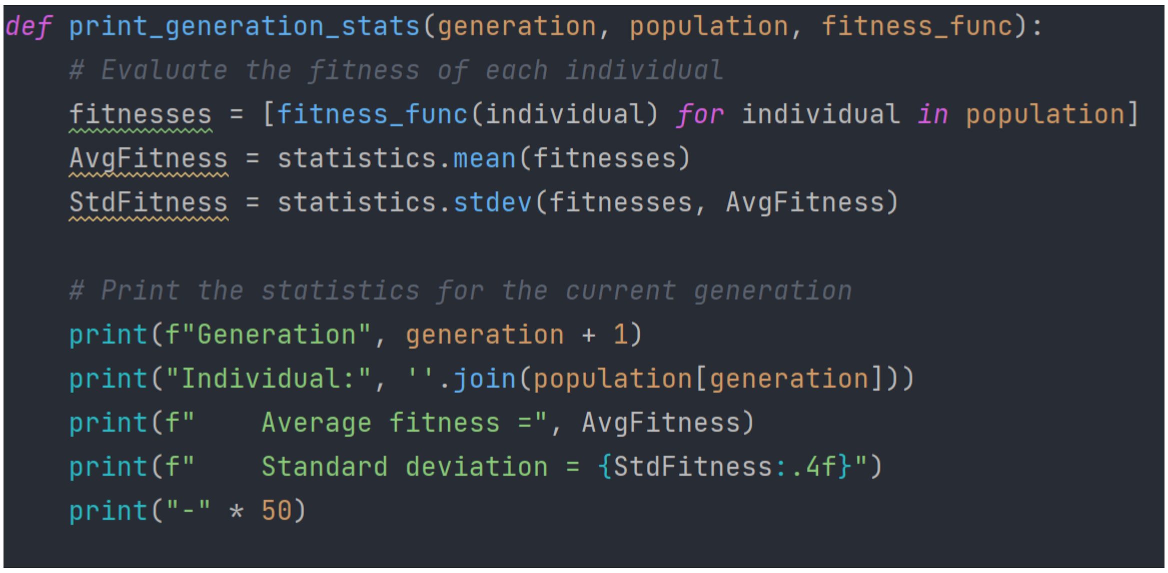
AI LAB – ASSIGNMENT 1A

אלגוריתמים גנטיים (מערכות מתארגנות עצמית) - חלק א' במשימה

SUBMITTERS:
Hakeem Abushqara – 207691312
Kareen Ghattas - 207478728

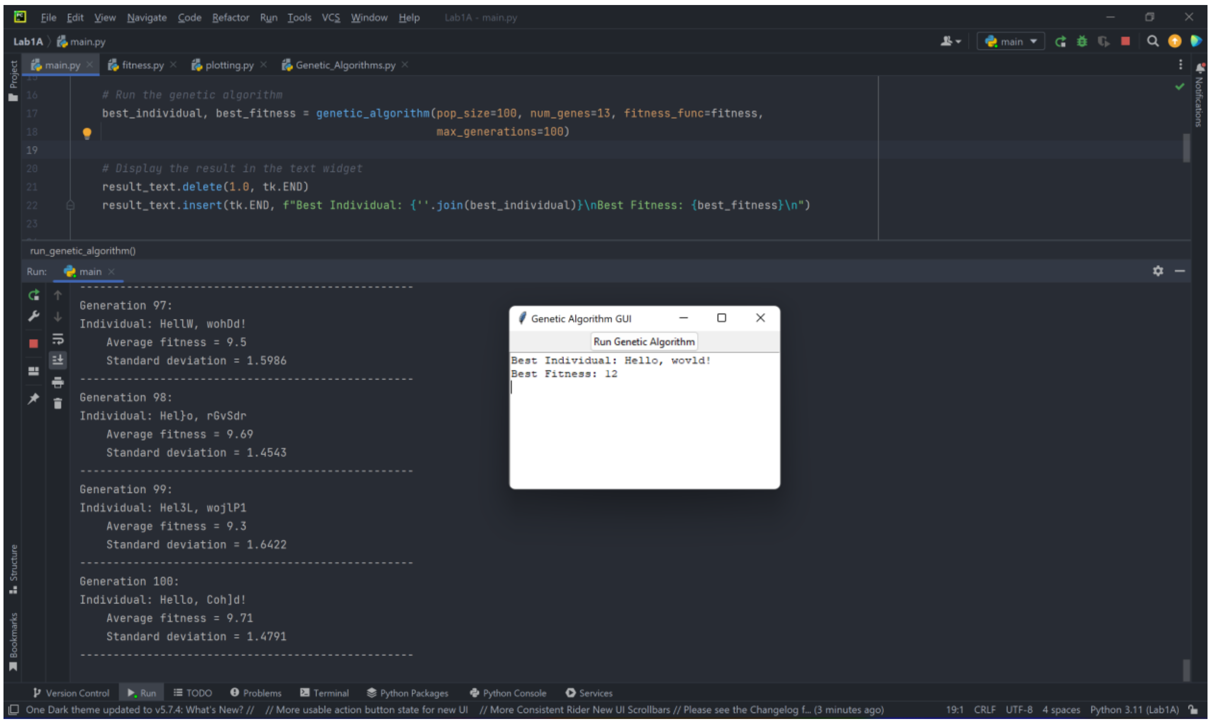
1. הוסיפו חישוב ודווח בכל דור של ממוצע ה– FITNESS של האוכלוסייה ושל סטיית התקן מהממוצע

We add a function to calculate the mean and average using the static package to get the right calculation like we learned in the lectures.



```python
def print_generation_stats(generation, population, fitness_func):
    # Evaluate the fitness of each individual
    fitnesses = [fitness_func(individual) for individual in population]
    AvgFitness = statistics.mean(fitnesses)
    StdFitness = statistics.stdev(fitnesses, AvgFitness)

    # Print the statistics for the current generation
    print(f"Generation", generation + 1)
    print("Individual:", ''.join(population[generation]))
    print(f"    Average fitness =", AvgFitness)
    print(f"    Standard deviation = {StdFitness:.4f}")
    print("-" * 50)
```



```python
    # Run the genetic algorithm
    best_individual, best_fitness = genetic_algorithm(pop_size=100, num_genes=13, fitness_func=fitness,
                                                      max_generations=100)

    # Display the result in the text widget
    result_text.delete(1.0, tk.END)
    result_text.insert(tk.END, f"Best Individual: {''.join(best_individual)}\nBest Fitness: {best_fitness}\n")
```

