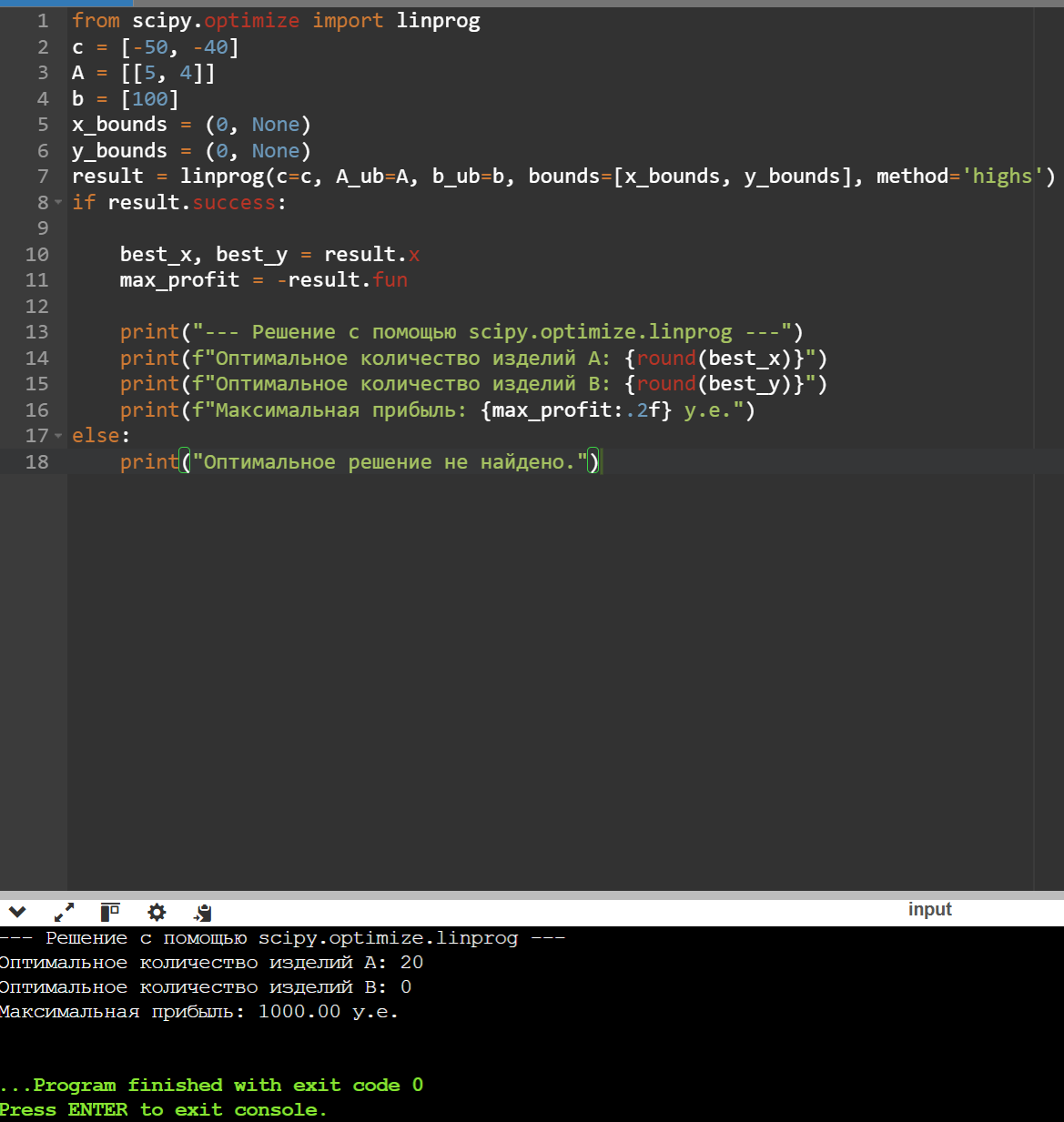
**Лабораторная работа 2**

**Тема:** Решение простейших однокритериальных задач  
**Дисциплина:** МДК 02.03 «Математическое моделирование»  
**Учебный год:** 2025–2026

**Цель работы**

1. Научиться формулировать простейшие однокритериальные задачи.
2. Освоить методы поиска оптимального решения с использованием Python.
3. Научиться анализировать результаты моделирования и выбирать наилучшее решение по одному критерию.

