Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №1
Вариант №1

Выполнил:
Бацанова Е. А.
Проверил
Мусаев А.А.

Санкт-Петербург,
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является ознакомление с языком программирования Python, его основными инструментами и библиотеками. Встроенные библиотеки Python значительно упрощают разработку программ и ускоряют их работу. Одной из таких библиотек является numpy, позволяющая работать с многомерными массивами и выполнять различные операции над ними.

В данной работе были решены задачи, в основном связанные с матрицами и операциями над ними. Для начала была написана простая программа «FizzBuzz» для ознакомления с основами программирования на Python. Затем без использования готовых библиотек для работы с матрицами были реализованы функции:

1) ввода и вывода матрицы (вспомогательные);
2) транспонирования матрицы;
3) умножения матриц;
4) определения ранга матрицы.

После этого была изучена библиотека numpy и с её помощью реализованы функции 2) – 4), проведен анализ достоинств и недостатков использования numpy в программировании. В конце была создана собственная функция для возведения матрицы размерности 3×3 в степень -1 и произведено сравнение ее быстродействия с функцией из numpy с помощью библиотеки timeit.

Задание 1

Задание: Написать программу «FizzBuzz» (рис. 1)

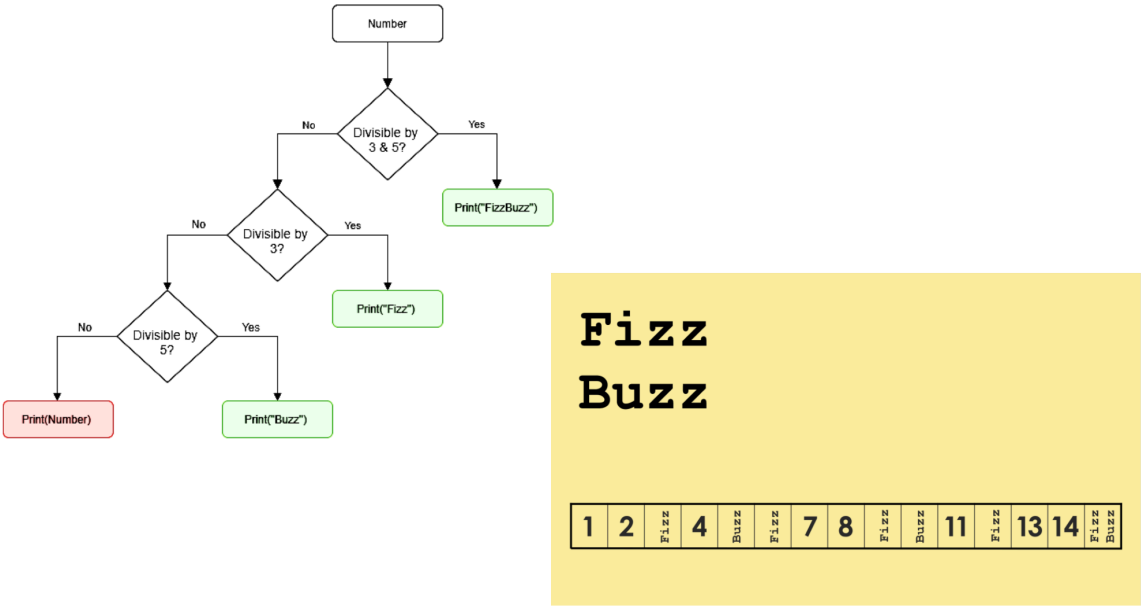

Рисунок 1 – Блок – схема программы «FizzBuzz» и ее вывод для диапазона чисел [1, 16]

Решение:

На рис. 2 приведена написанная программа «FizzBuzz». Для вывода результатов работы программы в цикле for перебираются числа от 1 до 15 включительно. Для каждого числа вызывается функция FizzBuzz. С помощью параметра *end=" "* реализуется вывод всех элементов в одну строку через пробел.

