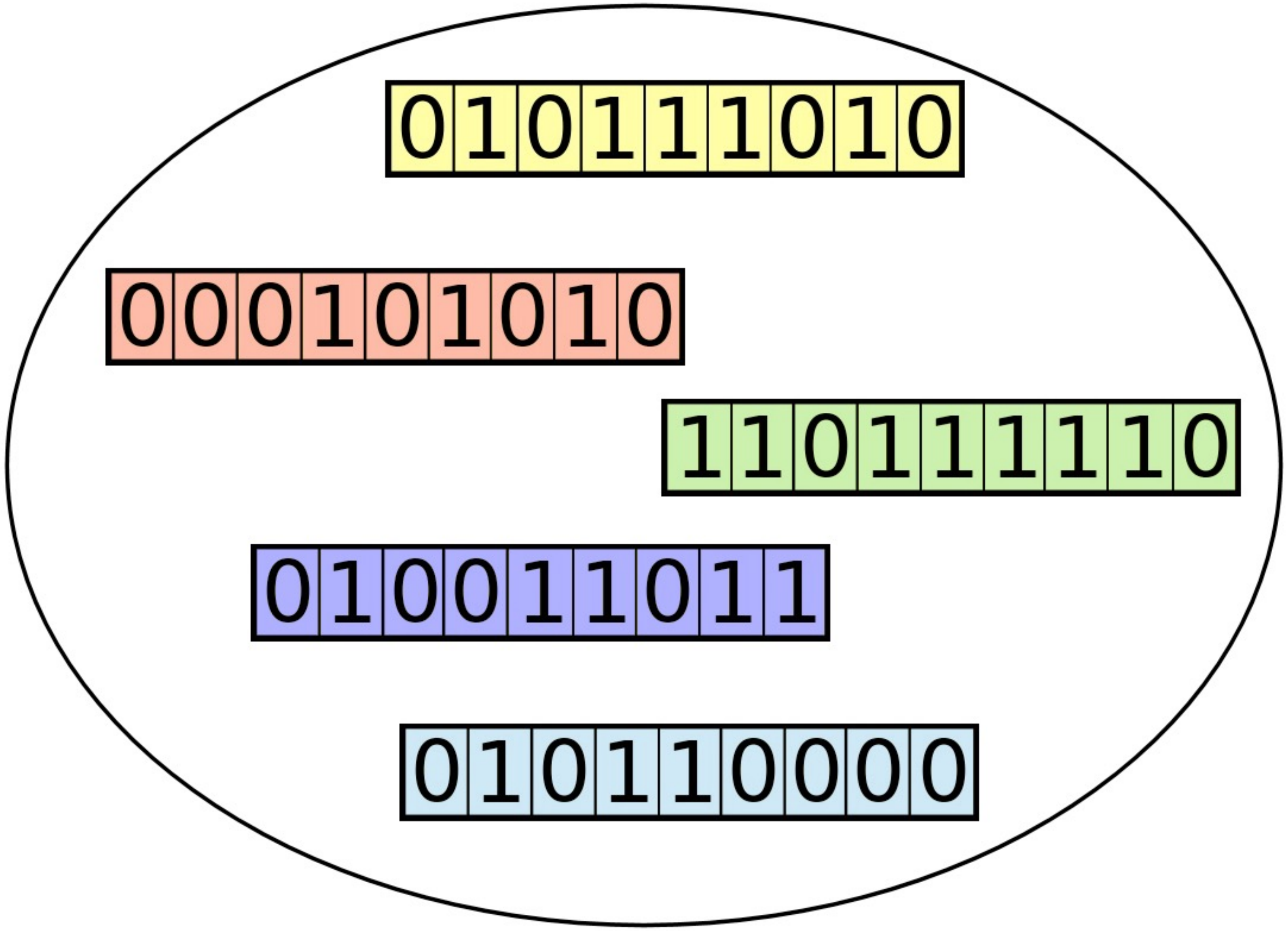
Genotype

- Each individual is represented by a chromosome representing a collection of genes.

- For example, each chromosome can be represented by a binary string, where each bit represents a single gene.



