**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №2

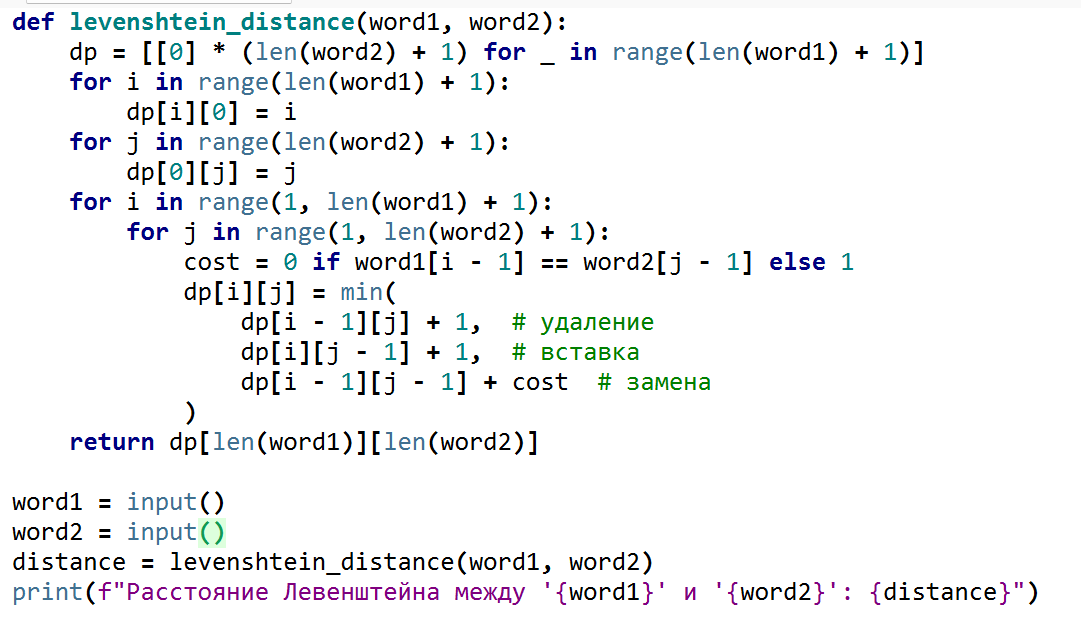
«Нахождение длины Левенштейна»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Беляев Иван |  | Нардид А.Н. |
| Подпись и дата: |  | Подпись и дата: |

**Задание:**

Даны два слова. Вычислить длину Левенштейна этих двух слов.

**Программа на языке Python:**



def levenshtein\_distance(word1, word2):

dp = [[0] \* (len(word2) + 1) for \_ in range(len(word1) + 1)]

for i in range(len(word1) + 1):

dp[i][0] = i

for j in range(len(word2) + 1):

dp[0][j] = j

for i in range(1, len(word1) + 1):

for j in range(1, len(word2) + 1):

cost = 0 if word1[i - 1] == word2[j - 1] else 1

dp[i][j] = min(

dp[i - 1][j] + 1, # удаление

dp[i][j - 1] + 1, # вставка

dp[i - 1][j - 1] + cost # замена

)

return dp[len(word1)][len(word2)]