Функция FizzBuzz принимает число в качестве аргумента, а затем проверяет остаток от деления числа на 3 и 5. Если число делится и на 3, и на 5, то выводится «FizzBuzz». Если число делится только на 3, то выводится «Fizz». Если число делится только на 5, то выводится «Buzz». Если число не делится ни на 3, ни на 5, то выводится само число.

```python
def FizzBuzz (number):
    if number % 3 == 0 and number % 5 == 0:
        print("FizzBuzz", end="  ")
    else:
        if number % 3 == 0:
            print("Fizz", end="  ")
        else:
            if number % 5 == 0:
                print("Buzz", end="  ")
            else:
                print(number, end="  ")


for i in range(1, 16):
    FizzBuzz(i)
```

Рисунок 2 – Реализация программы FizzBuzz и ее вывода для диапазона чисел [1, 16]

Вывод программы представлен на рис. 3.

```
1  2  Fizz  4  Buzz  Fizz  7  8  Fizz  Buzz  11  Fizz  13  14  FizzBuzz
```

Рисунок 3 – Вывод программы «FizzBuzz»

Задание 2

Задание: Создать программу на языке Python, которая будет содержать следующие функции:

1) Транспонирование матрицы;
2) Умножение матриц;
3) Определение ранга матрицы.

Для реализации функций использовать любой способ ввода данных. Обосновать выбранный метод. Не допускается использование готовых библиотек для работы с матрицами (numpy).

Решение:

Для удобной работы с матрицами реализуем функции для их ввода (*input_matrix*) и вывода (*print_matrix*).

Функция *input_matrix()* запрашивает у пользователя количество строк и столбцов для создания матрицы заданного размера. Для создания матрицы функция использует цикл while, который продолжается до тех пор, пока не будут введены все строки матрицы. Каждая строка считывается из консоли в виде списка целых чисел, разделенных пробелами. Если введенное количество элементов не соответствует заданному количеству столбцов, выводится сообщение об ошибке. После завершения ввода всех строк функция возвращает полученную матрицу.

Метод ввода данных с помощью отдельной функции *input_matrix* удобен в первую очередь потому, что он позволяет упразднить дублирование кода при вводе нескольких матриц и подходит для ввода матриц любого размера, так как пользователь сам задает количество строк и столбцов матрицы. Также данная функция удобна тем, что контролирует количество элементов в введенной строке – она проверяет соответствие числа столбцов матрицы числу элементов в строке и, в случае их несовпадения, выводит сообщение об ошибке. Всё это позволяет контролировать ввод данных, фильтровать ошибки ввода, повышает удобство кода.

Функция *print_matrix(matrix)* отображает содержимое матрицы в удобном для восприятия формате. Для каждого элемента матрицы определяется максимальная длина его текстового представления, чтобы корректно выровнять элементы столбцов. Затем построчно выводится содержимое матрицы. Выравнивание столбцов реализуется при помощи метода *ljust()*. После отображения матрицы выводится пустая строка для удобства чтения.

```python
def input_matrix():
    rows = int(input("Enter number of rows for matrix: "))
    columns = int(input("Enter number of columns for matrix: "))
    matrix, cnt = [], 1
    while cnt <= rows:
        row = list(map(int, input(f"Enter {columns} elements for row {cnt}: ").split()))
        if len(row) != columns:
            print(f"The string must contain {columns} elements!")
        else:
            matrix.append(row)
            cnt += 1
    return matrix


def print_matrix(matrix):
    n = 0
    for row in matrix:
        for item in row:
            n = max(n, len(str(item)))
    for row in matrix:
        for i in row:
            print(str(i).ljust(n), end=' ')
        print()
    print()
```

Рисунок 4 – Реализация ввода и вывода матрицы

После этого были реализованы основные функции задания. Функция *transpose_matrix(matrix)* (рис. 5) принимает матрицу в качестве аргумента и возвращает транспонированную матрицу. Транспонированная матрица получается путем замены строк исходной матрицы на ее столбцы. Внешний цикл реализует проход по столбцам исходной матрицы *(range(len(matrix[0]))*, где *len(matrix[0])* – количество столбцов в исходной матрице. Внутренний цикл реализует проход по строкам исходной матрицы *(range(len(matrix)))*, где *len(matrix)* – количество строк в исходной матрице. На каждой итерации создается новый список, содержащий элементы *j*-го столбца и *i*-й строки исходной матрицы. После прохода всех элементов, возвращается полученная транспонированная матрица.

```python
def transpose_matrix(matrix):
    transposed_matrix = [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]
    return transposed_matrix
```