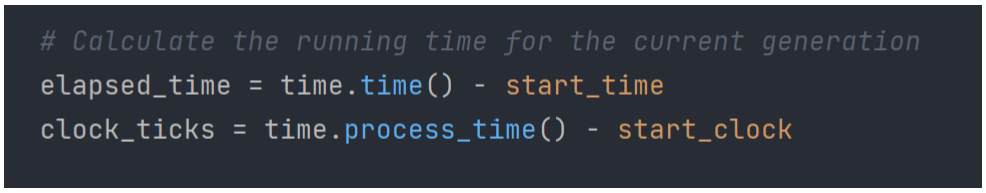
```
--------------------------------------------------
Generation 97:
Individual: HellW, wohDd!
    Average fitness = 9.5
    Standard deviation = 1.5986
--------------------------------------------------
Generation 98:
Individual: Hel}o, rGvSdr
    Average fitness = 9.69
    Standard deviation = 1.4543
--------------------------------------------------
Generation 99:
Individual: Hel3L, wojlP1
    Average fitness = 9.3
    Standard deviation = 1.6422
--------------------------------------------------
Generation 100:
Individual: Hello, Coh]d!
    Average fitness = 9.71
    Standard deviation = 1.4791
--------------------------------------------------
```

```
Best Individual: Hello, wovld!
Best Fitness: 12
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Lab1A - main.py
Lab1A main.py
main.py fitness.py plotting.py Genetic_Algorithms.py
run_genetic_algorithm()
Run: main
Genetic Algorithm GUI
Run Genetic Algorithm
Version Control Run TODO Problems Terminal Python Packages Python Console Services
One Dark theme updated to v5.7.4: What's New? // // More usable action button state for new UI // More Consistent Rider New UI Scrollbars // Please see the Changelog f... (3 minutes ago)
19:1 CRLF UTF-8 4 spaces Python 3.11 (Lab1A)

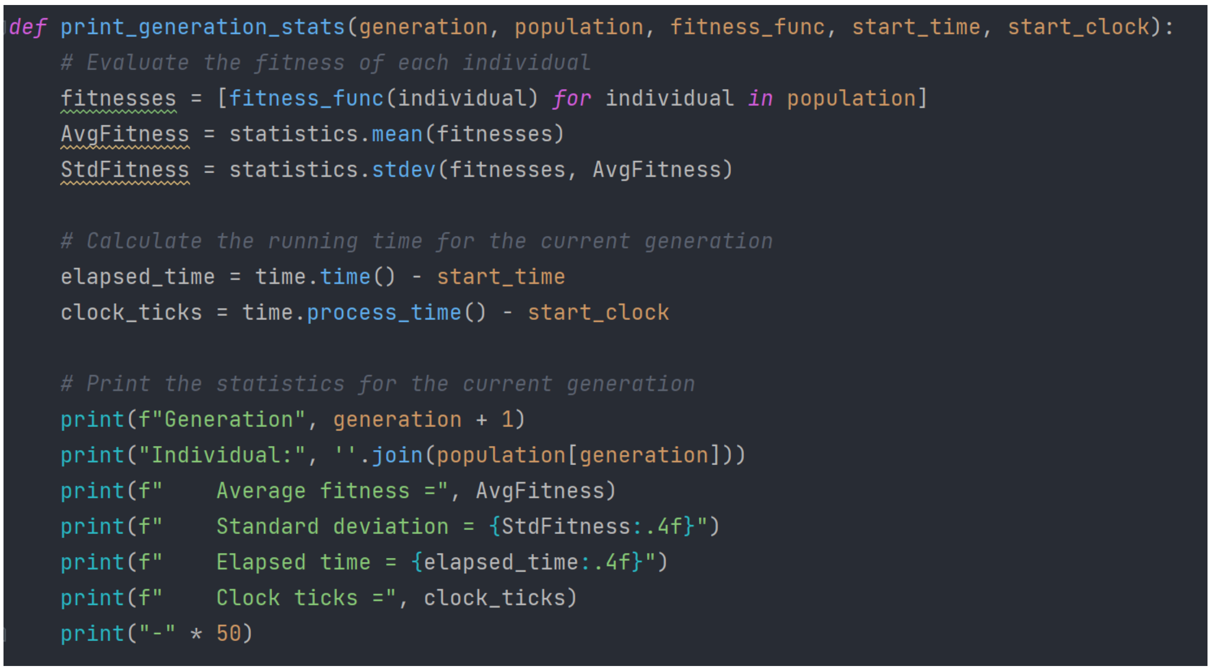
2. הוסיפו חישוב ודווח **בכל דור** של זמן ריצה CLOCK TICKS וזמן ריצה אבסולוטי ELAPSED וכן עד להתכנסות למינימום לוקאלי או גלובאלי

We added a declaration for starting time at the beginning of the GA function:



```python
# Calculate the running time for the current generation
elapsed_time = time.time() - start_time
clock_ticks = time.process_time() - start_clock
```

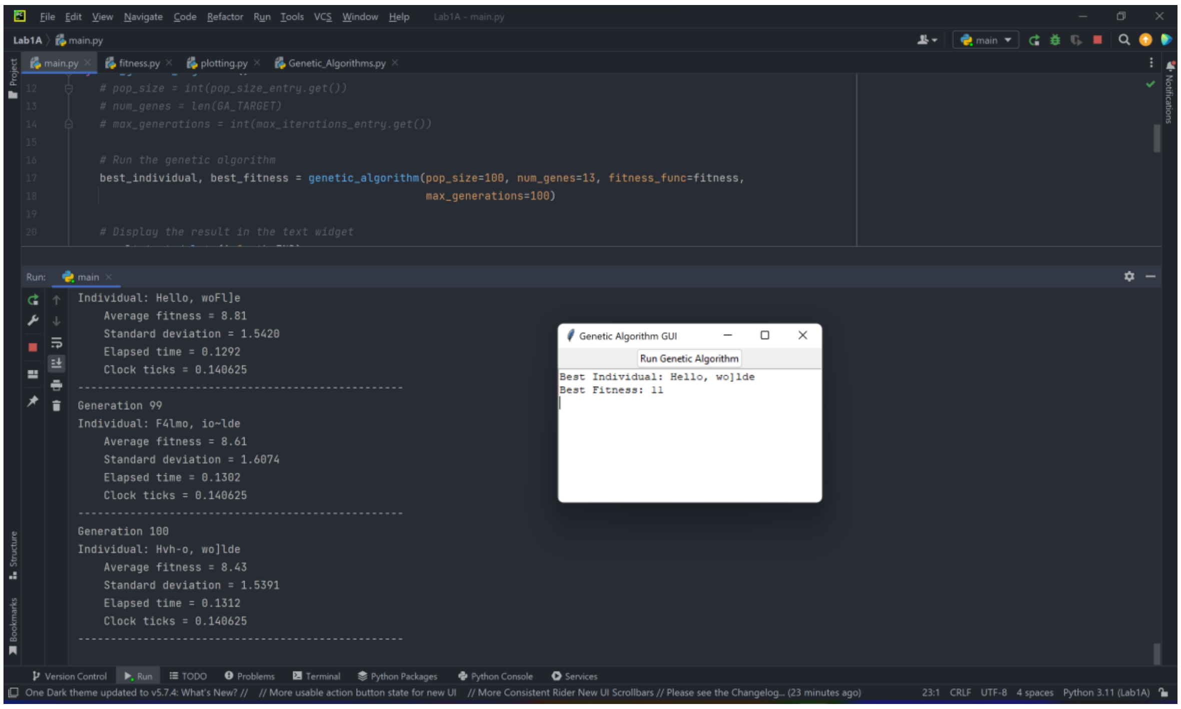
And for each generation we added the time calculations ("elapsed_time" and "clock_ticks"):



```python
def print_generation_stats(generation, population, fitness_func, start_time, start_clock):
    # Evaluate the fitness of each individual
    fitnesses = [fitness_func(individual) for individual in population]
    AvgFitness = statistics.mean(fitnesses)
    StdFitness = statistics.stdev(fitnesses, AvgFitness)

    # Calculate the running time for the current generation
    elapsed_time = time.time() - start_time
    clock_ticks = time.process_time() - start_clock

    # Print the statistics for the current generation
    print(f"Generation", generation + 1)
    print("Individual:", ''.join(population[generation]))
    print(f"    Average fitness =", AvgFitness)
    print(f"    Standard deviation = {StdFitness:.4f}")
    print(f"    Elapsed time = {elapsed_time:.4f}")
    print(f"    Clock ticks =", clock_ticks)
    print("-" * 50)
```



```python
    # pop_size = int(pop_size_entry.get())
    # num_genes = len(GA_TARGET)
    # max_generations = int(max_iterations_entry.get())

    # Run the genetic algorithm
    best_individual, best_fitness = genetic_algorithm(pop_size=100, num_genes=13, fitness_func=fitness,
                                                      max_generations=100)

    # Display the result in the text widget
```

```
Individual: Hello, woFl]e
    Average fitness = 8.81
    Standard deviation = 1.5420
    Elapsed time = 0.1292
    Clock ticks = 0.140625
--------------------------------------------------
Generation 99
Individual: F4lmo, io~lde
    Average fitness = 8.61
    Standard deviation = 1.6074
    Elapsed time = 0.1302
    Clock ticks = 0.140625
--------------------------------------------------
Generation 100
Individual: Hvh-o, wo]lde
    Average fitness = 8.43
    Standard deviation = 1.5391
    Elapsed time = 0.1312
    Clock ticks = 0.140625
--------------------------------------------------
```