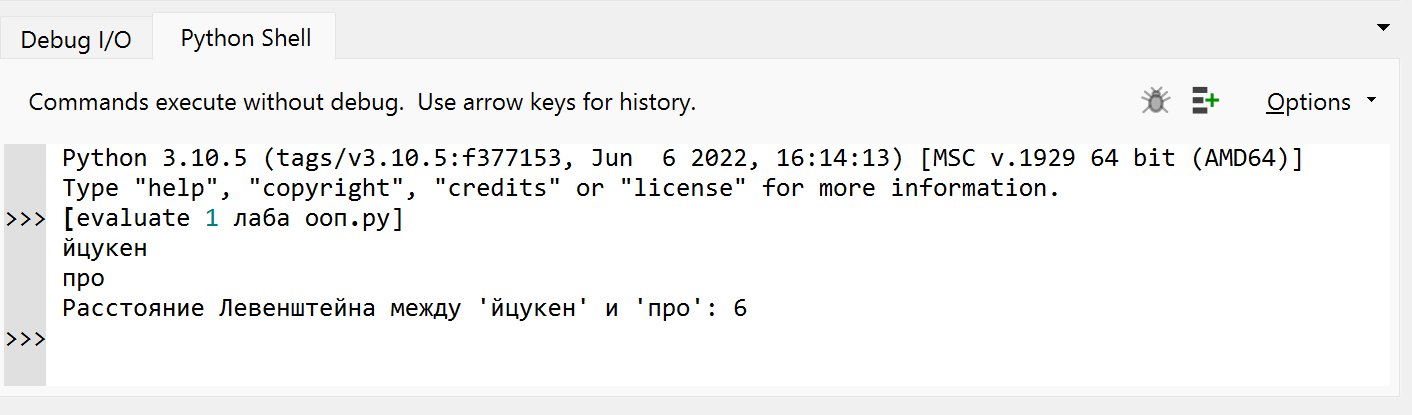
word1 = input()

word2 = input()

distance = levenshtein\_distance(word1, word2)

print(f"Расстояние Левенштейна между '{word1}' и '{word2}': {distance}")

**Результат работы программы:**



Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

**Программа на языке Pascal:**

program LevenshteinDistance;

function Min(a, b, c: Integer): Integer;

begin

if a < b then

if a < c then

Min := a

else

Min := c

else

if b < c then

Min := b

else

Min := c;

end;

function LevenshteinDistance(word1, word2: string): Integer;

var

len1, len2, i, j, cost: Integer;

dp: array of array of Integer;

begin

len1 := Length(word1);

len2 := Length(word2);

SetLength(dp, len1 + 1, len2 + 1);

for i := 0 to len1 do

dp[i][0] := i;

for j := 0 to len2 do

dp[0][j] := j;

for i := 1 to len1 do

begin

for j := 1 to len2 do

begin

if word1[i] = word2[j] then

cost := 0

else

cost := 1;

dp[i][j] := Min(

dp[i - 1][j] + 1, // удаление

dp[i][j - 1] + 1, // вставка

dp[i - 1][j - 1] + cost // замена

);

end;

end;

// Результат в правом нижнем углу матрицы

LevenshteinDistance := dp[len1][len2];

end;

var

word1, word2: string;

distance: Integer;

