

**КЛАСС\_11\_\_ ВАРИАНТ\_\_1\_\_**

**ШИФР ­­[ebruw1gq2z8]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **∑** | **Подпись** |
|  |  |  |  |  |  |

|  |
| --- |
| **Задание 1** |
| \* Формулировка задачи  Маша собирается на бал по случаю начала учебного года. В ее гардеробе присутствует 4 цвета: красный, зеленый, голубой и белый. Маша не определилась, сколько именно составляющих будет включать ее наряд, но точно знает, что не менее шести обязательных элементов одежды. При этом, существует как минимум 3 варианта каждой вещи в каждом цвете. Сколько возможных комбинаций одежды Маша может примерить, если учесть, что она уверена, что белый не сочетается ни с каким другим цветом?  Формат входных данных  Число элементов одежды модницы (не менее 6).  Формат выходных данных  Число возможных комбинаций одежды согласно условию задачи | |
| **Код решения:** | |
| \* Код решения задачи. Код программы копируется в виде текста.  count\_items = int(input())  print(9 \*\* count\_items + 3 \*\* count\_items) | |
| **Пример работы программы:** | |
| \* Снимок или снимки экрана работы программы | |
| **Комментарии (при необходимости):** | |
| 1) Не совсем понятно, что означает «как минимум 3 варианта каждой вещи». То есть их может быть сколько угодно.  2) Ответ, данный в условии, на мой взгляд, неверный. Разобьём задачу на конфигурации, посчитаем варианты только белой одежды. 3 белых на каждую из 6 позиций -> 3 ^ 6 = 729. Теперь посчитаем без белого цвета. 3 вещи, которые могут быть окрашены в 3 цвета на 6 позиций. (3 \* 3)^6 = 531441. 531441 + 729 = 532 170  Такое решение для 3 вариантов каждой вещи. | |