Рисунок 5 – Реализация транспонирования матрицы

Для умножения матриц была реализована функция *multiplying_matrices(matrix1, matrix2)*. Если количество столбцов первой матрицы равно количеству строк второй матрицы (то есть матрицы можно умножить), то функция создает новую матрицу *result* количество столбцов которой равно количеству столбцов второй матрицы, а количество

строк – количеству строк первой. Затем происходит циклическое перемножение элементов матриц *matrix1* и *matrix2* с использованием третьего цикла, который проходит по столбцам первой матрицы и строкам второй матрицы. В результате каждого умножения суммируются все произведения элементов и записываются в соответствующий элемент матрицы *result*. В конце функция возвращает полученную матрицу *result* или выводит сообщение об ошибке, если матрицы нельзя умножить.

```
def multiplying_matrices(matrix1, matrix2):
    if len(matrix1[0]) != len(matrix2):
        print("Specified matrices cannot be multiplied")
        return None

    result = [[0] * len(matrix2[0]) for _ in range(len(matrix1))]
    for i in range(len(matrix1)):
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):
                result[i][j] += matrix1[i][k] * matrix2[k][j]
    return result
```

Рисунок 6 – Реализация умножения матриц

В соответствии с определением: ранг матрицы равен количеству ненулевых строк после приведения матрицы к ступенчатому виду. Для определения ранга матрицы была реализована функция *rank_of_matrix(matrix)*. В качестве аргумента она принимает матрицу после чего приводит её к ступенчатому виду, а затем считает количество ненулевых строк. Для того, чтобы сохранить входную матрицу в первоначальном виде создается ее копия *mas*. Задаются переменные *rows* (количество строк) и *columns* (количество столбцов) матрицы, *lead* (вспомогательная переменная, с помощью которой будет реализовываться поиск подмассивов, содержащих наименьшее число нулей на первых позициях, и передвижение таких подмассивов наверх исходного массива *mas*) и *flag* (для проверки того, что мы еще не дошли до последнего элемента последнего подмассива). Приведение матрицы к ступенчатому виду производится методом Гаусса – с помощью последовательного вычитания ведущей строки из остальных строк.

```python
def rank_of_matrix(matrix):
    mas = matrix.copy()
    rows, columns = len(mas), len(mas[0])
    lead, flag = 0, True
    for r in range(rows):
        if lead < columns:
            i = r
            while mas[i][lead] == 0:
                i += 1
                if i == rows:
                    i = r
                    lead += 1
                    if columns == lead:
                        flag = False
                        break
            if flag:
                mas[i], mas[r] = mas[r], mas[i]
                k = mas[r][lead]
                mas[r] = [item / k for item in mas[r]]
                for i in range(rows):
                    if i != r:
                        k = mas[i][lead]
                        mas[i] = [item - k * item_r for item_r, item in zip(mas[r], mas[i])]
                lead += 1
            else:
                break
        else:
            break
    rank = rows
    for i in mas:
        if i == [0] * columns:
            rank -= 1
    return rank
```

Рисунок 7 – Реализация определения ранга матрицы

Протестируем работу программы – для этого введем 2 матрицы и применим к ним написанные ранее функции. Видим, что вывод программы корректен.

```
Enter number of rows for matrix: 2
Enter number of columns for matrix: 2
Enter 2 elements for row 1: 12 23
Enter 2 elements for row 2: 0 0
Enter number of rows for matrix: 2
Enter number of columns for matrix: 2
Enter 2 elements for row 1: 1 0
Enter 2 elements for row 2: 0 1
Matrix 1:
12 23
0  0

Matrix 2:
1 0
0 1

Transposed Matrix 1:
12 0
23 0

Transposed Matrix 2:
1 0
0 1

Multiplied Matrix:
12 23
0  0

Rank of Matrix 1:
1
Rank of Matrix 2:
2
```

Рисунок 8 – Вывод программы, реализующей транспонирование матрицы, поиск ранга и перемножение матриц

Задание 3

Задание:

1) Изучите библиотеку numpy;
2) Выполните задание 2 с использованием данной библиотеки;
3) Проанализируйте достоинства и недостатки использования numpy.

