الهدف من الجلسة:

التعرف على الخوارزمية الجينية وقدرتها على إيجاد حلول مثالية أو شبه مثالية .

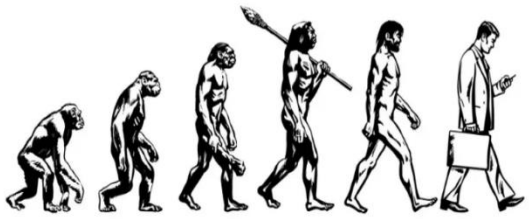
مقدمة:

الخوارزمية الجينية هي نوع من الخوارزميات التي تحاكي نظرية أو عملية التطور لإيجاد حلول optimal او near optimal لمشاكل معقدة.

تبحث الخوارزمية عن الحل ضمن مجتمع من الحلول population تتولد بطريقة عشوائية يسمى كل حل chromosome ويتم اختيار مجموعة جزئية من الحلول لتتم بعد ذلك مرحلة cross over والتي ينتج عنها جيل جديد من الحلول وبعد ذلك تتم عملية mutation (طفرة) ومن ثم تبديل الحلول ضمن population بحلول جديدة تم توليدها وفقا للعمليات السابقة.

نستنتج خطوات الخوارزمية الجينية:

1. توليد المجتمع population
2. Selection
3. Cross over
4. Mutation
5. Replace



Evolution

سيتم تطبيق الخوارزمية الجينية أثناء البحث عن حل لمشكلة إيجاد سلسة نصية "hello world" ابتداء من مجموعة حلول عشوائية

```
Import random
TARGET=”hello world”
Pop_size= 500
Mut_rat=0.1
GENES=” abcdefghijklmnpoqrstuvwxyz”
```

1. Initialization

```
def initialize_pop(TARGET):
  population = list()
  tar_len = len(TARGET)

  for i in range(POP_SIZE):
      temp = list()
      for j in range(tar_len):
          temp.append(random.choice(GENES))
      population.append(temp)

  return population
```

2. Fitness

```
def fitness_cal(TARGET, chromo_from_pop):
  difference = 0
  for tar_char, chromo_char in zip(TARGET, chromo_from_pop):
      if tar_char != chromo_char:
          difference+=1
  return [chromo_from_pop, difference]
```

يتم حساب أو قياس بُعد الكروموزوم عن الحل target باستخدام تابع fitness من خلال هذا التابع ممكن اختبار الحل ومن الممكن اعتماده أثناء عملية اختيار مجموعة جزئية من الحلول قبل عملية cross over

3. Selection

```
def selection(population, TARGET):
  sorted_chromo_pop = sorted(population, key= lambda x: x[1])
  return sorted_chromo_pop[:int(0.5*POP_SIZE)]
```

4. Cross over

```
def crossover(selected_chromo, CHROMO_LEN, population):
  offspring_cross = []
  for i in range(int(POP_SIZE)):
    parent1 = random.choice(selected_chromo)
    parent2 = random.choice(population[:int(POP_SIZE*50)])

    p1 = parent1[0]
    p2 = parent2[0]

    crossover_point = random.randint(1, CHROMO_LEN-1)
    child =  p1[:crossover_point] + p2[crossover_point:]
    offspring_cross.extend([child])
  return offspring_cross
```

5. Mutation

```
def mutate(offspring, MUT_RATE):
  mutated_offspring = []

  for arr in offspring:
      for i in range(len(arr)):
          if random.random() < MUT_RATE:
              arr[i] = random.choice(GENES)
      mutated_offspring.append(arr)
  return mutated_offspring
```

6. Replace

```
def replace(new_gen, population):
  for _ in range(len(population)):
      if population[_][1] > new_gen[_][1]:
        population[_][0] = new_gen[_][0]
        population[_][1] = new_gen[_][1]
  return population
```