**Задача 1: Максимизация прибыли (Линейное программирование)**

****

from scipy.optimize import linprog

c = [-50, -40]

A = [[5, 4]]

b = [100]

x\_bounds = (0, None)

y\_bounds = (0, None)

result = linprog(c=c, A\_ub=A, b\_ub=b, bounds=[x\_bounds, y\_bounds], method='highs')

if result.success:

best\_x, best\_y = result.x

max\_profit = -result.fun

print("--- Решение с помощью scipy.optimize.linprog ---")

print(f"Оптимальное количество изделий A: {round(best\_x)}")

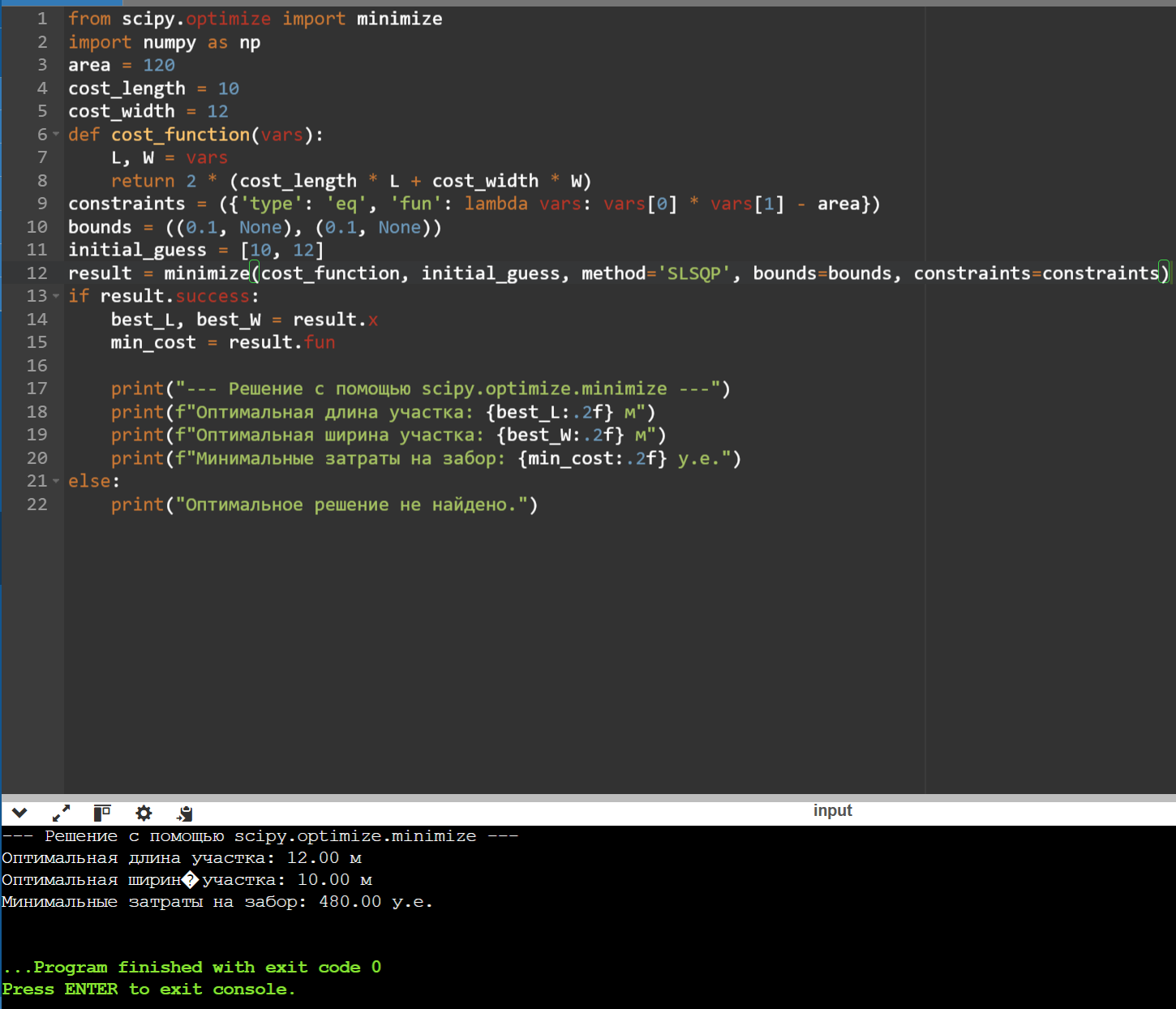
print(f"Оптимальное количество изделий B: {round(best\_y)}")

