# EPC3

**Alunos1:** Luiz Eduardo [971]
**Alunos2:** Vitor R. Di Toro [983]

## 2-

### A)

```
def initPopulation(self, num_individuals):

        num_individuals

        population = []
        #Cria todos os indiviudos e insere na populacao inicial
        for i in range(num_individuals):
            individual = np.random.binomial(1,0.5,self.individual_size)
            #print(individual)
            population.append(individual.tolist())
        return population
```

### B)

```
def fitness(self,population):

        print(population)
        fitnessPop=[]
        #calculate the fitness for each individual of population
        for individual in population:
            fitnessPop.append(self.getFitness(individual))

        #print(fitnessPop)
        return fitnessPop
```

## C)

```
def __selection(self):

        sorted_population = sorted(self.population,key=self.problem.getFitness,
reverse=True)
        pop_fit = self.problem.fitness(sorted_population )

        prob_fit = []
        for individual in pop_fit:
            prob_fit .append(1.0*individual/np.sum(pop_fit))

        #Get the cumulative sum of the probabilities.
        cumSumP = np.cumsum(prob_fit )
        #Get our random numbers - one for each column.
        randomNumber = np.random.rand()
        #Get the values from A.
        #If the random number is less than the cumulative probability then
        #that's the number to use from A.
        for i, total in enumerate(cumSumP):
            if randomNumber < total:
                break
        self.population = sorted_population
        selected = self.population[i]
        #Display it. Uncomment for log.
        print("Selected individual [%d] = %s"  %(i,selected))
        return selected
```

## D)

```
def __crossover(self,individual_x,individual_y):

        n=self.problem.getIndividualSize ()

        c = int(np.random.uniform(0,n-1))
        print("crossing point: %d"  %c)

        new_individual_x =[]
        new_individual_y =[]

        # concatenate the two fathers in the C element chosen randomly
        for gene in range(c):
            new_individual_x .append(individual_x [gene])
            new_individual_y .append(individual_y [gene])
        for gene in range(c,n):
            new_individual_x .append(individual_y [gene])
            new_individual_y .append(individual_x [gene])

        return new_individual_x ,new_individual_y
```

## E)

```
def __mutation(self,individual):

    if self.__mutationTest():
        randomPosition = int(np.random.uniform(0,self.problem.getIndividualSize()-1))
        print('Mutation at position: %d' %randomPosition)
        #get a random value for changing in the individual position selected before
        randomValue =
np.random.uniform(self.problem.getMinGeneSymbol(),self.problem.getMaxGeneSymbol())

        if(randomValue <= 0.5):
            randomValue = int(randomValue)
        else:
            randomValue = int(randomValue+1)

        print('New gene value: %d' %randomValue)
        individual[randomPosition]=randomValue
return individual
```

# 3-

```
MaxGeneration = 100
```

# 4-

**Ótimo da função:** `x1 = 1` e `x2 = 1`, que gera `f(x1,x2) = 0`

**Ótimo obtido:** `x1 = 0.996078` e `x2 = 0.992157`, que gerou `f(x1,x2) = 1.54024e-05`

**População:** 100 indivíduos

**Gerações:** 100 gerações

# 5-

**Ótimo obtido:** `x1 = 0.996078` e `x2 = 0.992157`, que gerou `f(x1,x2) = 1.54024e-05`

**População:** 350 indivíduos

**Gerações:** 150 gerações