Population



Fitness Function

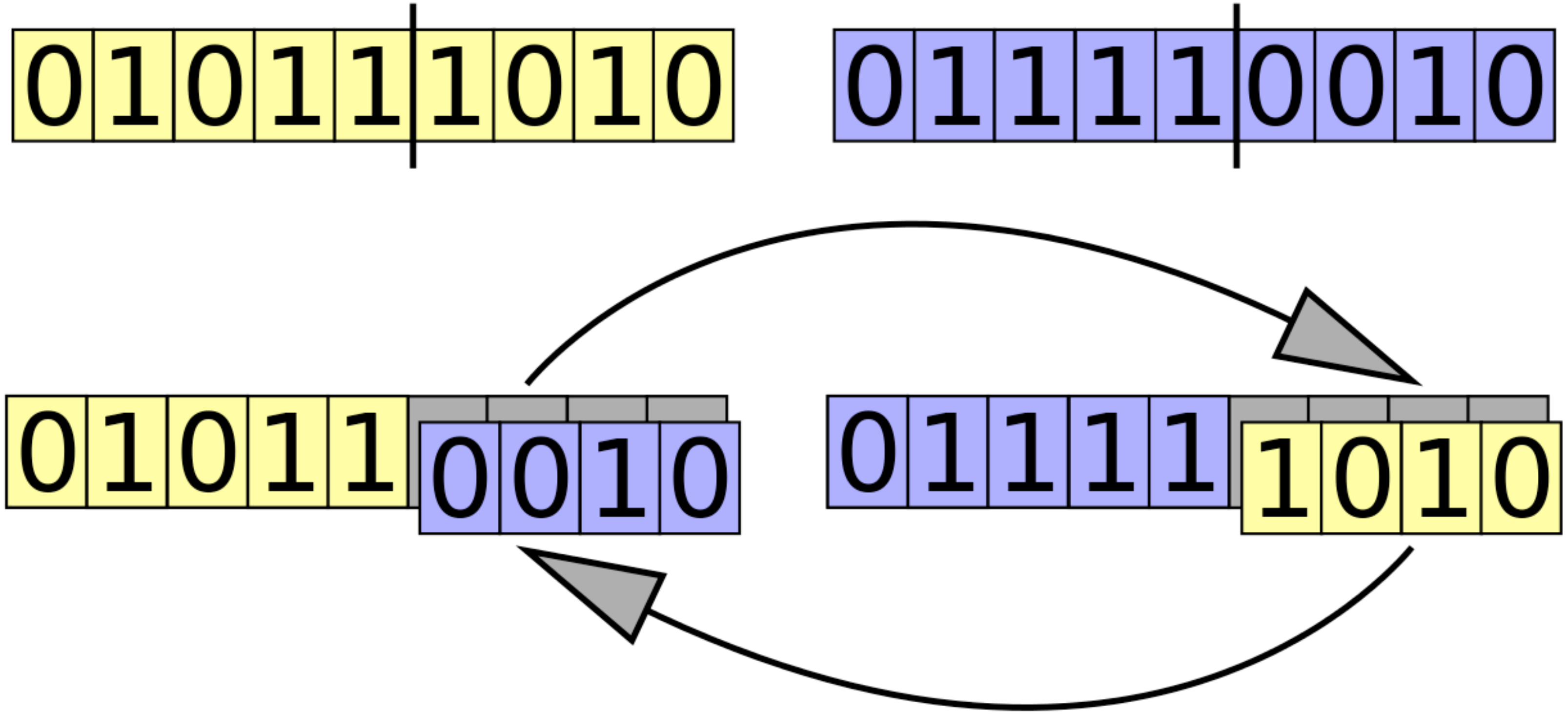
- The function we want to optimize or the problem we want to solve.
- Individuals score higher are more likely to be selected and reproduce.

Selection

- Following the calculation of the fitness score, individuals will be selected based on specific methods to reproduce and create their offspring to form the next generation.