```
Best Individual: Hello, wo]lde
Best Fitness: 11
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Lab1A - main.py
Lab1A main.py
main.py fitness.py plotting.py Genetic_Algorithms.py
Run: main
Genetic Algorithm GUI
Run Genetic Algorithm
Version Control Run TODO Problems Terminal Python Packages Python Console Services
One Dark theme updated to v5.7.4: What's New? // // More usable action button state for new UI // More Consistent Rider New UI Scrollbars // Please see the Changelog... (23 minutes ago)
23:1 CRLF UTF-8 4 spaces Python 3.11 (Lab1A)

3. הציגו חישוב ודווח בכל דור את החלוקה של הפרטים באוכלוסיה לדצילים לפי הפיטנס שלהם

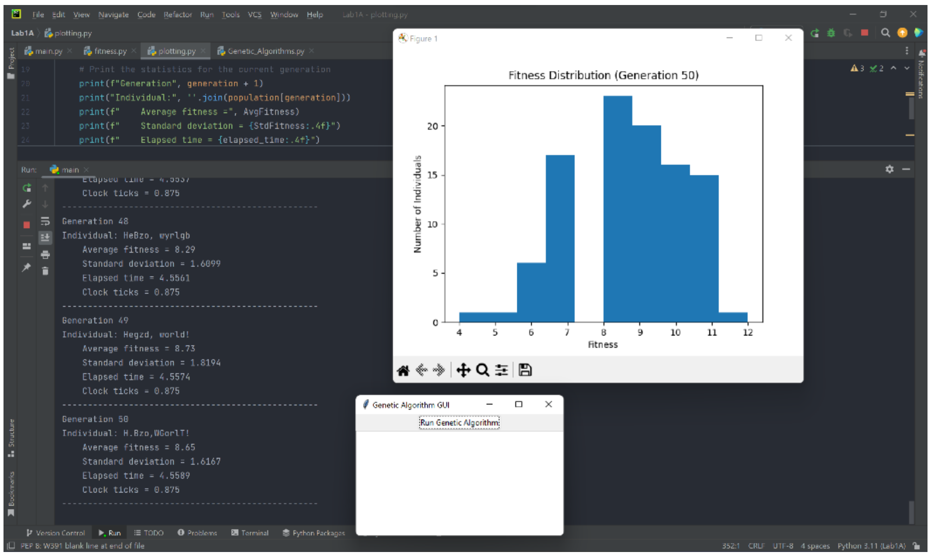
Creating the histogram drawing:

```python
# Define a function to plot a histogram of the fitness distribution
def plot_fitness_histogram(generation, population, fitness_func, plt=None):
    # Evaluate the fitness of each individual
    fitnesses = [fitness_func(individual) for individual in population]

    # Plot the histogram
    plt.hist(fitnesses, bins=math.ceil(math.sqrt(len(population))))
    plt.title(f"Fitness Distribution (Generation {generation+1})")
    plt.xlabel("Fitness")
    plt.ylabel("Number of Individuals")
    plt.show()
```

```python
# Plot the fitness histogram every 10 generations
if (generation + 1) % 10 == 0:
    plot_fitness_histogram(generation, population, fitness_func)
```

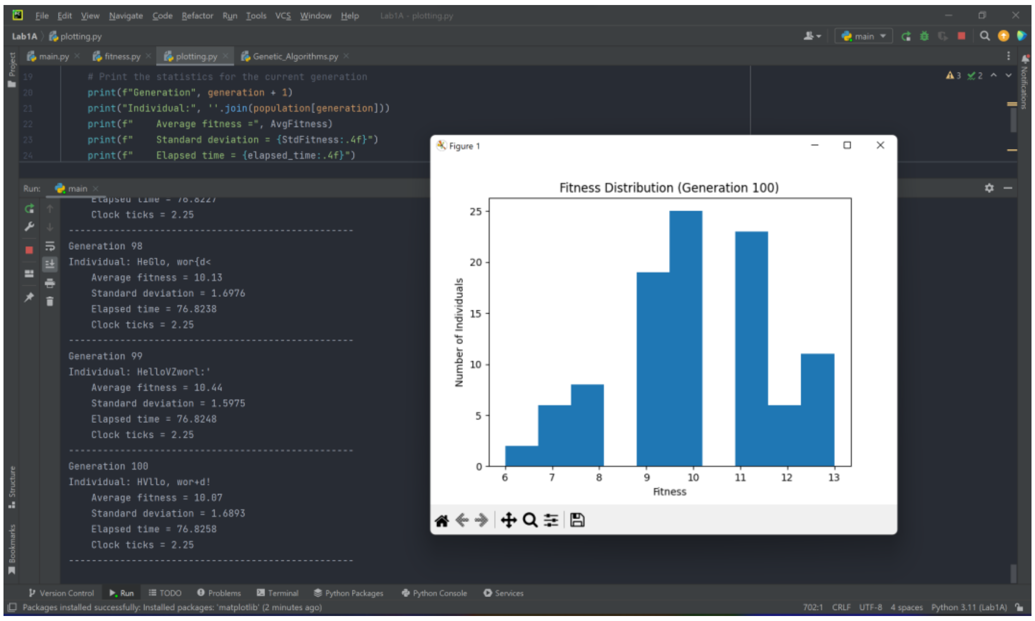
An example of the histograms:



```python
    # Print the statistics for the current generation
    print(f"Generation", generation + 1)
    print("Individual:", ''.join(population[generation]))
    print(f"    Average fitness =", AvgFitness)
    print(f"    Standard deviation = {StdFitness:.4f}")
    print(f"    Elapsed time = {elapsed_time:.4f}")
```

```
    Elapsed time = 4.5537
    Clock ticks = 0.875
---------------------------------------------
Generation 48
Individual: HeBzo, wyrlgb
    Average fitness = 8.29
    Standard deviation = 1.6099
    Elapsed time = 4.5561
    Clock ticks = 0.875
---------------------------------------------
Generation 49
Individual: Hegzd, world!
    Average fitness = 8.73
    Standard deviation = 1.8194
    Elapsed time = 4.5574
    Clock ticks = 0.875
---------------------------------------------
Generation 50
Individual: H.Bzo,WGorlT!
    Average fitness = 8.65
    Standard deviation = 1.6167
    Elapsed time = 4.5589
    Clock ticks = 0.875
---------------------------------------------
```

PEP 8: W391 blank line at end of file


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Lab1A - plotting.py
Lab1A plotting.py
main.py fitness.py plotting.py Genetic_Algorithms.py
Run: main
Figure 1
Fitness Distribution (Generation 100)
Number of Individuals
Fitness
0 5 10 15 20 25
6 7 8 9 10 11 12 13
Version Control Run TODO Problems Terminal Python Packages Python Console Services
702:1 CRLF UTF-8 4 spaces Python 3.11 (Lab1A)

```
        # Print the statistics for the current generation
        print(f"Generation", generation + 1)
        print("Individual:", ''.join(population[generation]))
        print(f"    Average fitness =", AvgFitness)
        print(f"    Standard deviation = {StdFitness:.4f}")
        print(f"    Elapsed time = {elapsed_time:.4f}")
```

```
    Elapsed time = 76.8227
    Clock ticks = 2.25
--------------------------------------------------
Generation 98
Individual: HeGlo, wor{d<
    Average fitness = 10.13
    Standard deviation = 1.6976
    Elapsed time = 76.8238
    Clock ticks = 2.25
--------------------------------------------------
Generation 99
Individual: HelloVZworl:'
    Average fitness = 10.44
    Standard deviation = 1.5975
    Elapsed time = 76.8248
    Clock ticks = 2.25
--------------------------------------------------
Generation 100
Individual: HVllo, wor+d!
    Average fitness = 10.07
    Standard deviation = 1.6893
    Elapsed time = 76.8258
    Clock ticks = 2.25
--------------------------------------------------
```