|  |
| --- |
| **Задание 2** |
| \* Формулировка задачи  Даны целые числа 0, 1, 2, 3, …, 63. Получить квадратную матрицу (таблицу) 8 на 8, элементами которой являются указанные числа, расположенные в ней по схемам, приведённым ниже.    Формат входных данных  Ничего не вводится.  Формат выходных данных  Требуется вывести целые числа от 0 до 63 | |
| **Код решения:** | |
| \* Код решения задачи. Код программы копируется в виде текста.  # зададим квадратную матрицу, изначально заполнив ее нулями.  # Её значения будем изменять  result = [[0] \* 8 for \_ in range(8)]  # зададим список, хранящий числа от 0 до 63  numbers = list(range(64))  # column\_number - номер столбца  # line\_number - номер строки  column\_number, line\_number = 0, 0  # номер текущего значения, которое будем добавлять в матрицу  current\_number = 0  # используем цикл while, чтобы иметь возможность изменять параметры  while column\_number < 8:  # ограничим выполнение цикла.  # То есть, когда закончатся числа в массиве numbers,  # и count\_number примет значение 64, выполнение цикла остановится  while line\_number < 8 and current\_number < 64:  # добавим в соответствующее место матрицы текущее значение  result[line\_number][column\_number] = numbers[current\_number]  # увеличим счётчик текущего значения  current\_number += 1  # если находимся в четной колонке, то номер строки увеличивается  if column\_number in [0, 2, 4, 6]:  line\_number += 1  # если находимся в нечетной колонке, то номер строки уменьшается  elif column\_number in [1, 3, 5, 7]:  line\_number -= 1  # если находимся в "точке перегиба", то есть в той точке,  # где направление меняется в противоположную сторону,  # то изменяем значение текущей строки  if current\_number % 8 == 0:  # если были в последней строке, то в ней же и остаемся  if line\_number == 8:  line\_number = 7  # если были в начальной строке, то в ней же и остаемся  else:  line\_number = 0  # перейдем к следущему столбцу  column\_number += 1  # последовательно выведем значения матрицы так, что бы  # они занимали ровно четыре позиции  for line\_number in range(8):  for column\_number in range(8):  print("%4d" % result[line\_number][column\_number], end=' ')  print()  print()  # опишем функцию поворота матрицы на 90 градусов вправо  def rotate(matrix):  # создадим новую матрицу, заполним нулями  newMatrix = [[0] \* len(matrix) for i in range(len(matrix))]  # расположим строки в обратном порядке  matrix = matrix[::-1]  # транспонируем матрицу  for line\_number in range(len(matrix)):  for column\_number in range(len(matrix)):  newMatrix[line\_number][column\_number] = matrix[column\_number][line\_number]  # вернем значение новой матрицы  return newMatrix  # применим функцию поворота  result = rotate(result)  # выведем значения новой матрицы  for line\_number in range(8):  for column\_number in range(8):  print("%4d" % result[line\_number][column\_number], end=' ')  print()  print()  # заново зададим матрицу для 3 схемы  result = [[0] \* 8 for \_ in range(8)]  # текущее значение, добавляемое в матрицу  current\_number = 0  # будем заполнять матрицу по диагоналям  # len\_matrix длина диагонали, которую заполняем  # 0 0 0 0 2 0 2 5  # 0 0 -> 1 -> 1 4  # 0 3  #  for len\_matrix in range(1, 8 + 1):  line\_number, column\_number = len\_matrix - 1, 0  for \_ in range(len\_matrix):  # добавим текущее значение  result[line\_number][column\_number] = current\_number  # увеличим текущее значение  current\_number += 1  # сдвинемся по строке вверх, а по столбцу вправо  line\_number -= 1  column\_number += 1  # заполним вторую часть матрицы  # текущее значение  current\_number = 63  #  #  #  # 0 0 0 60  # 0 0 => 0 0 => 0 62 => 59 62  # 0 0 0 0 0 63 0 61 63 58 61 63  #  for len\_matrix in range(1, 8):  line\_number, column\_number = - len\_matrix, -1  for \_ in range(len\_matrix):  result[line\_number][column\_number] = current\_number  current\_number -= 1  line\_number += 1 # сдвигаемся по строке вниз  column\_number -= 1 # сдвигаемся по колонке влево  # зададим новую матрицу, в которую по алгоритму будем перемещать элементы из до этого  # созданной матрицы  newMatrix = [[0] \* 8 for \_ in range(8)]  # левая половина матрицы  # меняем местами элементы в диагоналях с чётным количеством элементов  for len\_matrix in range(2, 8 + 1, 2):  line\_number, column\_number = len\_matrix - 1, 0  for \_ in range(len\_matrix):  newMatrix[line\_number][column\_number] = result[column\_number][line\_number]  line\_number -= 1  column\_number += 1  # элементы диагоналей с нечетным количеством элементов остаются, как есть  for len\_matrix in range(1, 7 + 1, 2):  line\_number, column\_number = len\_matrix - 1, 0  for \_ in range(len\_matrix):  newMatrix[line\_number][column\_number] = result[line\_number][column\_number]  line\_number -= 1  column\_number += 1  # правая половина матрицы  # меняем местами элементы в диагоналях с чётным количеством элементов  for len\_matrix in range(2, 8, 2):  line\_number, column\_number = - len\_matrix, -1  for \_ in range(len\_matrix):  newMatrix[line\_number][column\_number] = result[column\_number][line\_number]  line\_number += 1  column\_number -= 1  # элементы диагоналей с нечетным количеством элементов остаются, как есть  for len\_matrix in range(1, 8, 2):  line\_number, column\_number = - len\_matrix, -1  for \_ in range(len\_matrix):  newMatrix[line\_number][column\_number] = result[line\_number][column\_number]  line\_number += 1  column\_number -= 1  # вывод матрицы  for line\_number in range(len(newMatrix)):  for column\_number in range(len(newMatrix)):  print("%4d" % newMatrix[line\_number][column\_number], end=' ')  print() | |
| **Пример работы программы:** | |
| \* Снимок или снимки экрана работы программы | |
| **Комментарии (при необходимости):** | |
| При копировании решения в редактор кода может возникнуть сдвиг строк в тех местах, где я наглядно описываю принцип заполнения матрицы | |

