*МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ*

*НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ*

*„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”*

*НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС*

*„ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”*

***Лабораторна робота №1***

*з курсу «Штучний інтелект»*

*Тема: “Решение задачи коммивояжера с помощью генетических алгоритмов”*

*Виконала:*

*студентка III курсу*

*групи ДА-42*

*Балан Катерина*

*Київ*

*2017*

**Задание на лабораторную работу**

1.Решить задачу коммивояжера с помошью классического генетического алгоритма.

2. Исследовать сходимость алгоритма в зависимости от типа кроссовера (одноточечного или n-точечного).

**Виконання роботи**

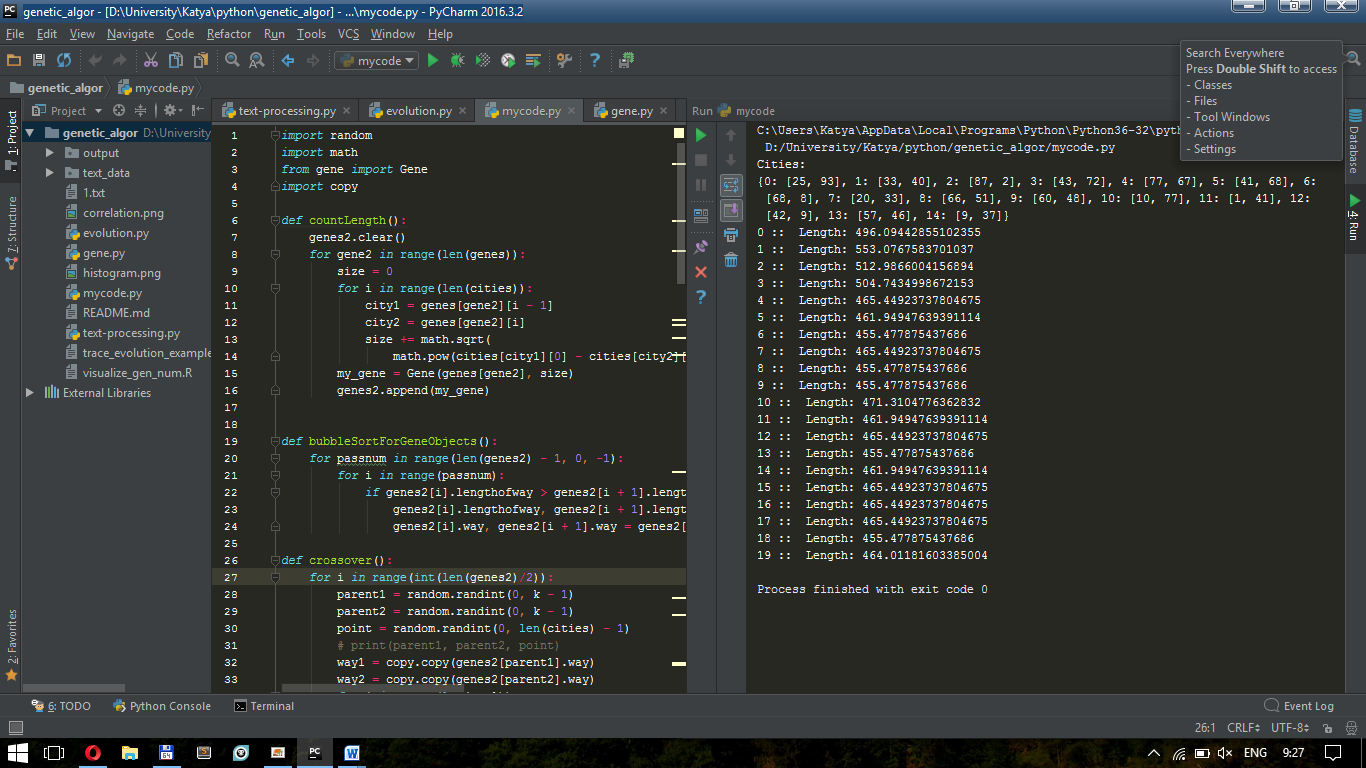
Створено клас, для зручності зберігання данних:

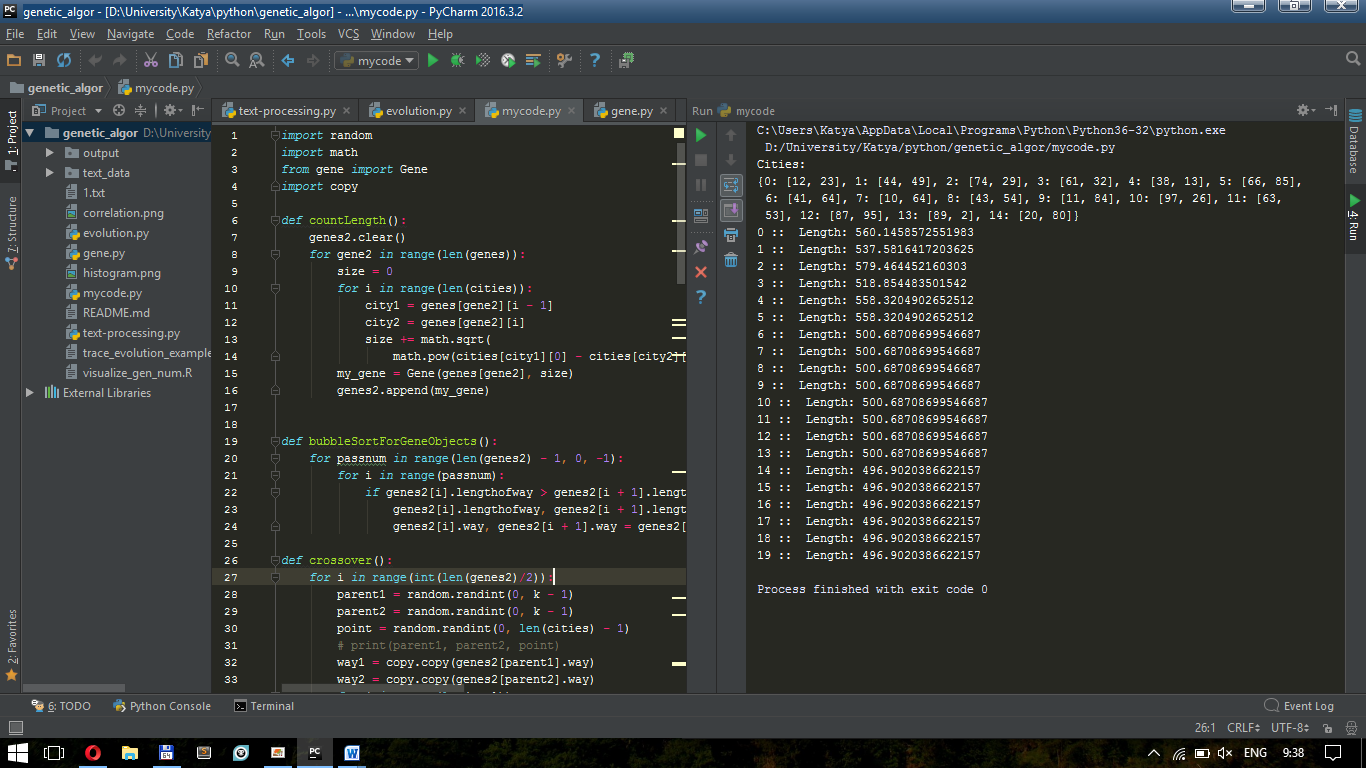
*class* Gene:  
 *def* \_\_init\_\_(self, *way*, *lengthofway*):  
 self.way = *way* self.lengthofway = *lengthofway  
  
 def* display\_gene(self):  
 *return* " Length: " + str(self.lengthofway)