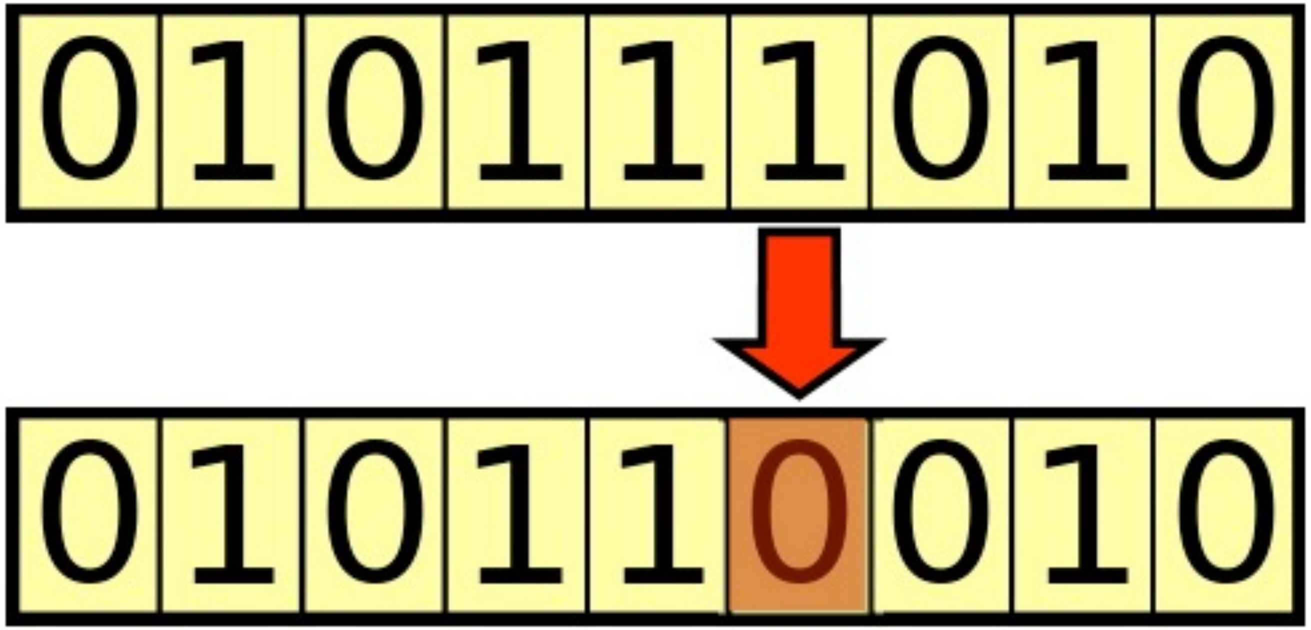
Crossover

- Chromosome exchange process

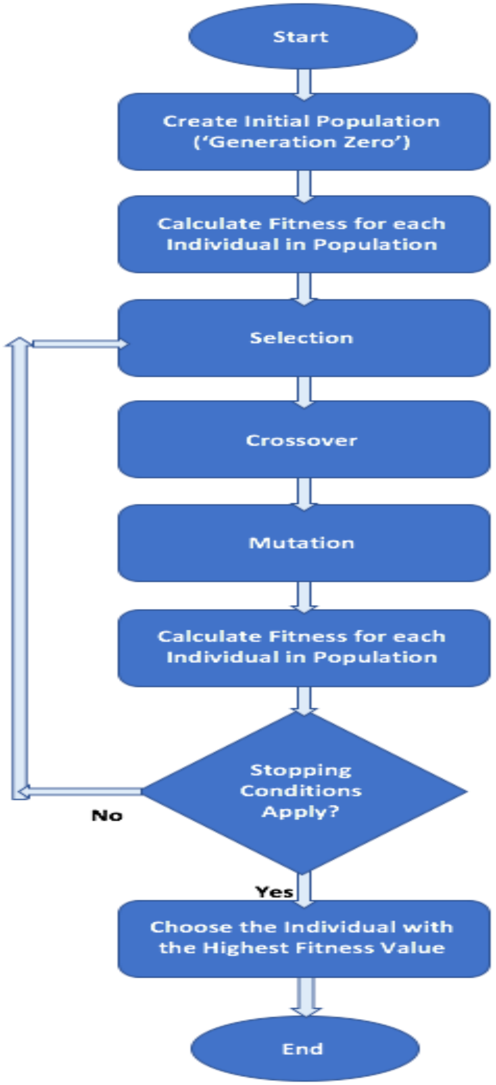


Mutation

- Introduce new patterns to the chromosomes



Basic Flow of Genetic Algorithm



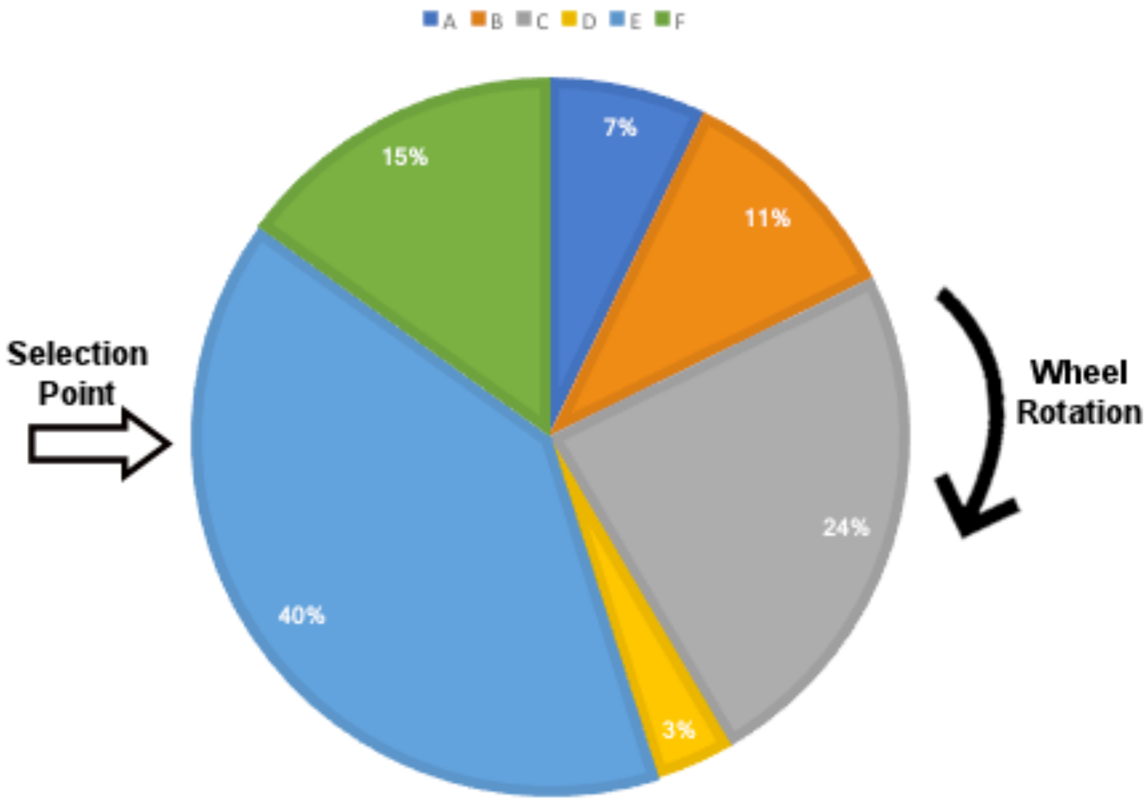
Stopping Conditions

- Time eclipsed
- CPU time/memory and associated cost
- Solution > preset threshold

