- Parameter settings

```
gene_length = 30   #二進制的長度
population_size= 500   #每一代染色體的數量
mutation_rate = 0.5   #變異機率
tolerance = 1e-11     # 停止條件
aus = 1e6
target = float(input("請輸入您要計算的根號值："))
```

- def Binary(chromosome)

這個部分主要是將我的數值轉為二進制的定義

- def decimal_number(chromosome)

這個部分主要是將我的數值轉為十進制的定義

- def generate_population(population_size, gene_length)

生成初始族群的函數

- def fitness_function(individual)

```
# 評估染色體適應度的函數
    chromosome = decimal_number(individual) / (10**11)
    score = aus - abs(chromosome**2 - target)
    return max(0, score)   #確保適應度值大於等於零
```

def crossover(parent1, parent2)

```
# 交叉遺傳的函數
    # 隨機選擇交叉點，避免在邊界處交叉
    point = random.randint(1, len(parent1) - 2)
    # 生成子代 1，將父母 1 的前段和父母 2 的後段結合
    child1 = parent1[:point] + parent2[point:]
    # 生成子代 2，將父母 2 的前段和父母 1 的後段結合
    child2 = parent2[:point] + parent1[point:]
    return child1, child2
```

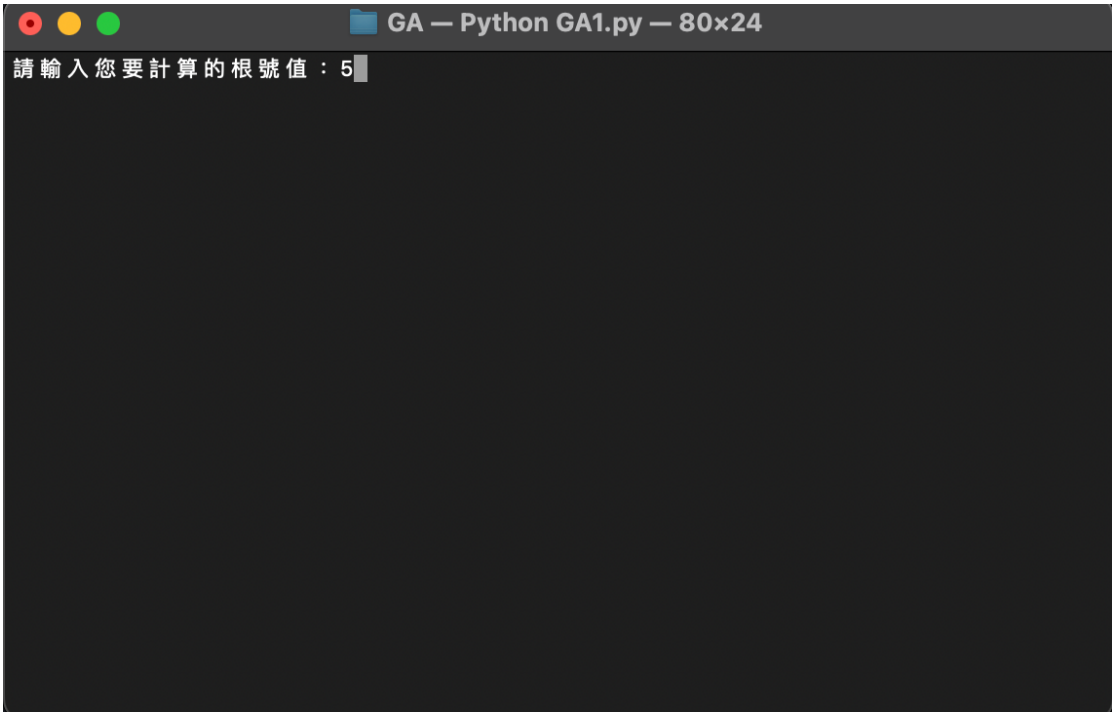
- def mutate(individual)

```
# 變異的函數
    # 隨機選擇染色體中的一個位置
    mutated_index = random.randint(0, len(individual) - 1)
    # 將選定位置的基因取反，即 0 變為 1，1 變為 0
    mutated_gene = '0' if individual[mutated_index] == '1' else '1'
    # 產生變異後的染色體，將選定位置的基因替換為變異後的基因
    return individual[:mutated_index] + mutated_gene +
individual[mutated_index + 1:]
```

