Nama : Stephen Prasetya Chrismawan
NIM : H1D021025

1. Mengimpor library yang dibutuhkan
   Menginisiasikan nilai awal yaitu jumlah individu, gen, generasi.

```
import random

import matplotlib.pyplot as plt

import numpy as np


# Menginisiasikan nilai

populasi = []

jml_individu = 50

jml_gen = 10

p = 0.5

jumlah_generasi = 500
```

2. Mendefinisikan fungsi untuk menghitung nilai fitness, dengan argumen pertama nilai decimal x1, dan argumen kedua nilai decimal x2

```
def fitness_function(x1,x2):

   return 3*x1 - 2*x2
```

3. Membuat funsgi untuk menghitung nilai biner ke decimal. Dengan argumen pertama adalah list / array bit bit individu, arg kedua adalah jumlah gen gabungan yaitu 10.
   Perhitungannya adalah dengan melakukan per-kuadratan mundur dari value di index terdepan, diawali dengan pangkat 4 diakhiri dengan pangkat 0.

```
def bintodecimal(X,jum_gen_asli):

   decimal = 0

   jumlah_pangkat_bit = jum_gen_asli/2-1

   for k in X:

       decimal = decimal+ ( k * 2 ** jumlah_pangkat_bit)

       jumlah_pangkat_bit = jumlah_pangkat_bit-1

   return decimal
```

4. Membuat class Individu, supaya setiap indivdu memiliki karakteristiknya masing masing, setiap individu yang di generate akan memiliiki id, bit

yang akan dipecah menjadi bitx1 dan bitx2, jadiparent (berfungsi pada tahap pemilihan parent), nilai probabilitas yang akan selalu diupdate kelak saat pembuatan generasi baru. Kemudian juga menyimpan nilai decimal dari bitx1 dan decimal bitx2, kemudian mempunyai nilai fitness.

```
class Individu :

   def __init__(self, bit, id) :

       self.id = id

       self.bit = bit

       self.jadiparent = 0

       self.probabilitas = 0

       self.bitx2= self.bit[int(jml_gen/2):int(jml_gen)]

       self.probkumulatif = 0

       self.bitx1= self.bit[0:int(jml_gen/2)]

       self.bitx2= self.bit[int(jml_gen/2):int(jml_gen)]


       self.decimalx1 = bintodecimal(self.bitx1, jml_gen)

       self.decimalx2 = bintodecimal(self.bitx2, jml_gen)

       self.fitness = fitness_function(self.decimalx1,self.decimalx2)
```

5. Berikut merupakan langkah generator individu sampai memenuhi 50 individu. Diperulangan ini juga diisikan id yatu sebagai penanda identitas masing masing individu, dibuat berdasarkan perulangan loop. Kemudian setiap individu yang telah terbentuk akan dimasukkan ke dalam array populasi

```
iloop = 1

while iloop <= jml_individu :

   calon = []


   #Melakukan perulangan untuk membuat 1 per satu gen

   for j in range(jml_gen):

       #melakukan pemilihan pengacakan antara 0 sampai 1 dengan digit
maksimal 4

       r = round(random.uniform(0, 1), 4)

       if r<p :

           r = 0 #jika nilai random kurang dari 0,5 maka jadi 0
```

```
        else :
            r = 1 #jika nilai random lebih dari 0,5 maka jadi 1


        #Menyisipkan gen (nilai r) ke calon individu
        calon.append(r)



    #Memasukan setiap individu ke kelas individu
    individu = Individu(calon, iloop)
    if individu.decimalx1 >=3 and individu.decimalx1<=10 and
individu.decimalx2 >=3 and individu.decimalx2<=10 and
individu.fitness>0:
        #Menyisipkan individu ke populasi
        populasi.append(individu)
        iloop+=1
    else :
        continue
```

