Лабораторная работа №8

Для выполнения данной лабораторной работы Вам необходимо выполнить любые **четыре** задания, представленные ниже, с ***использованием рекурсивных функций***.

Все задания должны быть помещены в репозиторий «*recursion\_lz*» в GitHUB. Каждое задание выполняется в отдельном файле, должно быть снабжено комментариями, а также иметь дружественный интерфейс (пользователь должен понимать, что вводит, и что получает назад).

Входные данные могут быть получены следующими способами: генерация с помощью библиотеки ***random***, ввод с клавиатуры, чтение из файла. Вывод должен осуществляться **в консоль**

**1. Решение задачи о выходе из лабиринта**

Создайте файл «labyrinth.py» или «labyrinth.ipynb».  
Напишите рекурсивную функцию, которая находит путь из начала до конца в лабиринте. Лабиринт задан как двумерный массив, где 0 — проходимая клетка, а 1 — стена. Функция должна возвращать список координат пути или None, если пути нет.

**Пример:**

maze = [

[0, 0, 1, 0, 0],

[1, 0, 1, 1, 0],

[0, 0, 0, 0, 0],

[1, 1, 1, 1, 0],

[0, 0, 0, 1, 0]

]

exit\_path = find\_path(maze, (0, 0), (4, 4))

**2. Перевод числа в систему счисления с основанием N**

Создайте файл «notation.py» или «notation.ipynb».  
Напишите рекурсивную функцию для перевода числа в произвольную систему счисления (от 2 до 16).

**Пример:**

convert\_to\_base(255, 16) = "FF"

convert\_to\_base(255, 2) = "11111111"

**3. Генерация всех перестановок элементов списка**

Создайте файл «exchanges.py» или «exchanges.ipynb».  
Напишите рекурсивную функцию для генерации всех возможных перестановок элементов списка.

**Пример:**

permutations([1, 2, 3]) = [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

**4. Сложение двух больших чисел (поразрядное)**

Создайте файл «addition.py» или «addition.ipynb».  
Напишите рекурсивную функцию для сложения двух больших чисел, представленных как строки, с учётом переноса.

**Пример:**

add\_large\_numbers("12345678901234567890", "98765432109876543210") = "111111111011111111100"

**5. Рекурсивный поиск наибольшего общего делителя (НОД) с помощью алгоритма Евклида**

Создайте файл «Euclid.py» или «Euclid.ipynb».  
Напишите рекурсивную функцию для нахождения наибольшего общего делителя (НОД) двух чисел с использованием алгоритма Евклида.

**Пример:**

gcd(56, 98) = 14

**6. Проверка на совершенное число**

Создайте файл «perfection.py» или «perfection.ipynb».  
Напишите рекурсивную функцию, которая проверяет, является ли число совершенным. Совершенное число — это число, равное сумме своих делителей (кроме самого числа).

**Пример:**

is\_perfect\_number(6) = True

is\_perfect\_number(28) = True

is\_perfect\_number(12) = False

**7. Построение треугольника Паскаля**

Создайте файл «Pascal.py» или «Pascal.ipynb».  
Напишите рекурсивную функцию для построения треугольника Паскаля до nn-го уровня.

**Пример:**

pascal\_triangle(5) = [

[1],

[1, 1],

[1, 2, 1],

[1, 3, 3, 1],

[1, 4, 6, 4, 1]

]

**8. Решение задачи о разбиении числа на сумму слагаемых**

Создайте файл «terms.py» или «terms.ipynb».  
Напишите рекурсивную функцию для нахождения всех возможных разбиений числа на сумму целых чисел.

**Пример:**

partitions(4) = [[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]

**9. Палиндромная последовательность (последовательность, читающаяся одинаково с обеих сторон)**

Создайте файл «subsequence.py» или «subsequence.ipynb».  
Напишите рекурсивную функцию для нахождения самой длинной палиндромной последовательности в строке.

**Пример:**

longest\_palindrome("babad") = "bab" (или "aba")

longest\_palindrome("cbbd") = "bb"