

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 = 30

sumbuy=[]

sumbuy2 = []
```

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 fungsi 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 individu memiliki karakteristiknya masing masing, setiap individu yang di generate akan memiliki 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 yaitu 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 individu 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. Saya memanggil pula sumbuy untuk memasukkan nilai fitness generasi 0 dari setiap individu.

```

#membuat persentase probabilitas
sum = 0
for i in populasi :
    sum += i.fitness
    sumbuy.append(i.fitness)

```

```

avg = sum/jml_individu
sumbuy2.append(avg)

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

Perulangan `for _ in range(int(jml_individu/2))`: dimaksudkan kita akan membuat 50/2 pasangan parent. Jadi setelah terbentuk Parfent 1 dan 2 maka akan dimasukkan ke pareent_pair. Kemudian parent pair akan diulang 25 kali untuk membentuk 25 pasangan yang akhirnya setiap pasangan akan di masukkan ke array semua_parent.

```
#Tahapan Cycle Generasi
igen = 1
while igen <= jumlah_generasi:
    # Tahap Seleksi menggunakan Probabilitas Kumulatif
    #random angka dari 0 - 1

    semua_parent = []

    for _ in range(int(jml_individu/2)):
        parents_pair = []

        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

    parents_pair.append(Parent1)
    parents_pair.append(Parent2)
    semua_parent.append(parents_pair)

```

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

```

masking_col = []

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

```

10. Langkah berikutnya, saya membentuk array childtotal kosong. Lalu melakukan perulangan untuk anggota array semua_parent yang diberi nama pairs. Pairs merupakan juga array karena untuk setiap index di

semua parent mengandung 1 pasang parent yaitu parent 1 dan 2. Maka dari itu pairs mengandung 2 index yaitu index 0 untuk Parent 1 dan index 1 untuk Parent 2. Selanjutnya kita coba pisahkan 2 parent tersebut.

Kemudian 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.

```
childtotal = []

for pairs in semua_parent:

    ParentOne = []

    ParentTwo = []

    ParentOne = pairs[0]

    ParentTwo = pairs[1]

    print(f"\n Parent 0 = {ParentOne}")
    print(f"\n Parent 1 = {ParentTwo}\n")

    child1 = []
    child2 = []

    i = 0

    while i < len(masking_col) and i < len(ParentOne) and i <
len(ParentTwo):

        if masking_col[i]==1 :

            child1.append(ParentTwo[i])
            child2.append(ParentOne[i])

        elif masking_col[i]==0 :

            child1.append(ParentOne[i])
            child2.append(ParentTwo[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. Lalu saya masukkan child 1 dan 2 ke dalam array childtotal sebagai hasil akhir child.

```
#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]

childtotal.append(child1termutasi)

childtotal.append(child2termutasi)
```

14. Selanjutnya adalah saya melakukan perulangan terhadap childtotal, dimana childtotal berisi 50 keturunan dari semua parents digenerasi ini. Kemudian adalah melakukan substitusi child ke dalam populasi dengan menggantikan individu yang memiliki nilai fitness paling kecil.

Namun, sebelum itu saya melakukan pengecekan terhadap nilai decimal dari child, 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 dengan 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.

```
for c in childtotal:

    childx1 = c[0:int(jml_gen/2)]
    childx2 = c[int(jml_gen/2):int(jml_gen)]
    decimalchildx1 = bintodecimal(childx1, jml_gen)
    decimalchildx2 = bintodecimal(childx2, jml_gen)
    fitnesschild = fitness_function(decimalchildx1, decimalchildx2)

    #Perhitungan dan Penggantian dari Child 1
    if decimalchildx1 >=3 and decimalchildx1<=10 and decimalchildx2
>=3 and decimalchildx2<=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 substitusi 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 nilai decimal akan digantikan oleh child baru.

```
for q in populasi :  
  
    if q.id == minimID:  
  
        q.bit = c  
  
        q.fitness = fitnesschild  
  
        q.decimalx1 = decimalchildx1  
  
        q.decimalx2 = decimalchildx2  
  
        break  
  
    else:  
  
        continue
```

16. Langkah selanjutnya, saya menghitung ulang untuk probabilitas dan probabilitas kumulatif dari setiap individu. Saya juga mengambil nilai fitness untuk setiap individu dari setiap generasi dengan array sumbuy

```
sum = 0  
  
for i in populasi :  
  
    sum += i.fitness  
  
    sumbuy.append(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 y2.

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

avg = sum/jml_individu
sumbuy2.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=0
sumbux=[]
while i <= jumlah_generasi:

    j = 0

    while j < jml_individu:

        sumbux.append(i)

        j+=1

    i+=1

xpoints = sumbux
```

```

ypoints = sumbuy
i=0
sumbux2=[]
while i <= jumlah_generasi:
    sumbux2.append(i)
    i+=1
plt.plot(xpoints, ypoints, 'o', markersize=2, label='Semua Fitness')
plt.plot(sumbux2, sumbuy2, 'o', markersize=3, color="red", linestyle="-",
label='Rerata Fitness')
plt.xlabel('Generasi')
plt.ylabel('Fitness')
plt.title('Perkembangan Fitness pada Setiap Generasi')
plt.legend()
plt.show()

```

Sekian pekerjaan saya, disini berhasil melakukan konvergensi sekitar generasi ke 15.