6. Langkah selanjutnya, yaitu membuat presentase probabilitas masing masing individu. Pertama, dengan mencari total nilai fitness setiap individu. Kemudian, menghitung probabilitas setiap indivdu dengan me-loop , nilai fitness nya dibagi dengan total nilai fitness keseluruhan. Menghitung probabilitas kumulatif dengan cara menjumlahkan seluruh probabilitas dan memasukkannya ke class individu.

```
sum = 0
for i in populasi    :
    sum += i.fitness


urut = 1
probkumulatif = 0
for i in populasi    :
    i.probabilitas = i.fitness /sum
    probkumulatif+=i.probabilitas
    # print(f"\nProbabilitas Individu ke -{urut}= {i.probabilitas}")
    # print(f"\nProbabilitas Kumulatif = {probkumulatif}")
```

```
    i.probkumulatif = probkumulatif

    urut +=1
```

7. Kemudian, saya mencoba menampilkan keseluruhan individu awal.

```
urut = 1

for manusia in populasi   :

    print(f"Bit Individu ke - {urut} = {manusia.bit}")

    print(f"Fitness Individu ke - {urut} = {manusia.fitness}")

    print(f"Probabilitas Individu ke - {urut} = {manusia.probabilitas}")

    print(f"Prob Kumulatif Individu ke - {urut} =

{manusia.probkumulatif}")

    urut +=1
```

8. Kemudian, masuk ke tahapan ke perputaran generasi. Dimulai dari tahapan seleksi menggunakan probabilitas kumulatif.
   Array sumbu di atasnya digunakan untuk menyimpan nilai fitness rata rata setiap generasi ke dalam sumbu y.

   Penentuan Parent 1, didasarkan pada terdapat perulangan di awal adalah untuk mengidentifikasi, jika jika ada kemungkinan nilai random yang terdapat pada angka random tidak menemukan probabilitas kumulatifnya di semua individu. Begitupun dengan parent 2.

   Pengecekan seleksi probabilitas kumulatif terjadi di
   if l.probkumulatif < s
   Bahwa ketika nilai random (s) lebih besar dari nilai probabilitas kumulatif individu yang di loop maka akan di *continue* ke perulangan berikutnya
   Terdapat juga pengecekan or l.jadiparent == 1:
   Artinya ketika individu sudah pernah menjadi parent di generasi yang sama maka jangan dipilih dan lanjutkan ke perulangan berikutnya. Ini sangat penting ketika sedang memilih parent2

   Ketika parent berhasil terpilih maka buat individu telah jadi parent , copykan bit individu ke array Parent baru, copy kan juga id nya untuk berjaga jaga.

```
sumbuy=[]
igen = 1
while igen <= jumlah_generasi:
   # Tahap Seleksi menggunakan Probabilitas Kumulatif
   #random angka dari 0 - 1

   Parent1 =[]
   idp1 = 0
   dapet = 0
   while dapet  == 0:
       s = round(random.uniform(0,1), 4)
       for l in populasi :
           if l.probkumulatif < s or l.jadiparent == 1:
               continue
           elif l.probkumulatif >= s:
               l.jadiparent = 1
               Parent1 = l.bit.copy()
               idp1 = l.id
               dapet =1
               break
   Parent2 = []
   idp2 = 0
   dapet = 0
   while dapet  == 0:
       r = round(random.uniform(0,1), 4)
       for j in populasi :
           if j.probkumulatif < r or j.jadiparent == 1 :
               continue
           elif j.probkumulatif >= r:

               j.jadiparent = 1
               Parent2 = j.bit.copy()
               idp2 = j.id
               dapet =1
```

```
            break
```

9. Tahap selanjutnya adalah crossover yaitu saya akan menggunakan unform crossover , saya memulai dengan membuat array kosong untuk masking. Kemudian saya melakukan generate masking secara random dan otomatis agar dinamis.

```
masking_co1 = []


   # Mengisi list masking dengan nilai acak antara 0 dan 1
   for _ in range(10):
       nilai_acak = random.randint(0, 1)
       masking_co1.append(nilai_acak)
```