begin

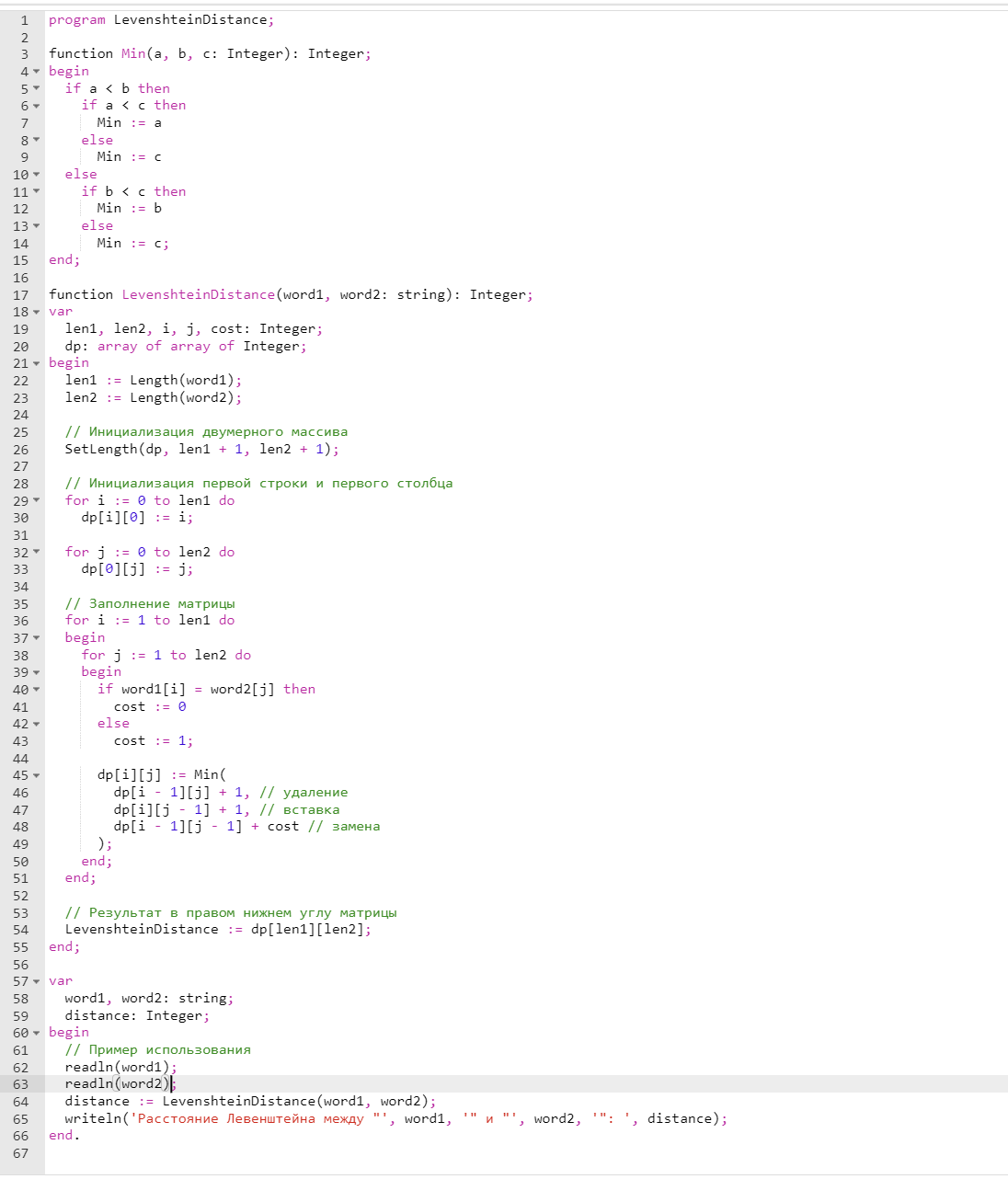
readln(word1);

readln(word2);

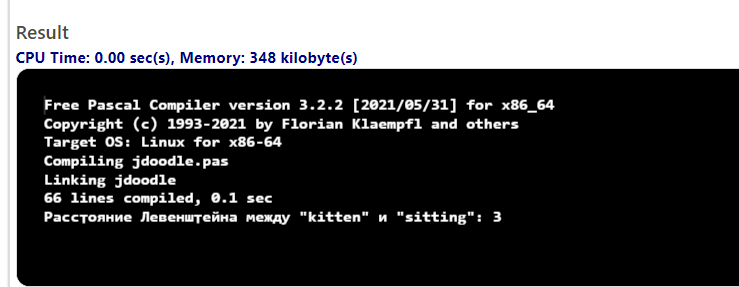
distance := LevenshteinDistance(word1, word2);

writeln('Расстояние Левенштейна между "', word1, '" и "', word2, '": ', distance);

end.



**Результат работы программы:**



**Программа на языке Kotlin:**

val scan = java.util.Scanner(System.`in`)

fun min(a: Int, b: Int, c: Int) = if (a < b) if (a < c) a else c else if (b < c) b else c

fun levenshteinDistance(word1: String, word2: String): Int {

val len1 = word1.length

val len2 = word2.length

// Инициализация двумерного массива

val dp = Array(len1 + 1) { IntArray(len2 + 1) }

for (i in 0..len1) dp[i][0] = i

for (j in 0..len2) dp[0][j] = j

for (i in 1..len1) {

for (j in 1..len2) {

val cost = if (word1[i - 1] == word2[j - 1]) 0 else 1

dp[i][j] = min(

dp[i - 1][j] + 1, // удаление

dp[i][j - 1] + 1, // вставка

dp[i - 1][j - 1] + cost // замена

)

}

}

return dp[len1][len2]