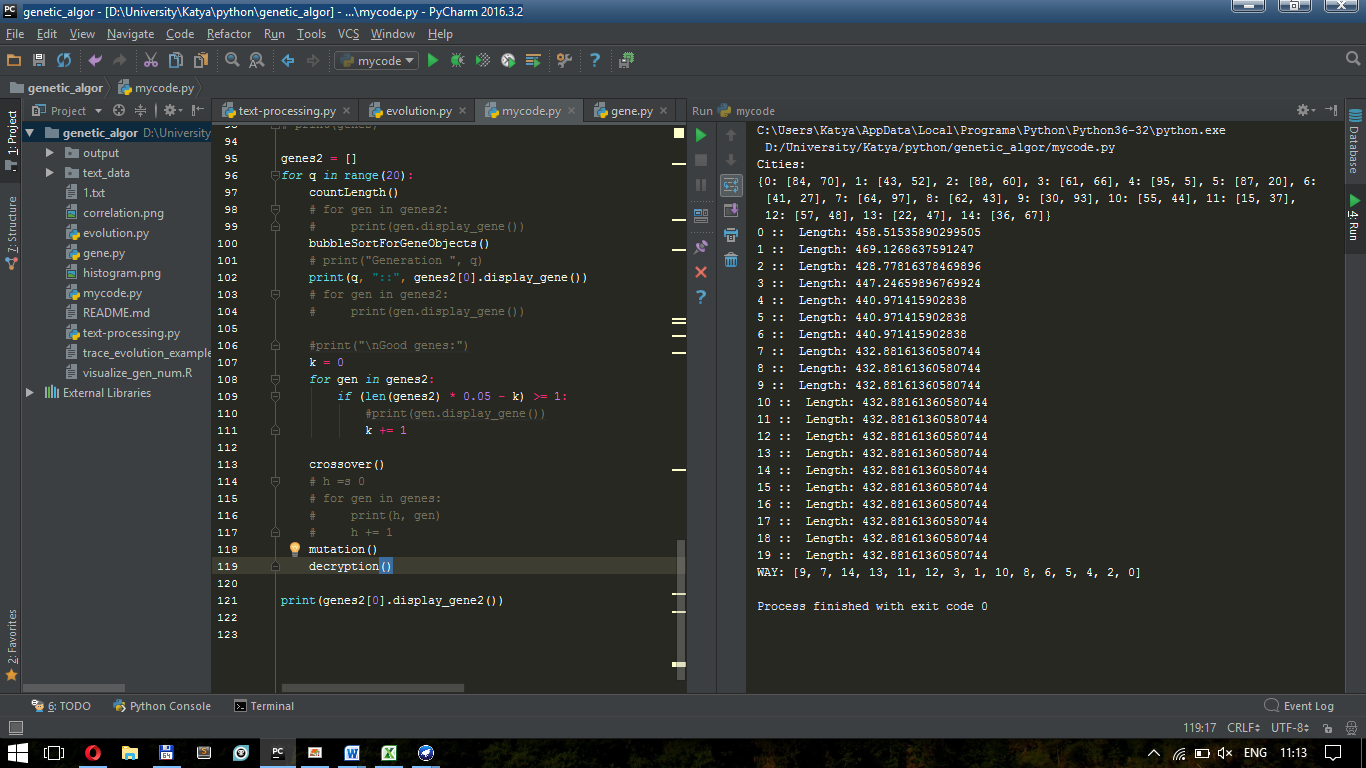
І написано програму, що реалізує нашу поставлену задачу:

*import* random  
*import* math  
*from* gene *import* Gene  
*import* copy  
  
*def* countLength():  
 genes2.clear()  
 *for* gene2 *in* range(len(genes)):  
 size = 0  
 *for* i *in* range(len(cities)):  
 city1 = genes[gene2][i - 1]  
 city2 = genes[gene2][i]  
 size += math.sqrt(  
 math.pow(cities[city1][0] - cities[city2][0], 2) + math.pow(cities[city1][1] - cities[city2][1], 2))  
 my\_gene = Gene(genes[gene2], size)  
 genes2.append(my\_gene)  
  