10. Langkah berikutnya, yaitu saya membuat perulangan supaya ketika masking bernilai 1 maka saya memasukan nilai parent 2 ke array child 1 baru, dan sebaliknya. Jika bernilai 0 maka tidak saya tukar dan saya masukan sesuai parentnya.

```
child1 = []
   child2 = []
   i = 0
   while i < len(masking_co1) and i < len(Parent1) and i < len(Parent2):
       if masking_co1[i]==1 :
           child1.append(Parent2[i])
           child2.append(Parent1[i])
       elif masking_co1[i]==0 :
           child1.append(Parent1[i])
           child2.append(Parent2[i])
       i+=1
```

11. Kemudian, saya jadikan jadiparent menjadi 0 lagi supaya di generasi berikutnya pemilihan parent dapat berjalan lancar.

```
for j in populasi :
           j.jadiparent = 0
```

12. Langkah berikutnya saya melakukan mutasi. Jadi, saya menentukan probabilitas mutasi terlebih dahulu, sebagai nilai acuan mana index child yang bisa dimutasi. Saya memberikan nilai random 0-1 kemudian dibandingkan dengan prob mutasi, jika kurang dari prob mutasi maka nilai gen akan dibalik, 0 menjadi 1 dan 1 menjadi 0. Dilakukan pada kedua child. Masukan ke array baru yaitu childtermutasi

```
prob_mutasi = 1/jml_gen


    #Mutasi Child 1
    child1termutasi = []
    randommutasi = random.uniform(0,1)
    for k in child1 :
        if randommutasi<prob_mutasi :
            if k==0:
                child1termutasi.append(1)
            elif k==1:
                child1termutasi.append(0)
        else :
            child1termutasi.append(k)
    #Mutasi Child 2
    child2termutasi = []
    randommutasi = random.uniform(0,1)
    for k in child1 :
        if randommutasi<prob_mutasi :
            if k==0:
                child2termutasi.append(1)
            elif k==1:
                child2termutasi.append(0)
        else :
            child2termutasi.append(k)
```

13. Kemudian melakukan mutasi kedua yaitu, memilih 2 index, kemudian menukar nilai 2 index tersebut

```
#MUTASI KEDUA
    #menggunakan swap mutation
    #CHILD 1
```

```
    index1=0


    index2 =0
    while 1 :


        index1 = random.randint(0,jml_gen-1)
        index2 = random.randint(0,jml_gen-1)
        if index1!=index2:
            break


    child1termutasi[index1],  child1termutasi[index2] =
child1termutasi[index2],  child1termutasi[index1]


    #CHILD 2
    index1=0


    index2 =0
    while 1 :


        index1 = random.randint(0,jml_gen-1)
        index2 = random.randint(0,jml_gen-1)
        if index1!=index2:
            break


    child2termutasi[index1],  child2termutasi[index2] =
child2termutasi[index2],  child2termutasi[index1]
```

14. Langkah terakhir adalah melakukan subtitusi child 1 dan atau 2 ke dalam populasi dengan menggantikan individu yang memiliki nilai fitness paling kecil.
    Namun, sebelum itu saya melakukan pengecekan terhadap nilai decimal dari child 1 dan child 2, saya melakukan pengkondisian bahwa saya harus meloloskan child yang hanya memiliki nilai decimal bit x1 3-10 dan bit x2 3-10.