print(f"Максимальная прибыль: {max\_profit:.2f} у.е.")

else:

print("Оптимальное решение не найдено.")

**Задача 2: Минимизация затрат (Нелинейное программирование)**



from scipy.optimize import minimize

import numpy as np

area = 120

cost\_length = 10

cost\_width = 12

def cost\_function(vars):

L, W = vars

return 2 \* (cost\_length \* L + cost\_width \* W)

constraints = ({'type': 'eq', 'fun': lambda vars: vars[0] \* vars[1] - area})

bounds = ((0.1, None), (0.1, None))

initial\_guess = [10, 12]

result = minimize(cost\_function, initial\_guess, method='SLSQP', bounds=bounds, constraints=constraints)

if result.success:

best\_L, best\_W = result.x

min\_cost = result.fun

print("--- Решение с помощью scipy.optimize.minimize ---")

print(f"Оптимальная длина участка: {best\_L:.2f} м")

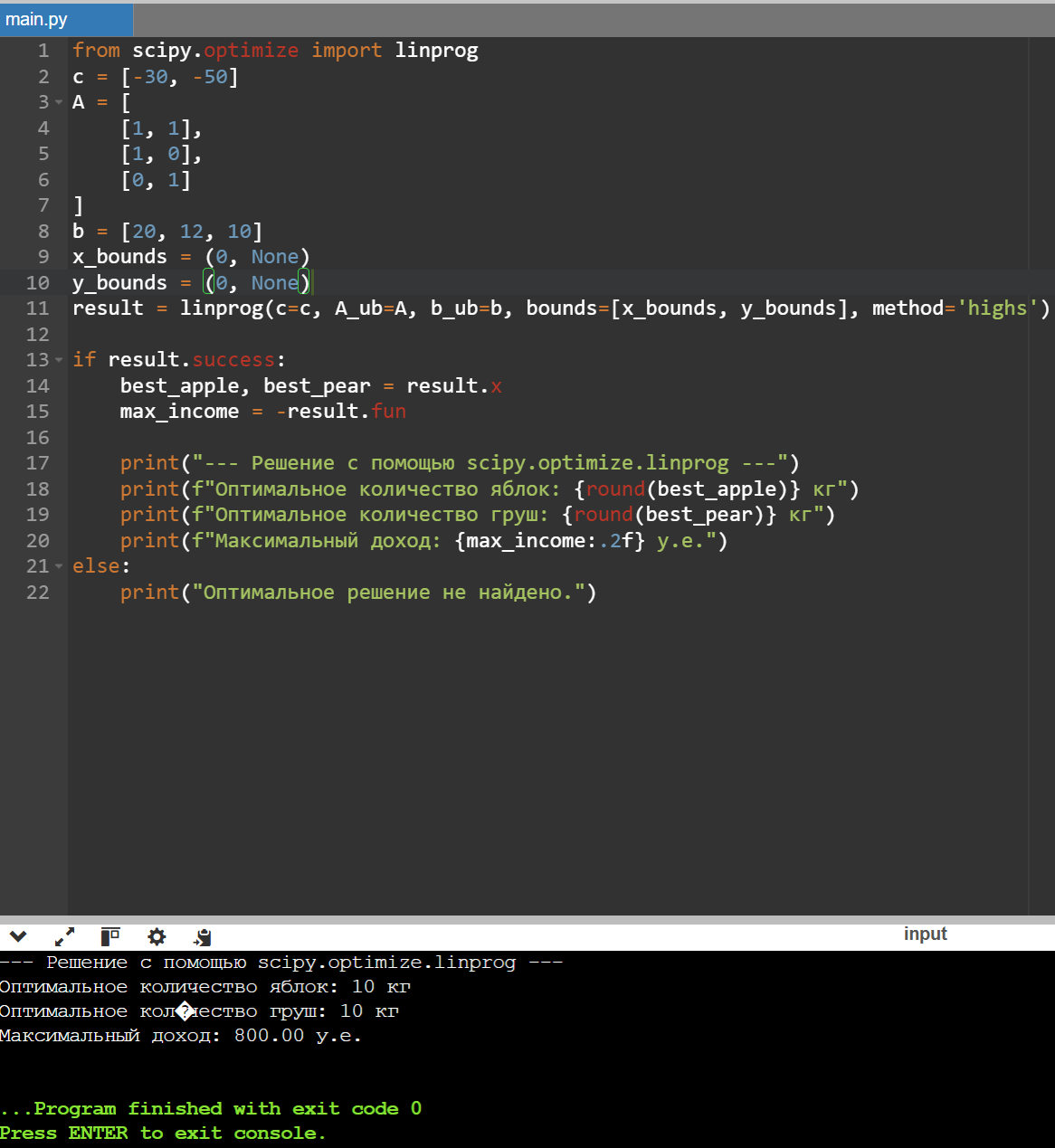
print(f"Оптимальная ширина участка: {best\_W:.2f} м")