Selection Methods

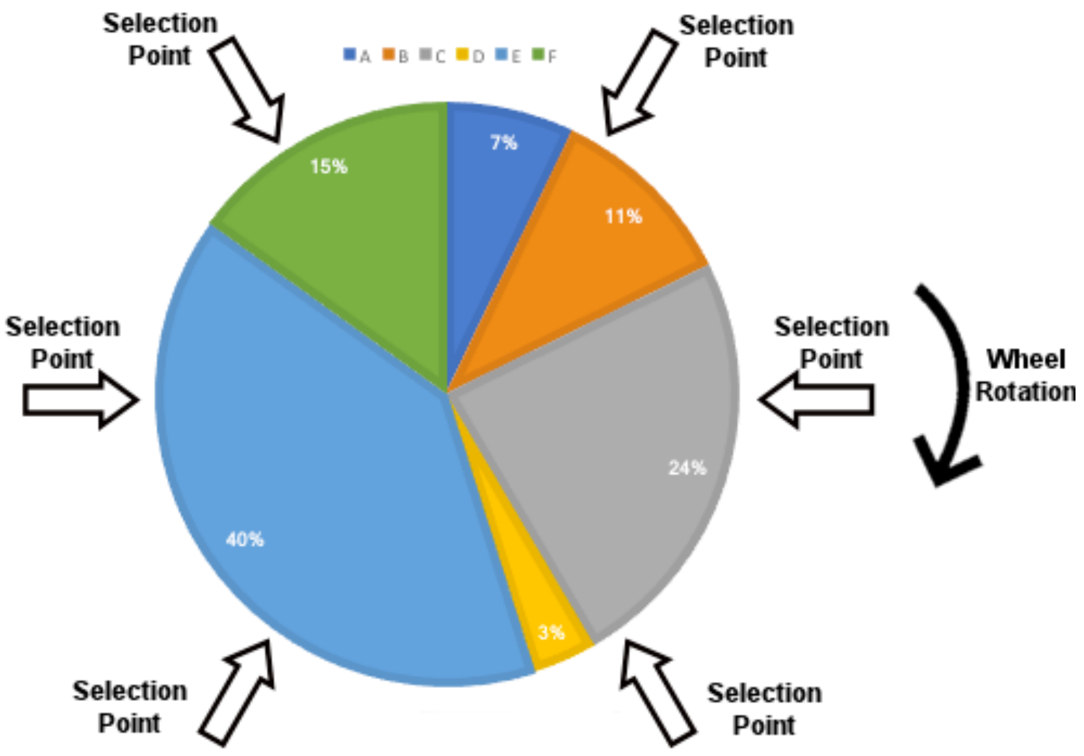
Roulette Wheel Selection (Fitness Proportionate Selection)

Individual	Fitness	Relative portion
A	8	7%
B	12	11%
C	27	24%
D	4	3%
E	45	40%
F	17	15%



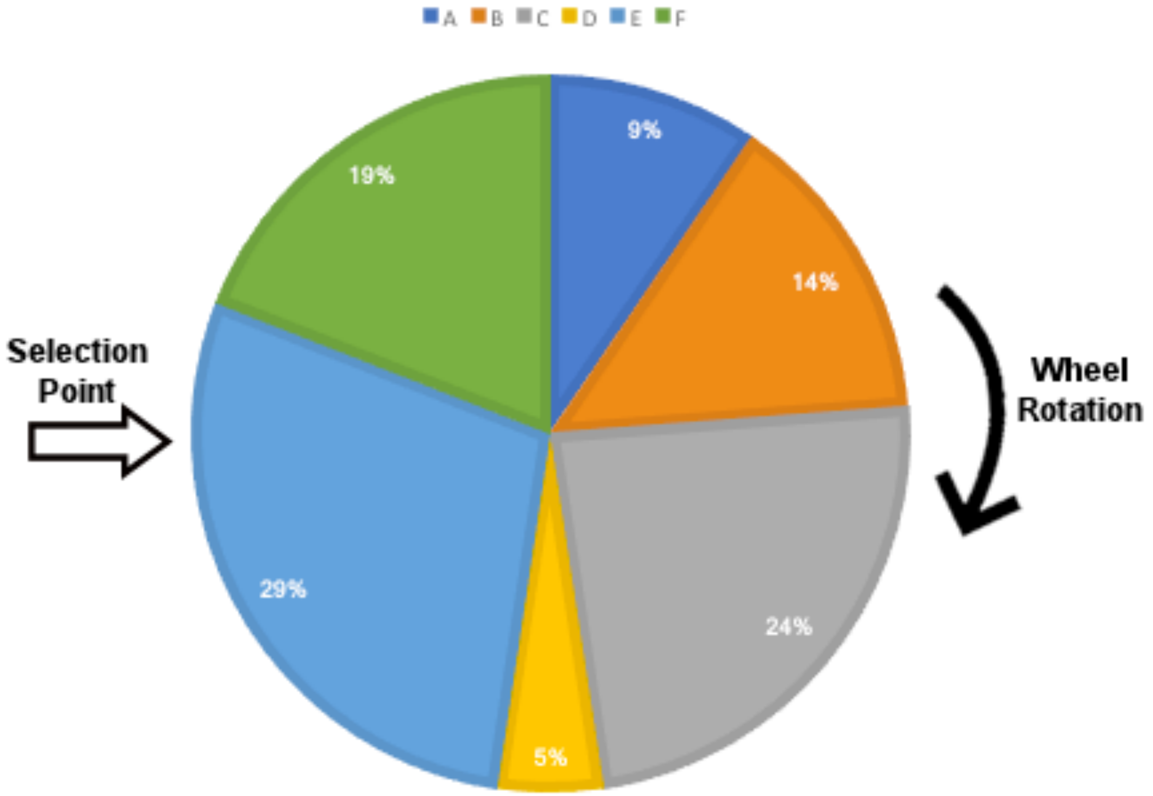
Stochastic Universal Sampling

- All individuals are chosen at the same time, offering more chances for those with lower fitness scores.