Решение:

Методы для ввода и вывода матрицы наследуются из прошлого задания (рис. 4).

Для того, чтобы начать работу с numpy необходимо импортировать библиотеку при помощи команды *import numpy as np*. При этом библиотека numpy будет доступна под псевдонимом «np», что упрощает обращение к ее методам и классам в коде.

Использованные функции из библиотеки numpy:

1) *np.transpose* – транспонирование матрицы;
2) *np.dot* – перемножение матриц;
3) *np.linalg.matrix_rank* – перемножение матриц.

```
print("Transposed Matrix 1:")
print_matrix(np.transpose(matrix1))

print("Transposed Matrix 2:")
print_matrix(np.transpose(matrix2))

print("Multiplied Matrix:")
print_matrix(np.dot(matrix1, matrix2))

print("Rank of Matrix 1:")
print(np.linalg.matrix_rank(matrix1))

print("Rank of Matrix 2:")
print(np.linalg.matrix_rank(matrix2))
```

Рисунок 8 – Программа, реализующая транспонирование матрицы, поиск ранга и перемножение матриц методами numpy

Протестируем работу программы – для этого введем 2 матрицы, которые вводили ранее при тестировании программы из задания 2. Вывод программы оказался такой же (аналогичен рис. 8).

Проанализируем достоинства и недостатки работы с numpy.

К достоинствам numpy можно отнести:

1) Высокую производительность;
2) Простота и удобство использования
3) Широкий набор функций для работы с матрицами;

К недостаткам использования numpy можно отнести:

1) Необходимость изучения дополнительных функций и методов
2) Ограничения на размер матрицы
3) Не всегда удобный и интуитивный синтаксис
4) Меньшая глубина понимание того, как работает программа

Тем не менее, в целом использование numpy имеет больше преимуществ, чем недостатков. Numpy является эффективным и легким инструментом для работы с матрицами.

Задание 4

Задание:

1) Создайте свою функцию для возведения матрицы A размерности 3×3 в степень -1, где входными переменными функции будут элементы матрицы A, вводимые вручную;

2) С помощью библиотеки timeit сравните быстродействие вашей функции с ее аналогом из библиотеки numpy

Решение:

Для начала реализуем собственную функцию возведения матрицы в степень -1 *inverse_matrix(matrix)* (рис. 11). Данная функция будет работать для матрицы любой размерности, в том числе для матриц 3×3.

Для того, чтобы реализовать вычисление обратной матрицы, также создадим вспомогательную функцию *matrix_det(matrix, parity)* (рис. 10) для вычисления ее определителя. В качестве аргументов в функцию передается сама матрица *matrix* и четность *parity*, используемая для фиксирования знака, с которым слагаемое будет суммироваться с общим значением определителя.

Внутри функции *inverse_matrix(matrix)* сразу проверяем возможно ли вычисление обратной матрицы. Если матрица не квадратная, выводится сообщение об ошибке *"It is impossible to calculate the determinant of a given matrix"*. Если определитель матрицы равен нулю выводится сообщение об ошибке *"Inverse matrix not defined"*. В обоих случаях обратную матрицу вычислить нельзя, поэтому функция возвращает *None*. Далее формируем обратную матрицу по формуле $A^{-1} = \frac{1}{\Delta_A} A_{ij}^{\ T}$, где Δ_A – определитель исходной матрицы, $A_{ij}^{\ T}$ – транспонированная матрицы алгебраических дополнений. Для этого последовательно вычисляем миноры (определители матрицы с зачеркнутой *i*-той строкой и *j*-тым столбцом) с помощью функции *matrix_det(matrix, parity)*. Если размер такой матрицы равен 1×1 или 2×2, определитель вычисляется по стандартным формулам. Если размер матрицы 3×3 или более – определитель вычисляется рекурсивно для каждого минора, умножается на соответствующий элемент исходной матрицы и прибавляется к общему определителю. Во всех случаях определитель вычисляем по первой строке.