|  |
| --- |
| **Задание 3** |
| \* Формулировка задачи  Дано натуральное число m. Вставить между некоторыми цифрами 1, 2, 3, 4, 5, 6, 7, 8, 9, записанными именно в таком порядке, знаки «+» и «-» так, чтобы значением получившегося выражения было число m. Например, если m = 122, то подойдёт следующая расстановка знаков: 12+34-5-6+78+9. Если требуемая расстановка знаков невозможна, то вывести «Нет решения».  Формат входных данных  Вводится целое число m.  Формат выходных данных  Требуется вывести, как показано в примере, числа с расставленными знаками «+» и «-», если решение задачи возможно. Иначе вывести фразу «Нет решения». | |
| **Код решения:** | |
| \* Код решения задачи. Код программы копируется в виде текста.  input\_number = int(input()) # вводим число  # во вложенных циклах перебираем  # все возможные варианты действий:  # сложение, вычитание или никакое  for action\_1 in ['', '-', '+']:  for action\_2 in ['', '-', '+']:  for action\_3 in ['', '-', '+']:  for action\_4 in ['', '-', '+']:  for action\_5 in ['', '-', '+']:  for action\_6 in ['', '-', '+']:  for action\_7 in ['', '-', '+']:  for action\_8 in ['', '-', '+']:  # конкатенируем числа строки и действия  final\_string = '1' + action\_1 + '2' + action\_2 + '3' \  + action\_3 + '4' + action\_4 + '5' \  + action\_5 + '6' + action\_6 + '7' \  + action\_7 + '8' + action\_8 + '9' \  if eval(final\_string) == input\_number: # если результат арифметического выражения,  # записанного в строке s, равен данному числу  print(final\_string) # выводим строку  exit() # прекращение работы программы  # если работа программы не была прекращена, то решений нет  print('Нет решения') | |
| **Пример работы программы:** | |
| \* Снимок или снимки экрана работы программы | |
| **Комментарии (при необходимости):** | |
|  | |

|  |
| --- |
| **Задание 4** |
| \* Формулировка задачи  В левом верхнем углу прямоугольной таблицы размером N×M находится черепашка. В каждой клетке таблицы записано некоторое число. Черепашка может перемещаться вправо или вниз, при этом маршрут черепашки заканчивается в правом нижнем углу таблицы. Подсчитаем сумму чисел, записанных в клетках, через которую проползла черепашка (включая начальную и конечную клетку). Найдите наибольшее возможное значение этой суммы и маршрут, на котором достигается эта сумма.  Формат входных данных  В первой строке входных данных записаны два натуральных числа N и M, не превосходящих 100 — размеры таблицы. Далее идет N строк, каждая из которых содержит M чисел, разделенных пробелами — описание таблицы. Все числа в клетках таблицы целые и могут принимать значения от 0 до 100.    Формат выходных данных  Первая строка выходных данных содержит максимальную возможную сумму, вторая — маршрут, на котором достигается эта сумма. Маршрут выводится в виде последовательности, которая должна содержать N-1 букву D, означающую передвижение вниз и M-1 букву R, означающую передвижение направо. Если таких последовательностей несколько, необходимо вывести ровно одну (любую) из них. | |
| **Код решения:** | |
| # опишем рекурсивную функцию для решения задачи  # когда достигаем края, то возвращем значение ячейки  def task(n, m, matrix):  if n > 1:  if m > 1:  (dv, dp), (rv, rp) = task(n - 1, m, matrix), task(n, m - 1, matrix)  else:  (dv, dp), (rv, rp) = task(n - 1, m, matrix), (0, "")  else:  return matrix[-n][-m] + 0, ""  # сравнение результатов  if dv > rv:  return matrix[-n][-m] + dv, "D " + dp  else:  return matrix[-n][-m] + rv, "R " + rp  # ввод первых двух чисел  n, m = [int(x) for x in input().split()]  # ввод матрицы  matrix = []  for \_ in range(n):  matrix.append([int(element) for element in input().split()])  result = task(n, m, matrix)  print(result[0]) # максимальная сумма  print(result[1]) # последовательность | |
| **Пример работы программы:** | |
| \* Снимок или снимки экрана работы программы | |
| **Комментарии (при необходимости):** | |
|  | |