Rank-based Selection

Individual	Fitness	Rank	Relative portion
A	8	2	9%
B	12	3	14%
C	27	5	24%
D	4	1	5%
E	45	6	29%
F	17	4	19%



Fitness Scaling

- scaled fitness = a × (raw fitness) + b

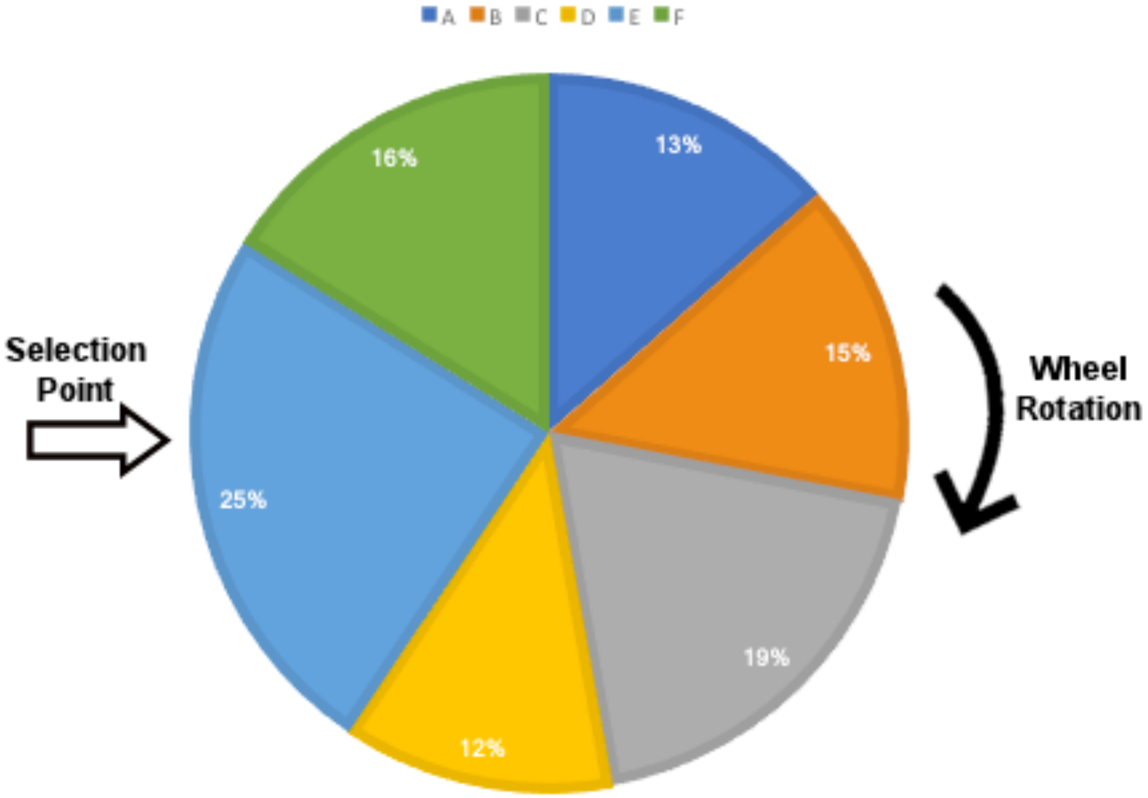
- 50 = a × 4 + b (lowest fitness value)
- 100 = a × 45 + b (highest fitness value)

- a = 1.22, b = 45.12

- scaled fitness = 1.22 × (raw fitness) + 45.12

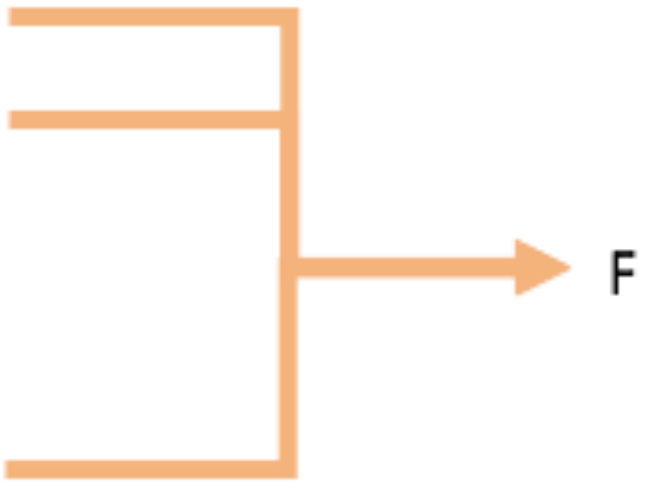
Fitness Scaling

Individual	Fitness	Scaled fitness	Relative portion
A	8	55	13%
B	12	60	15%
C	27	78	19%
D	4	50	12%
E	45	100	25%
F	17	66	16%