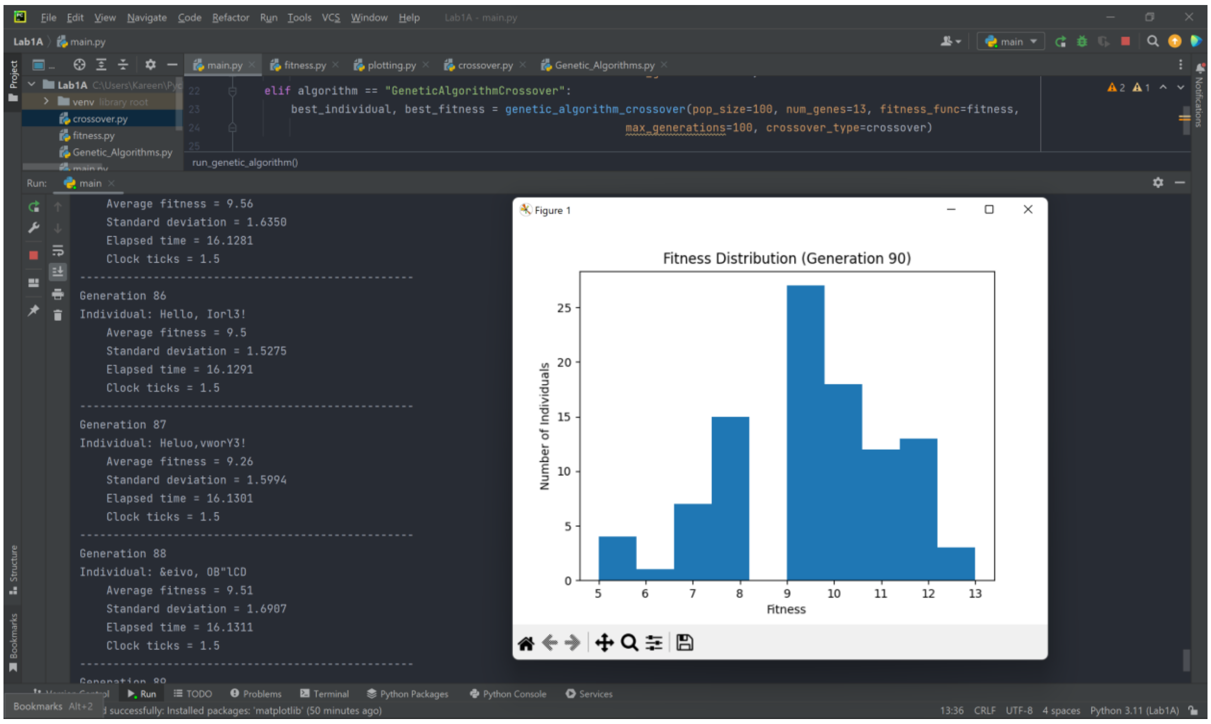
Packages installed successfully: Installed packages: 'matplotlib' (2 minutes ago)

4. ממשו כך שאפשר יהיה לבחור בין שלושת האופרטורים לשיחלוף SINGLE,TWO,UNIFORM

The function first checks which crossover type was specified and then performs the corresponding operation. If "**SINGLE**" is specified, a single point crossover is performed where a random point is chosen and the parents are split at that point to create two children. If "**TWO**" is specified, a two point crossover is performed where two random points are chosen and the parents are split at those points to create two children. If "**UNIFORM**" is specified, a uniform crossover is performed where each gene is chosen from one of the parents with a 50/50 probability to create two children.

```
def crossover(parent1, parent2, crossover_type):
    if crossover_type == "SINGLE":
        # Perform single point crossover
        point = random.randint(1, len(parent1) - 1)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]
    elif crossover_type == "TWO":
        # Perform two point crossover
        point1 = random.randint(1, len(parent1) - 2)
        point2 = random.randint(point1 + 1, len(parent1) - 1)
        child1 = parent1[:point1] + parent2[point1:point2] + parent1[point2:]
        child2 = parent2[:point1] + parent1[point1:point2] + parent2[point2:]
    elif crossover_type == "UNIFORM":
        # Perform uniform crossover
        child1 = [parent1[i] if random.random() < 0.5 else parent2[i] for i in range(len(parent1))]
        child2 = [parent2[i] if random.random() < 0.5 else parent1[i] for i in range(len(parent2))]
    else:
        raise ValueError("Invalid crossover type specified.")

    return child1, child2
```

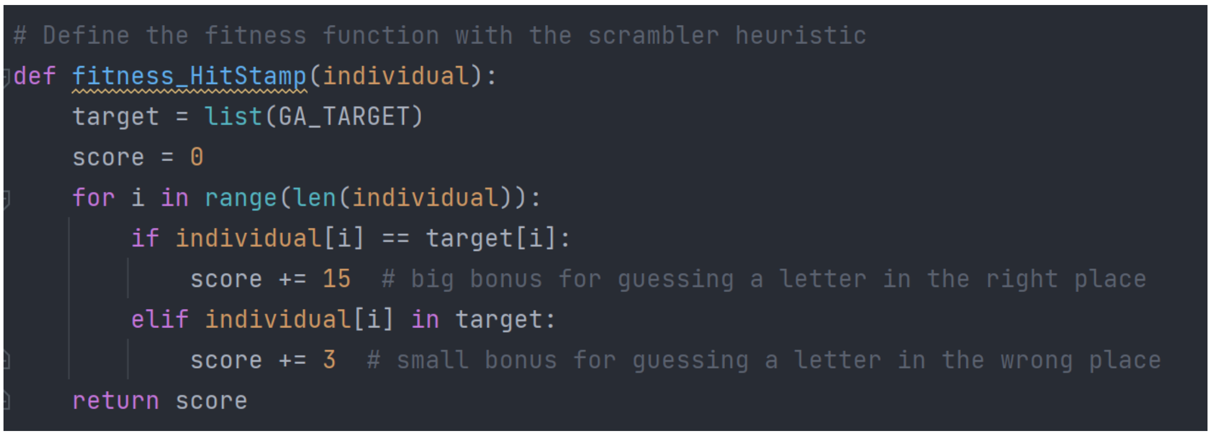


```
    elif algorithm == "GeneticAlgorithmCrossover":
        best_individual, best_fitness = genetic_algorithm_crossover(pop_size=100, num_genes=13, fitness_func=fitness,
                                                                    max_generations=100, crossover_type=crossover)
```

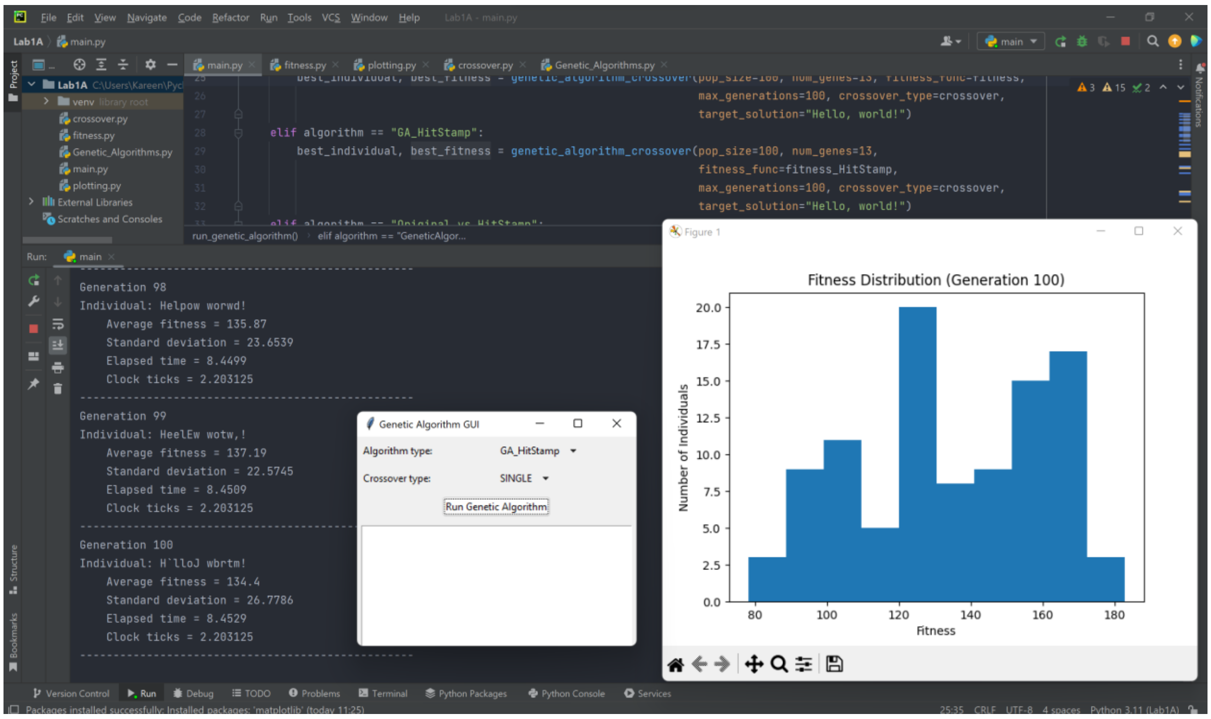
```
    Average fitness = 9.56
    Standard deviation = 1.6350
    Elapsed time = 16.1281
    Clock ticks = 1.5
--------------------------------------------------
Generation 86
Individual: Hello, Iorl3!
    Average fitness = 9.5
    Standard deviation = 1.5275
    Elapsed time = 16.1291
    Clock ticks = 1.5
--------------------------------------------------
Generation 87
Individual: Heluo,vworY3!
    Average fitness = 9.26
    Standard deviation = 1.5994
    Elapsed time = 16.1301
    Clock ticks = 1.5
--------------------------------------------------
Generation 88
Individual: &eivo, OB"lCD
    Average fitness = 9.51
    Standard deviation = 1.6907
    Elapsed time = 16.1311
    Clock ticks = 1.5
--------------------------------------------------
```