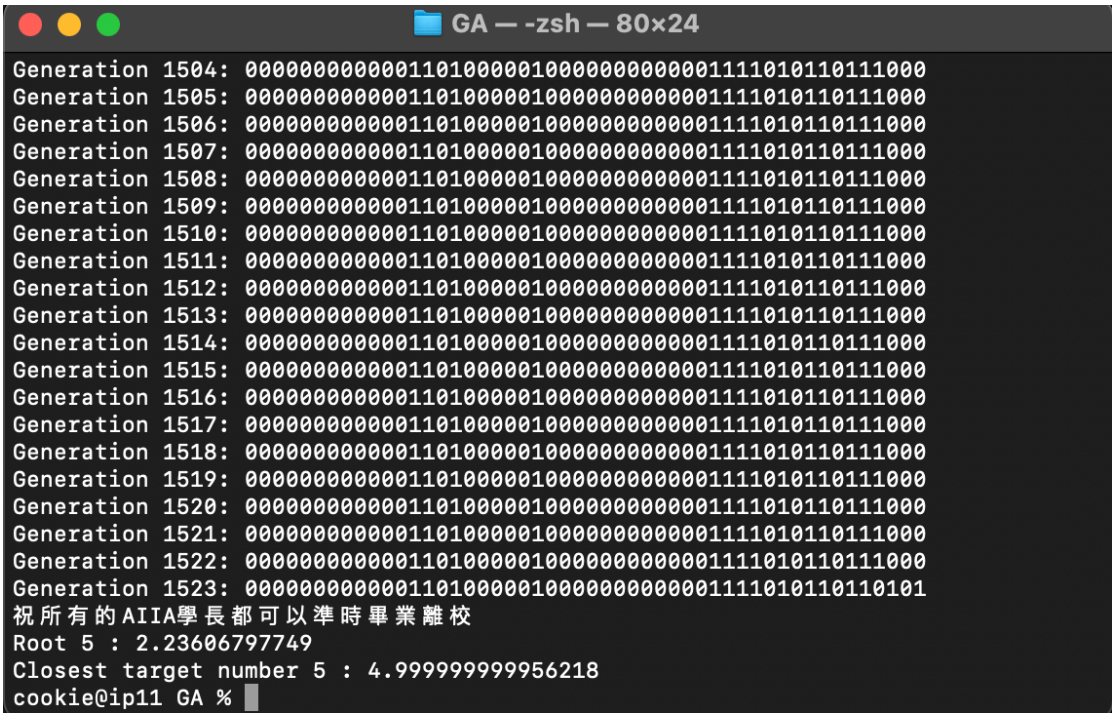
- def genetic_algorithm()

遺傳演算法主要的函數，會先生成一個族群，然後將族群按照適應的程度去排序，產生最佳的染色體，這邊也會去選擇父母的染色體，進行交叉遺傳，然後變異，將好的子# 檢查是否達到停止條件，子代留下來，產生新的族群

（以下為執行結果）



（圖一）可以輸入需要計算的根號數字



（圖二）為輸入(根號)5 所得到最終的結果

（以下為程式碼）

```python
import random


gene_length = 50 #binary length
population_size= 1000 #The number of chromosomes in each generation
mutation_rate = 0.4 #Mutation probability
tolerance = 1e-11   # Stop condition
aus = 1e6
# Enter the target number to be calculated
target = float(input("請輸入您要計算的根號值："))

def Binary(chromosome):
    #Convert to binary(str)
    binary_string=bin(chromosome)[2:]
    return binary_string

def decimal_number(chromosome):
    #轉乘10進制(str)
    return int(chromosome,2)

def generate_population(population_size,gene_length):
    return[''.join(random.choice('01') for _ in range(gene_length)) for _ in range(population_size)]


def fitness_function(individual):
    chromosome = decimal_number(individual) / (10**11)
    score = aus - abs(chromosome**2 - target)
    return max(0, score)  # 确保适应度值大于等于零

def select_parents(population):
    #選擇2個染色體(當父母)
    #population為這一代中所有的父母
    return random.choices(population, k=2, weights=[fitness_function(ind) for ind in population])

def crossover(parent1, parent2):
    #單點交叉(交配)
    #(1, len(parent1) - 2)是為了避免選到投跟尾
    point = random.randint(1, len(parent1) - 2)
    child1 = parent1[:point] + parent2[point:]
    child2 = parent2[:point] + parent1[point:]
    return child1, child2

def mutate(individual):
    #選擇一個點進行變異
    #使1-->0 和 0-->1
    mutated_index = random.randint(0, len(individual) - 1)
    mutated_gene = '0' if individual[mutated_index] == '1' else '1'
    return individual[:mutated_index] + mutated_gene + individual[mutated_index+1:]

def genetic_algorithm():
    #Randomly generate the first generation chromosomes
    population=generate_population(population_size,gene_length)
    generation=0

    while True:
        generation+=1
        #Rank dyeing replacements with high scores ranked first
        population=sorted(population,key=fitness_function,reverse=True)
        new_population=[population[0]]
        #Generate the next generation (population_size chromosomes)
        while len(new_population)<population_size:
            #Choose parents to have children
            parent1,parent2=select_parents(population)
            child1,child2=crossover(parent1,parent2)

            #Mutations
            if random.random()<mutation_rate:
                child1=mutate(child1)
            if random.random()<mutation_rate:
                child2=mutate(child2)

            new_population.extend([child1,child2])

        population=new_population

        #The best solution of this generation
        ans=population[0]
        a=fitness_function(ans)
        print(f"Generation {generation}: {ans}")
        # Check whether the stopping condition is met
        if -tolerance < aus - a < tolerance:
            break
        # print(f"100 - a: {100 - a}")
        # print(f"a - target = {a - target}")

    print('祝所有的AIIA學長都可以準時畢業離校')
    print(f'Root {int(target)} :', decimal_number(ans) / 10**11)
    print(f'Closest target number {int(target)} :', (decimal_number(ans) / 10**11)**2)

if __name__ == '__main__':
    genetic_algorithm()



# ans = # 2.23606797749979
```