Kemudian saya melakukan pencarian fitness minimum yaitu dnegan cara saya mendeklarasikan bahwa nilai minim fitness adalah 24 yaitu fitness tertinggi. Jika terdapat nilai fitness di populasi tersebut yang kurang dari 24 maka saya simpan sebagai nilai minim fitness terbaru, saya simpan di minim bit, dan minim id untuk mengidentifikasinya lebih lanjut. Saya melakukan perulangan hingga semua individu di populasi tersebut dicek.

```
#mencari dan menggantikan nilai fitness terendah
   child1x1 = child1termutasi[0:int(jml_gen/2)]
   child1x2 = child1termutasi[int(jml_gen/2):int(jml_gen)]
   decimalchild1x1 = bintodecimal(child1x1, jml_gen)
   decimalchild1x2 = bintodecimal(child1x2, jml_gen)
   fitnesschild1 = fitness_function(decimalchild1x1,decimalchild1x2)
   child2x1 = child2termutasi[0:int(jml_gen/2)]
   child2x2 = child2termutasi[int(jml_gen/2):int(jml_gen)]
   decimalchild2x1 = bintodecimal(child2x1, jml_gen)
   decimalchild2x2 = bintodecimal(child2x2, jml_gen)
   fitnesschild2 = fitness_function(decimalchild2x1,decimalchild2x2)

   #Perhitungan dan Penggantian dari Child 1
   if decimalchild1x1 >=3 and decimalchild1x1<=10 and decimalchild1x2
>=3 and decimalchild1x2<=10:
       minimfitness = 24

       minimID = 0
       minimbit = []
       for p in populasi :
           if p.fitness <minimfitness :
               minimfitness = p.fitness
               minimbit = p.bit
               minimID = p.id
           else:
               continue
```

15. Kemudian saya melakukan subtitusi yaitu dengan melakukan pencarian terhadap minimID yang telah tersimpan sebagai individu yang memiliki ID paling kecil. Kemudian setelah ditemukan individu dengan nilai ID seperti itu maka nilai bit, nilai fitness, dan nilia decimal akan digantikan oleh child baru.

```
if decimalchild2x1 >=3 and decimalchild2x1<=10 and decimalchild2x2 >=3
and decimalchild2x2<=10:
      minimfitness = 24

      minimID = 0
      minimbit = []
      for p in populasi :
          if p.fitness <minimfitness :
              minimfitness = p.fitness
              minimbit = p.bit
              minimID = p.id
          else:
              continue
      for q in populasi :
          if q.id == minimID:
              q.bit =  child2termutasi
              q.fitness = fitnesschild2
              q.decimalx1 = decimalchild2x1
              q.decimalx2 = decimalchild2x2
              break
          else:
              continue
```

16. Langkah selanjutnya, saya menghitung ulang untuk probabilitas dan probabilitas kumulatif dari setiap individu.

```
  sum = 0
  for i in populasi   :
      sum += i.fitness

  urut = 1
```

```
    probkumulatif = 0
    for i in populasi    :
        i.probabilitas = i.fitness /sum
        probkumulatif+=i.probabilitas
        i.probkumulatif = probkumulatif
        urut +=1
```

17. Kemudian saya menghitung nilai rata-rata nilai fitness per generasi dan saya masukkan ke array sumbu y.

```
    #Perhitungan rata rata nilai fitness generasi ke generasi

    avg = sum/jml_individu
    sumbuy.append(avg)
    print(f"\nFitness average Generasi ke - {igen} = {avg}")
    igen+=1
```

18. Langkah terakhir adalah menuliskan detail setiap individu terakhir di terminal dan menghasilkan diagram menggunakan matplotlib.

```
urut = 1
print(f"\n50 Individu Hasil Iterasi Terakhir :")
for manusia in populasi    :

    print(f"Bit Individu ke - {urut} = {manusia.bit}")
    print(f"Fitness Individu ke - {urut} = {manusia.fitness}")
    print(f"Probabilitas Individu ke - {urut} = {manusia.probabilitas}")
    print(f"Prob Kumulatif Individu ke - {urut} =
{manusia.probkumulatif}")
    urut +=1
i=1
sumbux=[]
while i <= jumlah_generasi:
```

```
    sumbux.append(i)
    i+=1
xpoints = sumbux
ypoints = sumbuy


plt.plot(xpoints, ypoints, 'o')
plt.show()
```

Sekian pekerjaan saya, disini mengapa saya menggunakan jumlah generasi yang sangat besar yaitu 500, karena sepemahaman saya dalam 1 generasi hanya menghasilkan 2 child. Jadi pergerakan nilai fitness sangat lama.