5. הוסיפו היוריסטיקה נוספת של "בול פגיעה" – פונקציה "המצ'פרת" ניחוש אות במחרוזת ולו אם אינה במיקום הנכון וכן נותנת בונוס גדול על ניחוש אות במקום הנכון.

In this modified **fitness** function, we first count the number of letters in **individual** that match letters in **GA_TARGET** in the correct position. We then add the score returned by the **hit stamp** heuristic to the score. This way, the **fitness** function takes into account both letters in the correct position and letters in the wrong position.



```
# Define the fitness function with the scrambler heuristic
def fitness_HitStamp(individual):
    target = list(GA_TARGET)
    score = 0
    for i in range(len(individual)):
        if individual[i] == target[i]:
            score += 15  # big bonus for guessing a letter in the right place
        elif individual[i] in target:
            score += 3  # small bonus for guessing a letter in the wrong place
    return score
```



```
best_individual, best_fitness = genetic_algorithm_crossover(pop_size=100, num_genes=13, fitness_func=fitness,
                                                            max_generations=100, crossover_type=crossover,
                                                            target_solution="Hello, world!")

elif algorithm == "GA_HitStamp":
    best_individual, best_fitness = genetic_algorithm_crossover(pop_size=100, num_genes=13,
                                                                fitness_func=fitness_HitStamp,
                                                                max_generations=100, crossover_type=crossover,
                                                                target_solution="Hello, world!")

elif algorithm == "Original_vs_HitStamp":
```

```
--------------------------------------------------
Generation 98
Individual: Helpow worwd!
    Average fitness = 135.87
    Standard deviation = 23.6539
    Elapsed time = 8.4499
    Clock ticks = 2.203125
--------------------------------------------------
Generation 99
Individual: HeelEw wotw,!
    Average fitness = 137.19
    Standard deviation = 22.5745
    Elapsed time = 8.4509
    Clock ticks = 2.203125
--------------------------------------------------
Generation 100
Individual: H`lloJ wbrtm!
    Average fitness = 134.4
    Standard deviation = 26.7786
    Elapsed time = 8.4529
    Clock ticks = 2.203125
--------------------------------------------------
```

Genetic Algorithm GUI

Algorithm type: GA_HitStamp

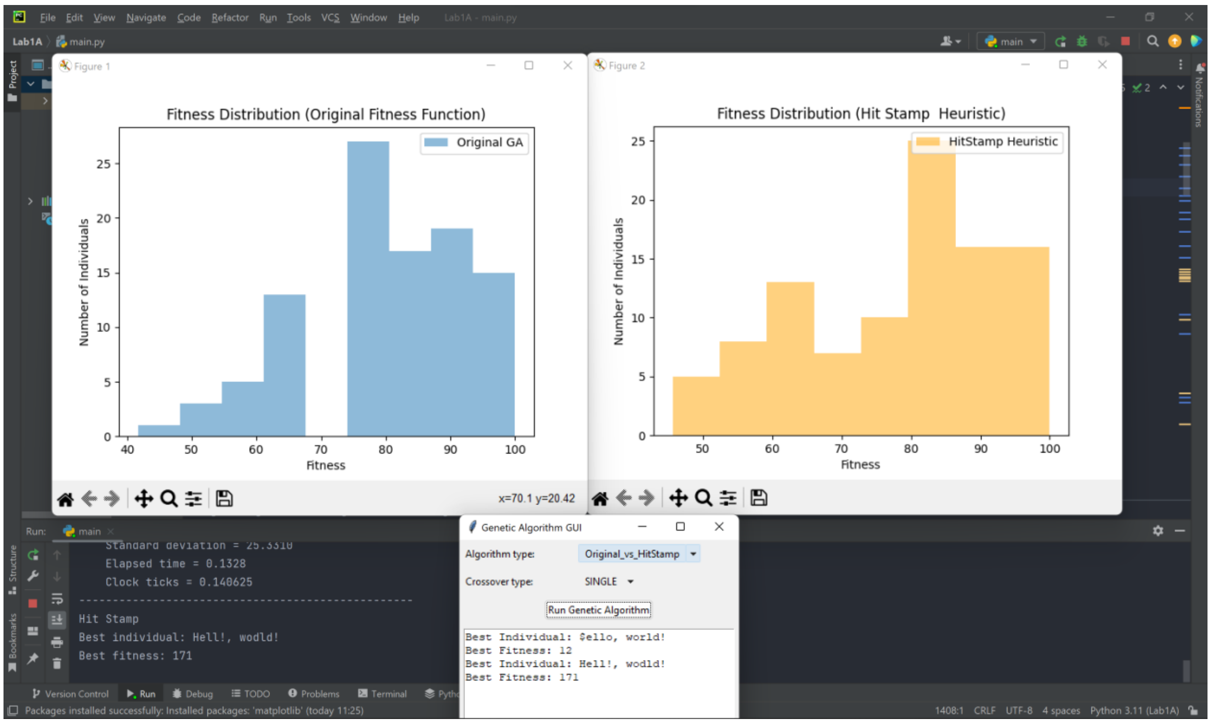
Crossover type: SINGLE

Packages installed successfully: Installed packages: 'matplotlib' (today 11:25)

6. השוו את ההיוריסטיקה הזו מול המקורית. (הסבירו כיצד היא קשורה למוטציות במהלך האלגוריתם?) – האם היא משפרת את ההיוריסטיקה המקורית ואם לאו מדוע?

We add a histogram to show the difference between the two heuristics, the histogram show the distribution of the fitness scores of the best individuals in each generation for both versions of the algorithm.

```python
def plot_original_vs_HitStamp(population, fitness_func):
    if fitness_func == fitness:
        # Plot the fitness histogram for the original fitness function
        plt.figure()
        OrigFitnesses = [fitness(individual) for individual in population]
        Normalfitnesses = [f / max(OrigFitnesses) for f in OrigFitnesses]
        # Scale the normalized fitness scores to a range of 0-100
        fitnesses_scaled = [f * 100 for f in Normalfitnesses]
        plt.hist(fitnesses_scaled, bins='auto', alpha=0.5, label='Original GA')
        plt.title("Fitness Distribution (Original Fitness Function)")
        plt.xlabel("Fitness")
        plt.ylabel("Number of Individuals")
        plt.legend(loc='upper right')
    elif fitness_func == fitness_HitStamp:
        # Plot the fitness histogram for the scrambler heuristic
        plt.figure()
        fitnesses_scrambler = [fitness_HitStamp(individual) for individual in population]
        Normalfitnesses = [f / max(fitnesses_scrambler) for f in fitnesses_scrambler]
        # Scale the normalized fitness scores to a range of 0-100
        fitnesses_scaled = [f * 100 for f in Normalfitnesses]
        plt.hist(fitnesses_scaled, bins='auto', alpha=0.5, color='orange', label='HitStamp Heuristic')
        plt.title("Fitness Distribution (Hit Stamp  Heuristic)")
        plt.xlabel("Fitness")
        plt.ylabel("Number of Individuals")
        plt.legend(loc='upper right')
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Lab1A - main.py
Lab1A main.py
Figure 1
Fitness Distribution (Original Fitness Function)
Original GA
Number of Individuals
Fitness
x=70.1 y=20.42
Figure 2
Fitness Distribution (Hit Stamp Heuristic)
HitStamp Heuristic
Number of Individuals
Fitness
Run: main
Version Control
Run
Debug
TODO
Problems
Terminal
1408:1 CRLF UTF-8 4 spaces Python 3.11 (Lab1A)