  
*def* bubbleSortForGeneObjects():  
 *for* passnum *in* range(len(genes2) - 1, 0, -1):  
 *for* i *in* range(passnum):  
 *if* genes2[i].lengthofway > genes2[i + 1].lengthofway:  
 genes2[i].lengthofway, genes2[i + 1].lengthofway = genes2[i + 1].lengthofway, genes2[i].lengthofway  
 genes2[i].way, genes2[i + 1].way = genes2[i + 1].way, genes2[i].way  
  
*def* crossover():  
 *for* i *in* range(int(len(genes2)/2)):  
 parent1 = random.randint(0, k - 1)  
 parent2 = random.randint(0, k - 1)  
 point = random.randint(0, len(cities) - 1)  
 # print(parent1, parent2, point)  
 way1 = copy.copy(genes2[parent1].way)  
 way2 = copy.copy(genes2[parent2].way)  
 *for* j *in* range(len(way1)):  
 *if* (j - point) >= 0:  
 way1[j], way2[j] = way2[j], way1[j]  
 genes[i \* 2] = copy.copy(way1)  
 genes[i \* 2 + 1] = copy.copy(way2)  
 # print(way1, genes[i \* 2])  
 # print(way2, genes[i \* 2 + 1])  
  
*def* mutation():  
 *for* gen *in* genes:  
 position = random.randint(0, len(cities) - 1)  
 # print("Position: ", position)  
 mutationGene = random.randint(0, len(cities) - 1)  
 gen[position] = mutationGene  
 # print(gen)  
  
*def* decryption():  
 *for* i *in* range(len(genes)):  
 temp = [a *for* a *in* range(len(cities))]  
 temp\_way = copy.copy(genes[i])  
 # print(temp\_way)  
 new\_temp\_way = []  
 *for* j *in* range(len(temp\_way)):  
 # print(j)  
 # print(len(temp), temp\_way[j])  
 *if* (len(temp) - temp\_way[j]) > 0:  
 dddd = temp.pop(temp\_way[j])  
 # print("dddd = ", dddd)  
 new\_temp\_way.append(dddd)  
 # print(temp, temp\_way, new\_temp\_way)  
 *else*:  
 ddddd = temp.pop()  
 # print("dddd = ", ddddd)  
 new\_temp\_way.append(ddddd)  
 # print("ololol")  
 # print(temp, temp\_way, new\_temp\_way)  
 genes[i] = copy.copy(new\_temp\_way)  
 new\_temp\_way.clear()  
  
  
  
  
  
cities = {a: *None for* a *in* range(15)}  
  
*for* city *in* cities:  
 a = {city: [random.randrange(0, 100), random.randrange(0, 100)]}  
 cities.update(a)  
print("Cities: ")  
print(cities)  
  
forRange = [a *for* a *in* range(len(cities))]  
  
genes = []  
*for* gene *in* range(1000): # 30  
 random.shuffle(forRange)  
 genes.append(forRange.copy())  
  
# print("Genes: ")  
# print(genes)  
  
genes2 = []  
*for* q *in* range(20):  
 countLength()  
 # for gen in genes2:  
 # print(gen.display\_gene())  
 bubbleSortForGeneObjects()  
 # print("Generation ", q)  
 print(q, "::", genes2[0].display\_gene())  
 # for gen in genes2:  
 # print(gen.display\_gene())  
  
 #print("\nGood genes:")  
 k = 0  
 *for* gen *in* genes2:  
 *if* (len(genes2) \* 0.05 - k) >= 1:  
 #print(gen.display\_gene())  
 k += 1  
  
 crossover()  
 # h =s 0  
 # for gen in genes:  
 # print(h, gen)  
 # h += 1  
 mutation()  
 decryption()

Результати роботи програми:







**Висновок:** Данна лабораторна робота дала нам явне розуміння генетичного алгоритму і яким способом його можна використовувати для вирішення поставлених задач. Ми розібрали алгоритм достатньо доре, щоб вирішити як саме його можна вдосконалити для вирішення наприклад нашої задачі. Для початку хотілось би використовувати в новому поколінні батьків, а також зменшити вірогідність для мутації.