Функции ввода и вывода матрицы также заимствуем из задания 2, однако для ввода матрицы теперь не просим ввести количество столбцов и строк, а устанавливаем фиксированное значение – 3 (рис. 9).

```
def input_matrix():
    rows = 3
    columns = 3
    matrix, cnt = [], 1
    while cnt <= rows:
        row = list(map(int, input(f"Enter {columns} elements for row {cnt}: ").split()))
        if len(row) != columns:
            print(f"The string must contain {columns} elements!")
        else:
            matrix.append(row)
            cnt += 1
    return matrix
```

Рисунок 9 – Реализация ввода матрицы 3×3

```
def matrix_det(matrix, parity):
    n = len(matrix[0])
    if n == 2:
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
    elif n == 1:
        return matrix[0][0]
    det = 0
    for i in range(n):
        modified_matrix = [row[:i] + row[i + 1:] for row in matrix[:0] + matrix[1:]]
        modified_parity = 0 if parity == 1 else 0
        if ((i % 2) + modified_parity) % 2 == 0:
            det += matrix[0][i] * matrix_det(modified_matrix, modified_parity)
        else:
            det -= matrix[0][i] * matrix_det(modified_matrix, modified_parity)
    return det
```

Рисунок 10 – Реализация вычисления определителя матрицы

```
def inverse_matrix(matrix):
    det = matrix_det(matrix, 0)
    if len(matrix[0]) != len(matrix):
        print("It is impossible to calculate the determinant of a given matrix")
        return None
    if det == 0:
        print("Inverse matrix not defined")
        return None
    inversed = []
    for i in range(len(matrix[0])):
        inversed.append([])
        for j in range(len(matrix)):
            modified_matrix = [row[:i] + row[i + 1:] for row in matrix[:j] + matrix[j + 1:]]
            inversed[i].append(matrix_det(modified_matrix, 0) / det * (-1) ** (i + j))
    return inversed
```

Рисунок 11 – Реализация вычисления обратной матрицы

Проверим корректность работы программы, сравнив вывод собственной функции вычисления обратной матрицы и ее аналога из библиотеки numpy. Видим, что все работает корректно для матрицы размерности 3×3, а также 4×4.

```
matrix = input_matrix()

print("Inversed Matrix:")
inverse = inverse_matrix(matrix)
if inverse:
    print_matrix(inverse)

print("Inversed Matrix With NumPy:")
print_matrix(np.linalg.inv(matrix))
```

Рисунок 12 – Реализация вывода обратной матрицы, вычисленной двумя способами (собственная функция и функция библиотеки numpy)

```
Enter 3 elements for row 1: 3 4 5
Enter 3 elements for row 2: 43 2 3
Enter 3 elements for row 3: 88 3 1
Inversed Matrix:
-0.011146496815286623 0.01751592356687898   0.0031847133757961785
0.3519108280254777    -0.695859872611465    0.32802547770700635
-0.07484076433121019  0.5461783439490446    -0.2643312101910828

Inversed Matrix With NumPy:
-0.011146496815286623 0.017515923566878977  0.0031847133757961815
0.3519108280254777    -0.6958598726114649   0.3280254777070063
-0.07484076433121019  0.5461783439490445    -0.2643312101910828
```