```
    Standard deviation = 25.3310
    Elapsed time = 0.1328
    Clock ticks = 0.140625
-----------------------------------------------------
Hit Stamp
Best individual: Hell!, wodld!
Best fitness: 171
```

Packages installed successfully: Installed packages: 'matplotlib' (today 11:25)

Genetic Algorithm GUI

Algorithm type: Original_vs_HitStamp

Crossover type: SINGLE

Run Genetic Algorithm

```
Best Individual: $ello, world!
Best Fitness: 12
Best Individual: Hell!, wodld!
Best Fitness: 171
```

The fitness scores of the individuals in the population should increase over time, which is reflected in the histograms as a shift towards the right. The second histogram (with the hit stamp heuristic) should show a higher peak and a wider range of fitness scores compared to the first histogram (with the original fitness function), as the hit stamp heuristic gives a higher fitness score to individuals that have guessed letters in the right place or in the wrong place.

7. ציינו ונמקו אלו חלקים באלגוריתם אחראיים ל EXPLORATION ואילו לEXPLOITATION?

In the genetic algorithm, exploration refers to the process of searching for new and diverse solutions in the search space, while exploitation refers to the process of refining and improving existing solutions that are already found.

Parts of the algorithm responsible for exploration are:

1. Random initialization of the population - the genetic algorithm starts with a random population, which allows for a wide range of diversity in the initial solutions.
2. Selection of parents for reproduction - the genetic algorithm uses a selection process to choose parents for mating, which ensures that a variety of individuals in the population have a chance to be chosen as parents.

3. Application of crossover and mutation operators - the crossover and mutation operators introduce new genetic material to the population, which allows for exploration of new solutions in the search space.

Parts of the algorithm responsible for exploitation are:

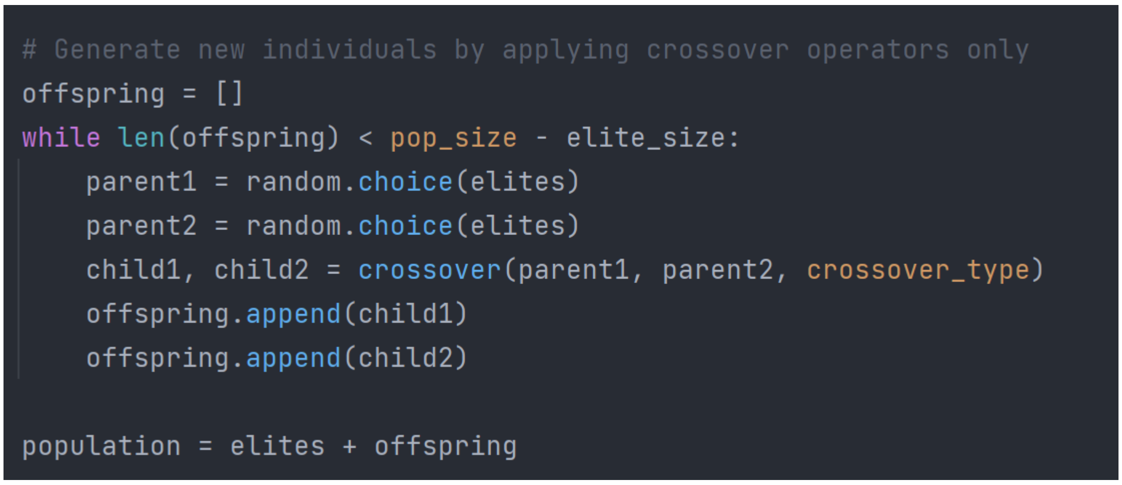
1. Fitness evaluation and selection of the fittest individuals - the genetic algorithm evaluates the fitness of each individual in the population and selects the fittest ones to be part of the next generation, which ensures that the best solutions are preserved and improved over time.
2. Elitism - a portion of the fittest individuals from the current generation is carried over to the next generation unchanged, which helps to prevent the loss of good solutions that have already been found.
3. Application of crossover and mutation operators - the crossover and mutation operators can also improve the quality of existing solutions in the population, by introducing beneficial genetic changes that lead to better fitness scores.

Overall, the genetic algorithm strikes a balance between exploration and exploitation to efficiently search for high-quality solutions in the search space.

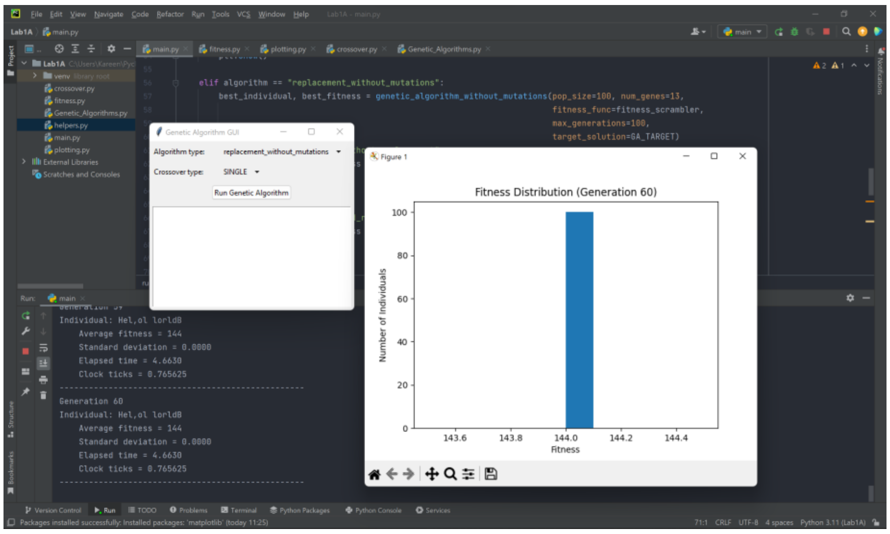
8. השוו בין מצב בו אתם מריצים את האלגוריתם בקונפיגורציות הבאות
   a. רק עם שיחלוף ללא מוטציות
   b. רק עם מוטציות ללא שיחלוף
   c. עם שניהם
   d. מה המסקנות שלכם מריצות אלו

a. Only with replacement without mutations:

For this configuration, we will only apply the crossover operator without any mutations. This means that we will generate new individuals by selecting parents from the previous generation and applying crossover, without any random changes to their genes. This configuration will allow the best individuals from the previous generation to be carried over to the next generation unchanged, which could potentially lead to stagnation and a lack of diversity in the population.