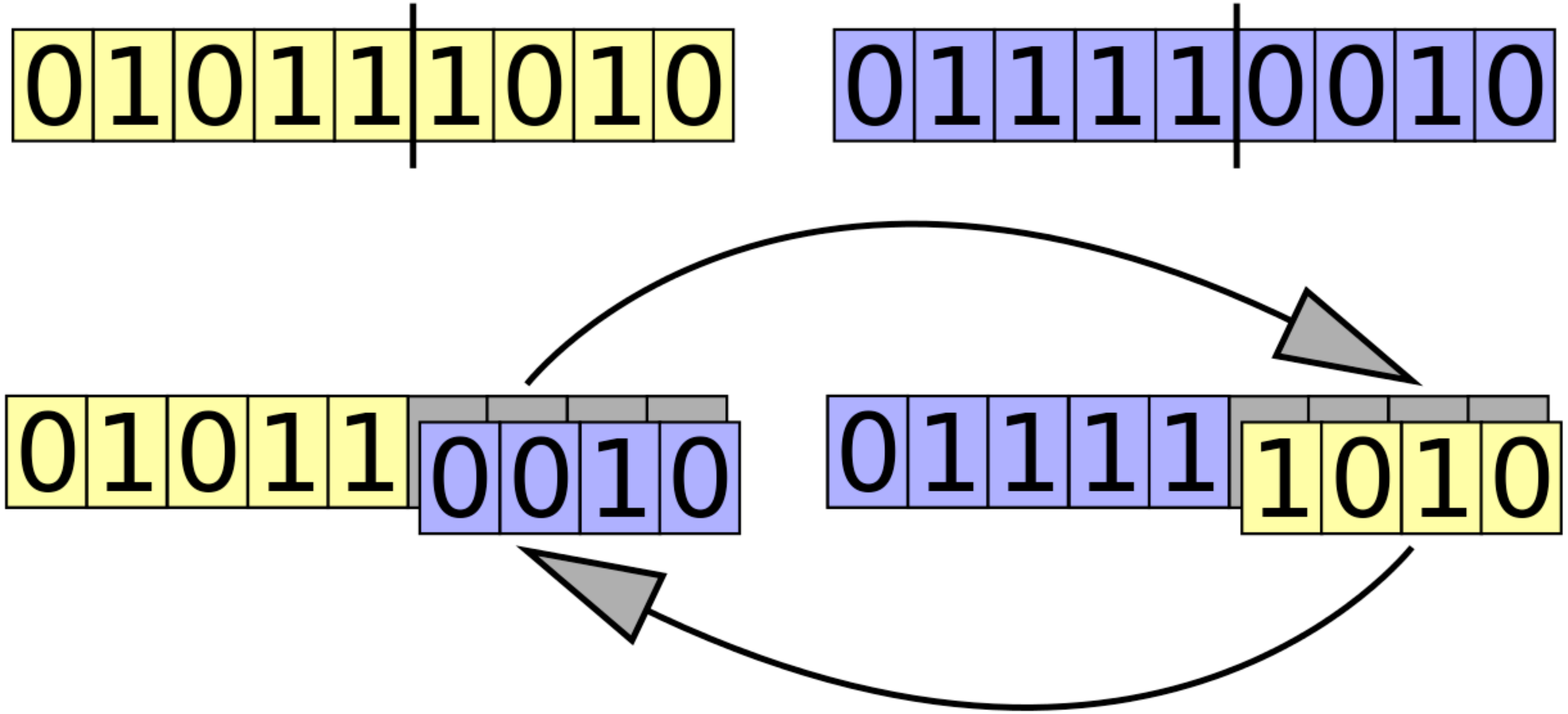
Tournament Selection

Individual	Fitness
A	8
B	12
C	27
D	4
E	45
F	17

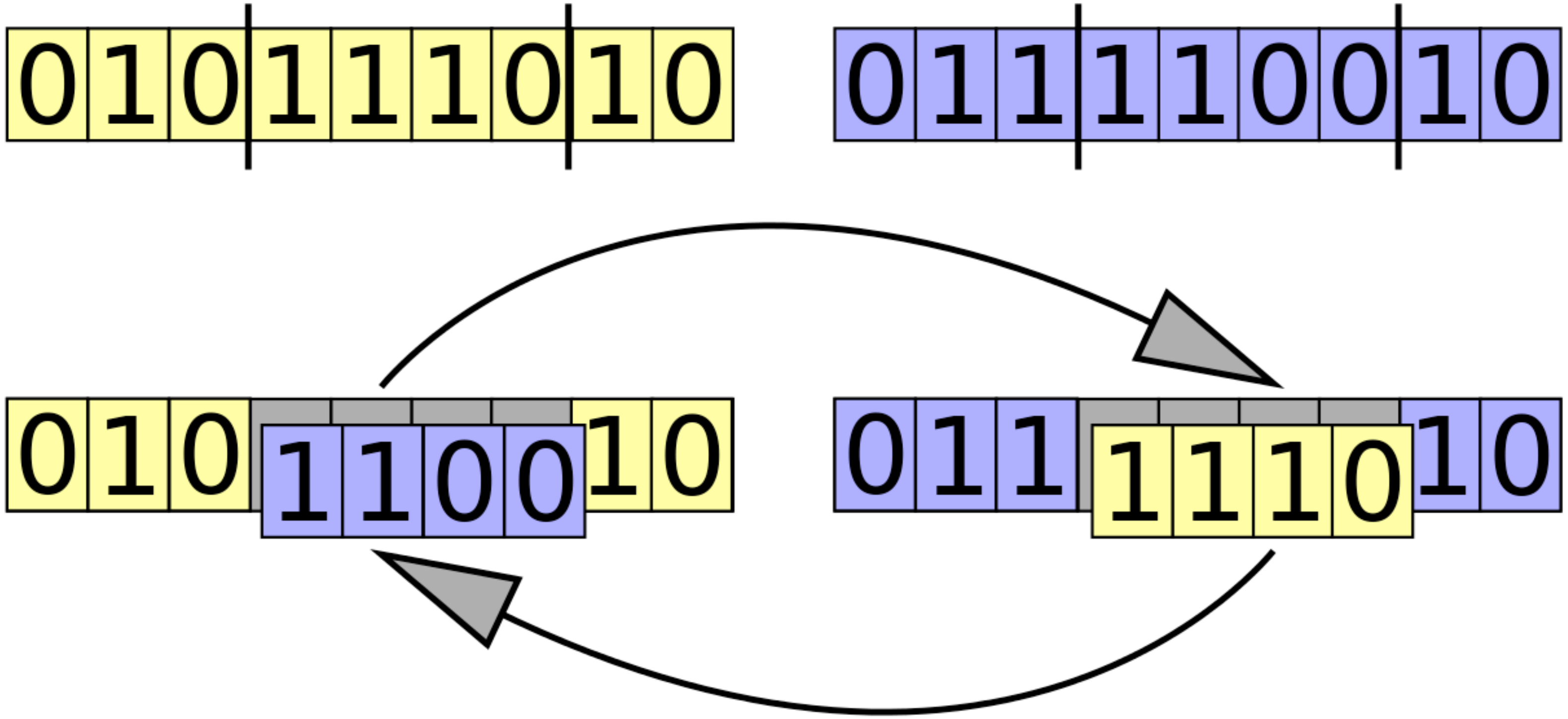


Crossover Methods

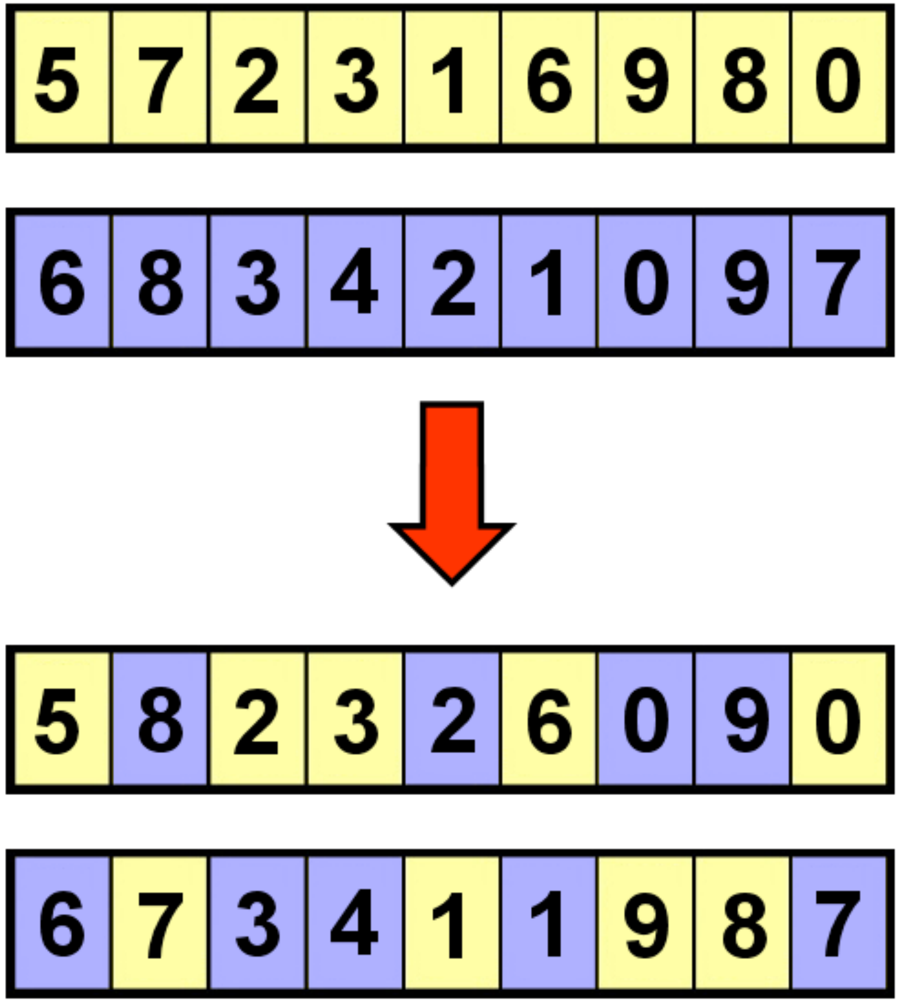
Single Point Crossover



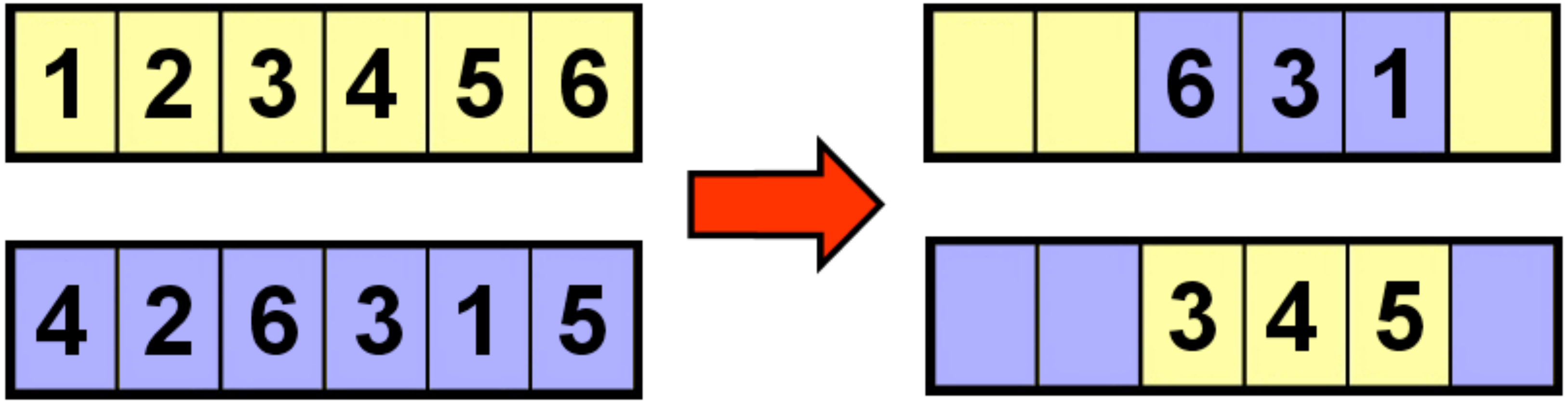
Two-Point or K-Point Crossover



Uniform Crossover

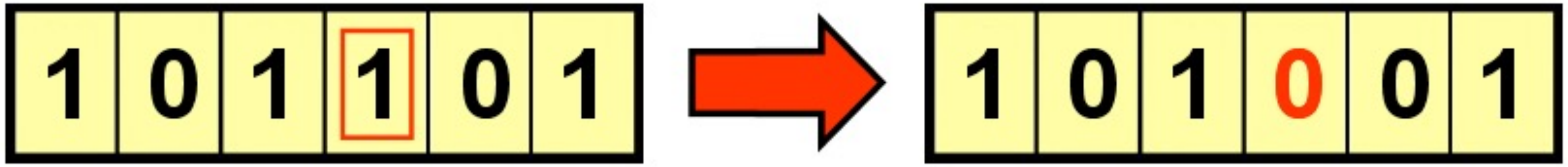


Ordered Crossover

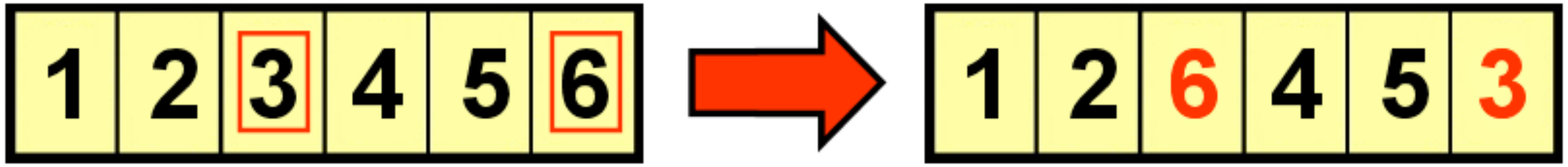


Mutation Methods

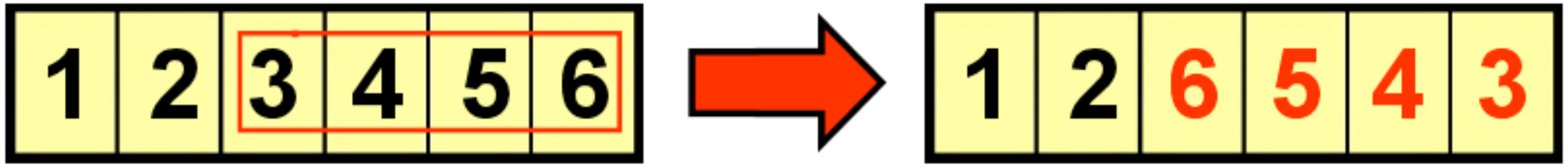
Flit Bit Mutation



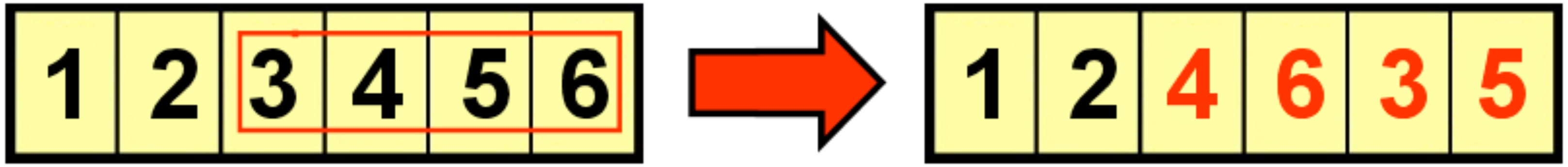
Swap Mutation



Inverse Mutation



Scramble Mutation



Elitism

- A strategy to retain the best individuals in the population of a genetic algorithm across generations.

- Purpose
  - Prevents loss of the best-found solutions.