Рисунок 13 – Пример вывода обратной матрицы, вычисленной двумя способами (собственная функция и функция библиотеки numpy)

```
Enter 4 elements for row 1: 23 45 34 45
Enter 4 elements for row 2: 34 445 45 1
Enter 4 elements for row 3: 12 2 3 4
Enter 4 elements for row 4: 34 2 2 2
Inversed Matrix:
0.03965248384940967   -0.002450434395188238 -0.5439964357317888   0.1970372020494542
0.27255513477389176   -0.014034306081532635 -3.615615950100245    1.1057585208286924
-2.765649365114725    0.16529293829360658   36.69503230118067     -11.245600356426822
1.8190020049008688    -0.10960124749387391  -23.83147694364001    7.290209400757407

Inversed Matrix With NumPy:
0.03965248384941547   -0.002450434395188563 -0.5439964357318692   0.19703720204948053
0.2725551347739315    -0.014034306081534767 -3.6156159501007723   1.1057585208288534
-2.7656493651151286   0.16529293829362823   36.69503230118602     -11.245600356428458
1.8190020049011315    -0.10960124749388798  -23.83147694364349    7.290209400758471
```

Рисунок 13 – Пример вывода обратной матрицы, вычисленной двумя способами (собственная функция и функция библиотеки numpy)

Теперь с помощью библиотеки timeit сравним быстродействие собственной функции с ее аналогом из библиотеки numpy. Импортируем библиотеку с помощью *import timeit* и проведем несколько тестов. На рисунке 14 представлен код, в котором для измерения времени работы программы используется функция timeit.timeit. Эта функция принимает два обязательных аргумента и несколько необязательных:

1) строка с кодом, который нужно измерить по времени;
2) *globals = globals()* – это параметр, который передает глобальные переменные и функции в вызываемый код
3) number = 1000 – это необязательный параметр, определяющий количество запусков кода для измерения времени.

```
# Создаем тестовую матрицу
matrix = [[1, 2, 3], [23231, 2, 34], [143, 2343, 77]]



# Замеряем время выполнения функции inverse_matrix
time_inv = timeit.timeit('inverse_matrix(matrix)', globals=globals(), number=1000)

# Замеряем время выполнения функции inverse_matrix_numpy
time_inv_numpy = timeit.timeit('inverse_matrix_numpy(matrix)', globals=globals(), number=1000)

print(f"Time for inverse_matrix: {time_inv}")
print(f"Time for inverse_matrix_numpy: {time_inv_numpy}")
```

Рисунок 14 – Реализация тестирования быстродействия функций

Таблица 1. Результаты тестов быстродействия функций.

Собственная функция	Функция из библиотеки numpy
0.011881290993187577	0.011473458027467132
0.011944457946810871	0.010947040980681777
0.01197537500411272	0.011442833987530321
0.01200620800955221	0.010103125008754432

В ходе проведения тестов было замечено, что при малых числах, содержащихся в матрице, функции сравнимы друг с другом по скорости. Однако с увеличением чисел собственная функция показывает результаты значительно хуже.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы было проведено знакомство с синтаксисом языка Python, а также с его библиотеками такими как numpy и timeit. Сначала была написана простая программа «FizzBuzz». После этого были написаны собственные функции для транспонирования и определения ранга матрицы, поиска обратной матрицы и перемножения матриц. Данные программы были протестированы на корректность работы, а программа для поиска обратной матрицы дополнительно была проверена на быстродействие. Также данные функции были реализованы функциями библиотеки numpy. После этого были сделаны выводы о библиотеке numpy, выделены достоинства и недостатки работы с ней. Было наглядно показано преимущество в скорости выполнения функции из numpy по отношению к функции, написанной самостоятельно.

В качестве вывода можно сказать, что использование numpy существенно ускоряет и упрощает выполнение операций над матрицами и многомерными массивами в Python. Однако при разработке программ необходимо учитывать как плюсы, так и минусы данной библиотеки, выбирая подходящий инструмент в зависимости от поставленной задачи.

СПИСОК ЛИТЕРАТУРЫ

1) Wikipedia. Ранг матрицы. [Электронный ресурс] – (Дата последнего обращения 19.05.2024).

ПРИЛОЖЕНИЕ

Для удобства все файлы выгружены на GitHub: https://github.com/kathykkKk/Algorithms-and-Data-Structures-ICT.git