```python
# Generate new individuals by applying crossover operators only
offspring = []
while len(offspring) < pop_size - elite_size:
    parent1 = random.choice(elites)
    parent2 = random.choice(elites)
    child1, child2 = crossover(parent1, parent2, crossover_type)
    offspring.append(child1)
    offspring.append(child2)

population = elites + offspring
```



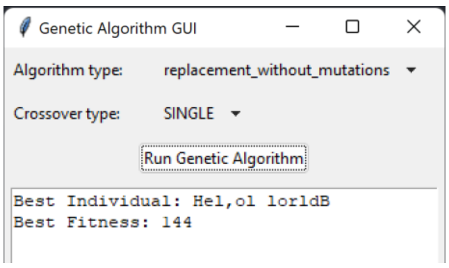
```
elif algorithm == "replacement_without_mutations":
    best_individual, best_fitness = genetic_algorithm_without_mutations(pop_size=100, num_genes=13,
                                                                        fitness_func=fitness_scrambler,
                                                                        max_generations=100,
                                                                        target_solution=GA_TARGET)
```

Algorithm type: replacement_without_mutations

Crossover type: SINGLE

```
Individual: Hel,ol lorldB
    Average fitness = 144
    Standard deviation = 0.0000
    Elapsed time = 4.6630
    Clock ticks = 0.765625
--------------------------------------------------
Generation 60
Individual: Hel,ol lorldB
    Average fitness = 144
    Standard deviation = 0.0000
    Elapsed time = 4.6630
    Clock ticks = 0.765625
--------------------------------------------------
```

Genetic Algorithm GUI
Run Genetic Algorithm
Figure 1
Fitness Distribution (Generation 60)
Number of Individuals
Fitness
100
80
60
40
20
0
143.6
143.8
144.0
144.2
144.4
Packages installed successfully: Installed packages: 'matplotlib' (today 11:25)



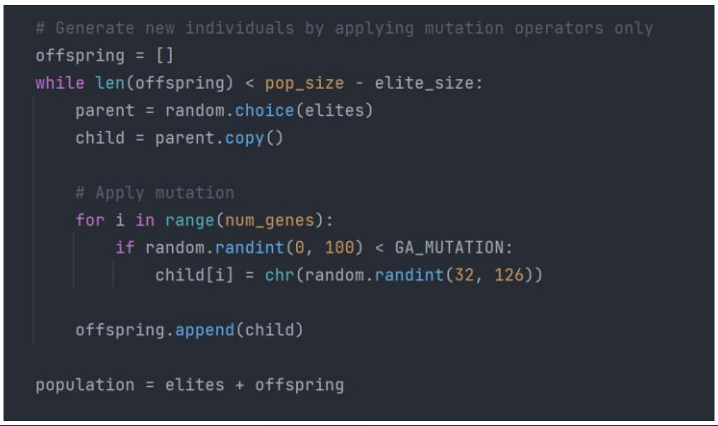
Algorithm type: replacement_without_mutations

Crossover type: SINGLE

```
Best Individual: Hel,ol lorldB
Best Fitness: 144
```

b. Only with mutations without replacement:

For this configuration, we will only apply the mutation operator without any crossover. This means that we will generate new individuals by randomly mutating genes in the current population without using any information from the previous generation. This configuration will introduce new genetic diversity into the population, but will not allow for any beneficial gene combinations to be preserved from the previous generation.

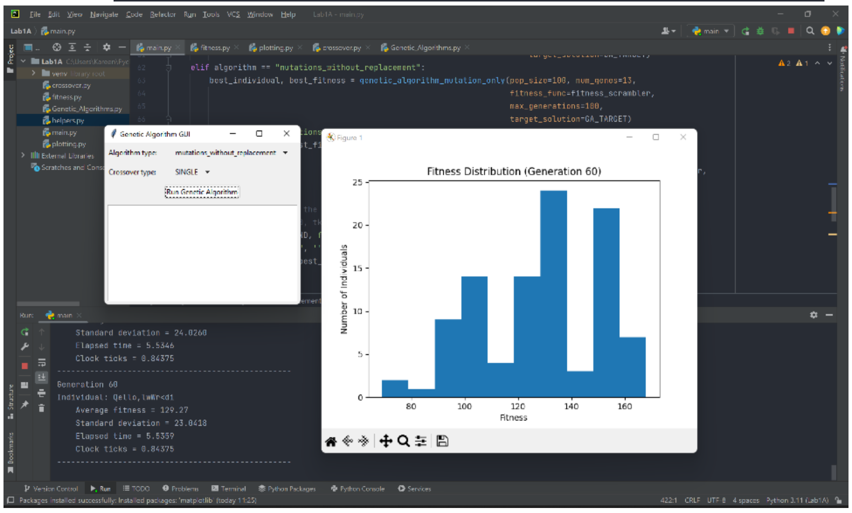


```python
# Generate new individuals by applying mutation operators only
offspring = []
while len(offspring) < pop_size - elite_size:
    parent = random.choice(elites)
    child = parent.copy()

    # Apply mutation
    for i in range(num_genes):
        if random.randint(0, 100) < GA_MUTATION:
            child[i] = chr(random.randint(32, 126))

    offspring.append(child)

population = elites + offspring
```



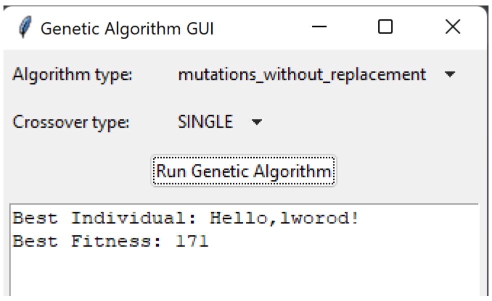
```
elif algorithm == "mutations_without_replacement":
    best_individual, best_fitness = genetic_algorithm_mutation_only(pop_size=100, num_genes=13,
                                                                    fitness_func=fitness_scrambler,
                                                                    max_generations=100,
                                                                    target_solution=GA_TARGET)
```

Algorithm type: mutations_without_replacement

Crossover type: SINGLE

```
    Standard deviation = 24.0260
    Elapsed time = 5.5346
    Clock ticks = 0.84375
--------------------------------------------------
Generation 60
Individual: Qello,lwWr<d1
    Average fitness = 129.27
    Standard deviation = 23.0418
    Elapsed time = 5.5359
    Clock ticks = 0.84375
--------------------------------------------------
```

Genetic Algorithm GUI
Run Genetic Algorithm
Figure 1
Fitness Distribution (Generation 60)
Number of Individuals
Fitness
25
20
15
10
5
0
80
100
120
140
160
Packages installed successfully: Installed packages: 'matplotlib' (today 11:25)



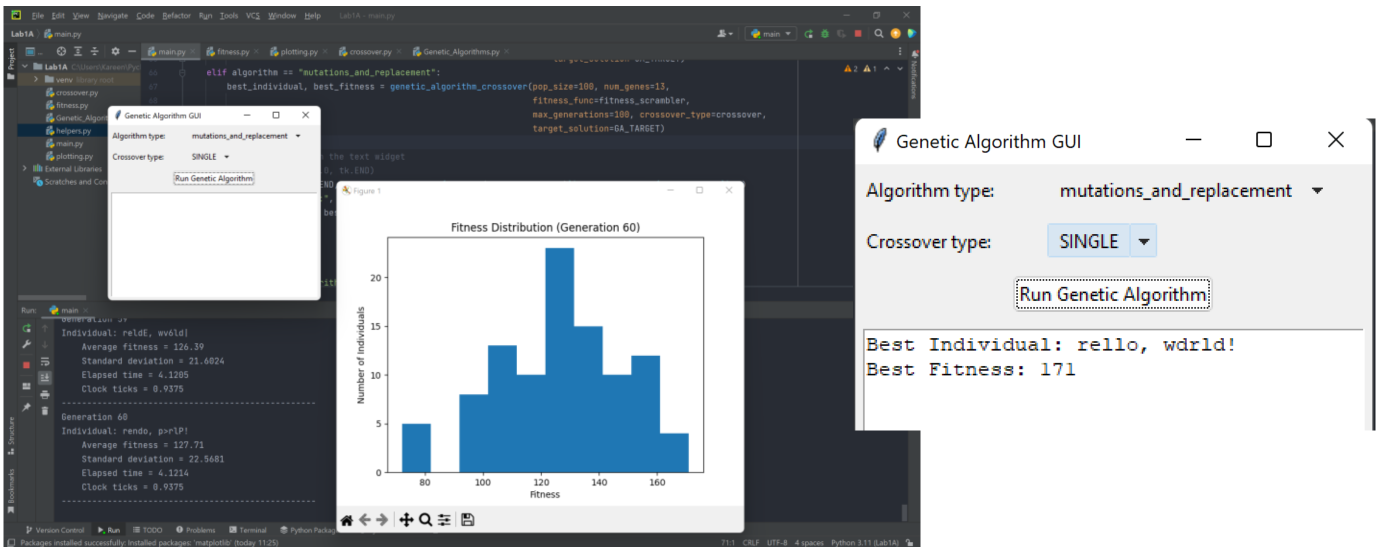
Algorithm type: mutations_without_replacement

Crossover type: SINGLE

```
Best Individual: Hello,lworod!
Best Fitness: 171
```

c. With both:

For this configuration, we will apply both crossover and mutation operators. This means that we will generate new individuals by selecting parents from the previous generation, applying crossover to create new gene combinations, and then randomly mutating some genes in the resulting offspring. This configuration will combine the benefits of both the previous configurations: preserving the best gene combinations from the previous generation, while also introducing new genetic diversity.