print(f"Минимальные затраты на забор: {min\_cost:.2f} у.е.")

else:

print("Оптимальное решение не найдено.")

**Задача 3: Максимизация дохода от фруктов (Линейное программирование)**

****

from scipy.optimize import linprog

c = [-30, -50]

A = [

[1, 1],

[1, 0],

[0, 1]

]

b = [20, 12, 10]

x\_bounds = (0, None)

y\_bounds = (0, None)

result = linprog(c=c, A\_ub=A, b\_ub=b, bounds=[x\_bounds, y\_bounds], method='highs')

if result.success:

best\_apple, best\_pear = result.x

max\_income = -result.fun

print("--- Решение с помощью scipy.optimize.linprog ---")

print(f"Оптимальное количество яблок: {round(best\_apple)} кг")

print(f"Оптимальное количество груш: {round(best\_pear)} кг")

print(f"Максимальный доход: {max\_income:.2f} у.е.")

else:

print("Оптимальное решение не найдено.")

**Вывод по работе**

В ходе выполнения лабораторной работы я научился формулировать простейшие однокритериальные задачи оптимизации, составлять для них математические модели и решать их с использованием языка программирования Python.

**1.Метод полного перебора (brute-force):** Этот метод заключается в итеративном переборе всех возможных комбинаций управляющих переменных в заданном диапазоне и выборе той, которая дает наилучшее значение целевой функции. Метод прост в реализации и понимании, он гарантированно находит оптимальное решение для задач с небольшим целочисленным пространством вариантов (как в задачах 1 и 3).

**2. Метод с использованием библиотеки scipy.optimize:** Этот подход использует профессиональные, математически обоснованные алгоритмы оптимизации.

* Для задач линейного программирования (задачи 1 и 3) была использована функция linprog, которая эффективно находит решение, минимизируя или максимизируя линейную функцию при линейных ограничениях.
* Для задачи нелинейного программирования (задача 2) была применена функция minimize, способная работать с нелинейными ограничениями и находить точное решение в пространстве непрерывных переменных.

**3. Сравнение методов показало, что:**

* Результаты обоих методов совпали в представленных задачах, что подтверждает корректность реализации простого перебора для данных условий.
* Метод с scipy является более универсальным, масштабируемым и эффективным. Он способен решать сложные задачи с большим числом переменных и непрерывными значениями, где полный перебор был бы невозможен.