  - Ensures consistent improvement in solution quality over time.

- How It Works
  - Selects top-performing individuals based on fitness.
  - Copies them unchanged into the next generation.

Introduction to DEAP

- A versatile Python library for evolutionary algorithms.
- Key Features
  - Easy to use and highly customizable.
  - Provides tools for implementing custom genetic operators.
- Installation
  - Install via pip: pip install deap

Setting Up a Genetic Algorithm with DEAP

- Importing Required Modules
  - from deap import base, creator, tools, algorithms

- Defining Fitness and Individual
  - Create fitness: creator.create("FitnessMax", base.Fitness, weights=(1.0,))
  - Define individual: creator.create("Individual", list, fitness=creator.FitnessMax)

Population Setup and Evaluation Function

- Registering Components with Toolbox
  - toolbox.register("attribute", random.randint, 0, 100)
  - toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attribute, n=10)
  - toolbox.register("population", tools.initRepeat, list, toolbox.individual)

- Defining the Evaluation Function
  - def evalOneMax(individual): return (sum(individual),)

- Register Evaluation Function
  - toolbox.register("evaluate", evalOneMax)

Genetic Operators and Algorithm Execution

- Registering Genetic Operators
  - Selection: toolbox.register("select", tools.selTournament, tournsize=3)
  - Crossover: toolbox.register("mate", tools.cxTwoPoint)
  - Mutation: toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)

- Executing the Algorithm
  - population = toolbox.population(n=300)
  - result = algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=40, verbose=False)