}

fun main() {

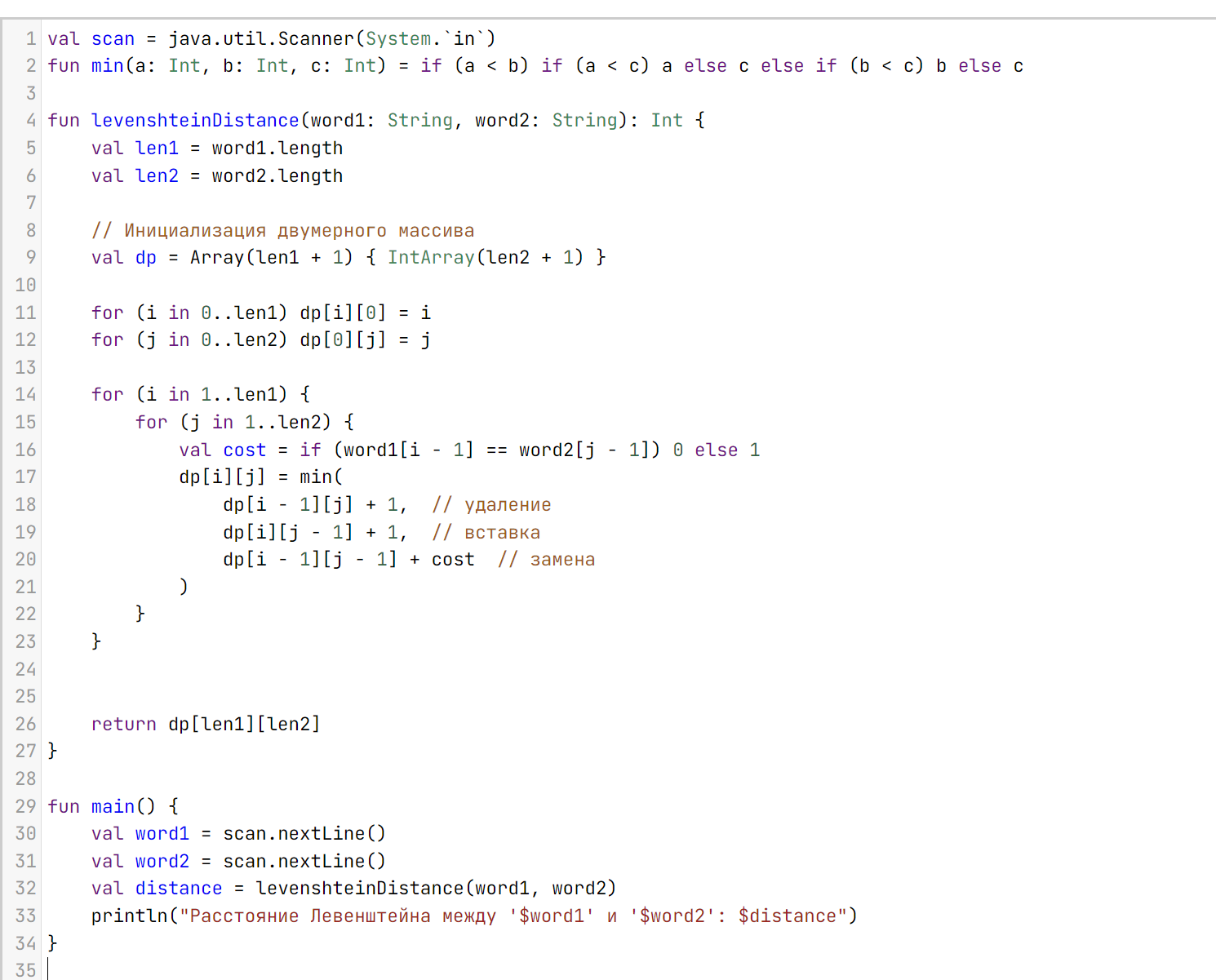
val word1 = scan.nextLine()

val word2 = scan.nextLine()

val distance = levenshteinDistance(word1, word2)

println("Расстояние Левенштейна между '$word1' и '$word2': $distance")

}



**Результат работы программы:**