```python
elif algorithm == "mutations_and_replacement":
    best_individual, best_fitness = genetic_algorithm_crossover(pop_size=100, num_genes=13,
                                                                fitness_func=fitness_scrambler,
                                                                max_generations=100, crossover_type=crossover,
                                                                target_solution=GA_TARGET)
```

Algorithm type: mutations_and_replacement

Crossover type: SINGLE

```
Individual: reldE, wv6ld|
    Average fitness = 126.39
    Standard deviation = 21.6024
    Elapsed time = 4.1205
    Clock ticks = 0.9375
--------------------------------------------------
Generation 60
Individual: rendo, p>rlP!
    Average fitness = 127.71
    Standard deviation = 22.5681
    Elapsed time = 4.1214
    Clock ticks = 0.9375
--------------------------------------------------
```

```
Best Individual: rello, wdrld!
Best Fitness: 171
```

Packages installed successfully: Installed packages: 'matplotlib' (today 11:25)

d. Conclusions:

Conclusion:

- Using both replacement and mutation operators in the genetic algorithm leads to faster convergence towards the target and maintains diversity in the population.
- The mutation operator helps the algorithm to explore new regions of the search space and potentially find better solutions.
- The replacement operator helps the algorithm to exploit the current solutions and preserve the best individuals over time.

9. השוו בסימולציה בין ביצועי שני האלגוריתמים לגבי הבעיה הנתונה תחת ההיוריסטיקה המועדפת והפרמטריזציה המיטבית – התיחסו לאיכות הפתרונות ולמהירות ההגעה אליהם

We added a histogram to show the differences, and also the simulation's results are printed (can see in the running code).

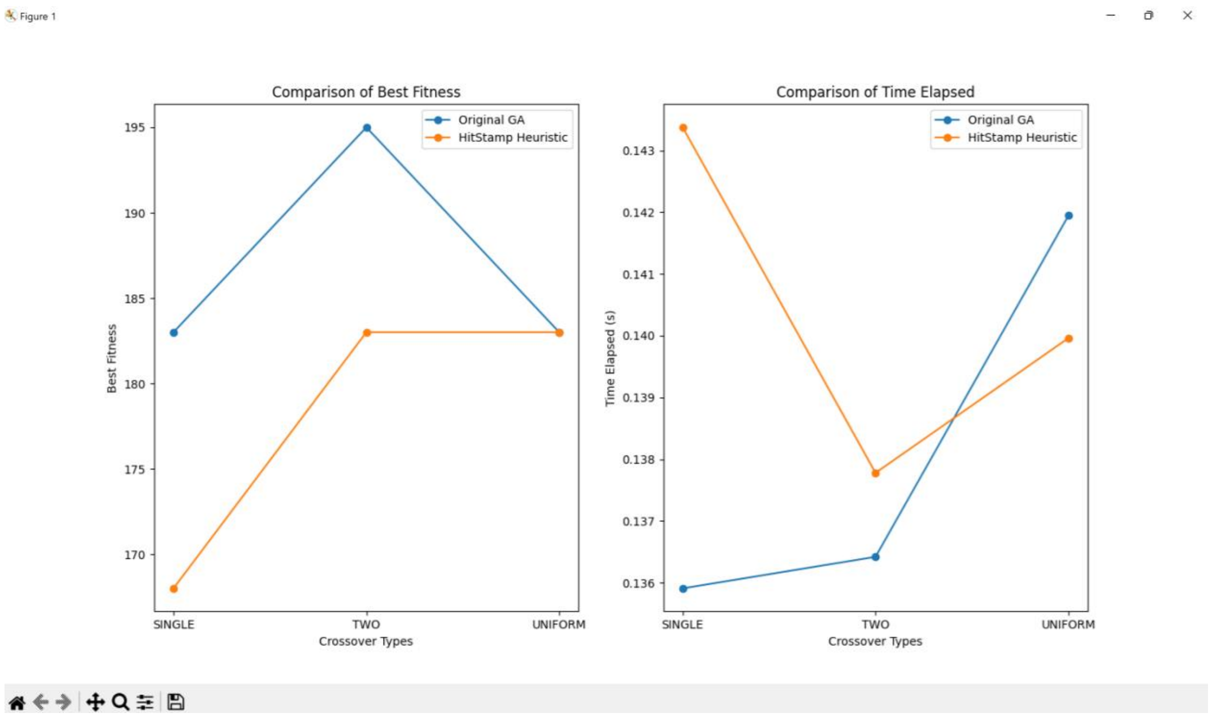


Figure 1
Comparison of Best Fitness
Best Fitness
Crossover Types
SINGLE
TWO
UNIFORM
Original GA
HitStamp Heuristic
Comparison of Time Elapsed
Time Elapsed (s)

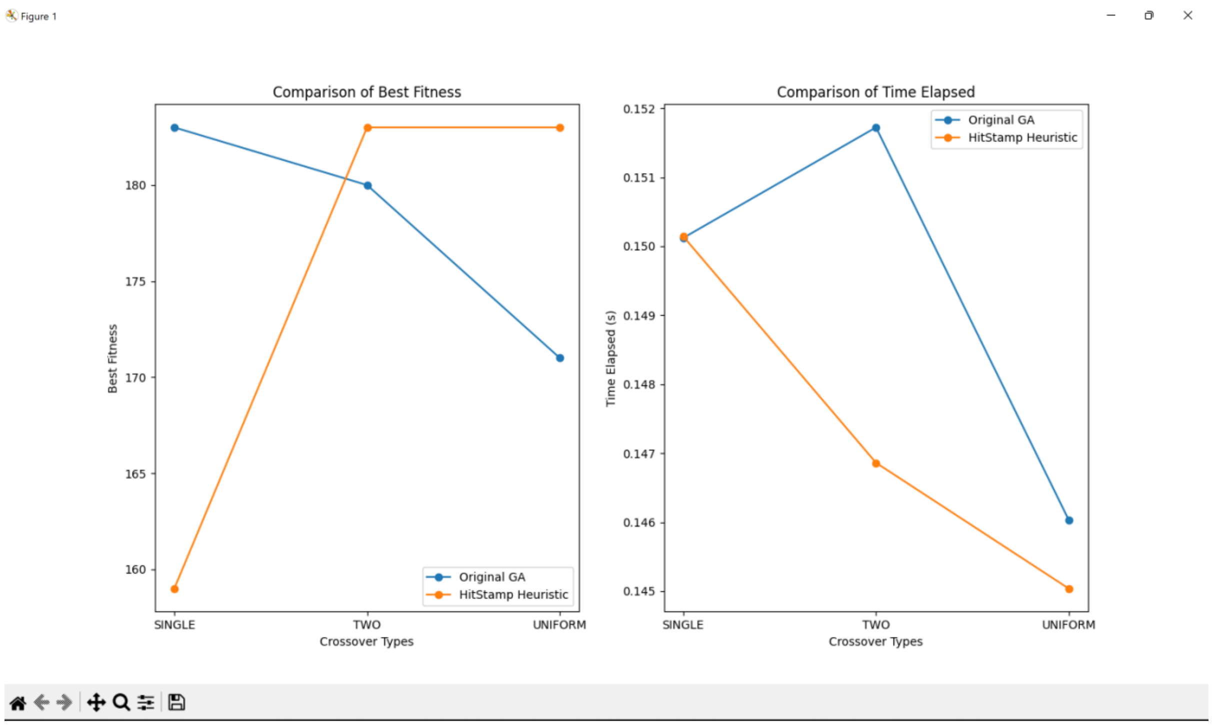


Figure 1
Comparison of Best Fitness
Best Fitness
Crossover Types
SINGLE
TWO
UNIFORM
Original GA
HitStamp Heuristic
Comparison of Time Elapsed
Time Elapsed (s)