# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №1

# По дисциплине: «Инструментальное программное обеспечение»

Группа ПО-455

## Выполнил К.В.Инкин

Шифр 19

# 2021

**23 Дайте определение понятию массив.**

Часто в программах бывает надо работать с большим количеством однотипных переменных. Например, пусть вам надо записать рост каждого человека в классе — это много целых чисел. Вы можете завести по одной переменной на каждого ученика, но это очень неудобно. Специально для этого придуманы массивы.

Массив (в питоне еще принято название "список", это то же самое) — это переменная, в которой хранится много значений. Массив можно представлять себе в виде такой последовательности ячеек, в каждой из которых записано какое-то число:

Значения, хранящиеся в массиве (говорят: элементы массива) нумеруются последовательно, **начиная с нуля**.

Соответственно, переменная теперь может хранить целиком такой массив. Создается такой массив, например, путем перечисления значений в квадратных скобках:

a = [7, 5, -3, 12, 2, 0]

Теперь переменная a хранит этот массив. К элементам массива можно обращаться тоже через квадратные скобки: a[2] — это элемент номер 2, т.е. в нашем случае это -3. Аналогично, a[5] — это 0. В квадратных скобках можно использовать любые арифметические выражения, и даже другие переменные: a[2\*2-1] — это 12, a[i] обозначает "возьми элемент с номером, равным значению переменной i", аналогично a[2\*i+1] обозначает "возьми элемент с номером, равным 2\*i+1", или даже a[a[4]] обозначает "возьми элемент с номером, равным четвертому элементу нашего массива" (в нашем примере a[4] — это 2, поэтому a[a[4]] — это a[2], т.е. -3).

Если указанный номер слишком большой (больше длины массива), то питон выдаст ошибку (т.е. в примере выше a[100] будет ошибкой, да и даже a[6] тоже). Если указан отрицательный номер, то тут действует хитрое правило. Отрицательные номера обозначают нумерацию массива с конца: a[-1] — это всегда последний элемент, a[-2] — предпоследний и т.д. В нашем примере a[-6] равно 7. Слишком большой отрицательный номер тоже дает ошибку (в нашем примере a[-7] уже ошибка).

С элементами массива можно работать как с привычными вам переменными. Можно им присваивать значения: a[3] = 10, считывать с клавиатуры: a[3] = int(input()), выводить на экран: print(a[3]), использовать в выражениях: a[3+i\*a[2]] = 3+abs(a[1]-a[0]\*2+i) (здесь i — какая-то еще целочисленная переменная для примера), использовать в if'ах: if a[i]>a[i-2]:, или for a[2] in range(i) и т.д. Везде, где раньше использовались переменные, можно теперь использовать элемент массива.

## Обход массива

Но обычно надо работать сразу со всеми элементами массива. Точнее, сразу со всеми, как правило, не надо, надо по очереди с каждым (говорят: "пробежаться по массиву"). Для этого пригодится очень полезная вещь — это цикл for. Если в массиве n элементов (т.е. если с есть переменная n и в ней хранится число элементов в массиве), то это делается так:

for i in range(n):

... что-то сделать с элементом a[i]

например, вывести все элементы массива на экран:

for i in range(n):

print(a[i])

или увеличить все элементы массива на единицу:

for i in range(n):

a[i] += 1

и т.п.

Конечно, в цикле можно и несколько действий делать, если надо.

Если же нет переменной n, то всегда можно воспользоваться специальной функцией len, которая возвращает количество элементов в массиве:

for i in range(len(a)): ...

Функцию len, конечно, можно использовать где угодно, не только в заголовке цикла. Например, просто вывести длину массива — print(len(a)).

## Операции на массиве

Еще ряд полезных операций с массивами:

* a[i]— элемент массива с номером i.
* len(a)— длина массива.
* a.append(x) — приписывает к массиву новый элемент со значением x, в результате длина массива становится на 1 больше. Конечно, вместо x может быть любое арифметическое выражение.
* a.pop() — симметричная операция, удаляет последний элемент из массива. Длина массива становится на 1 меньше. Если нужно запомнить значение удаленного элемента, надо просто сохранить результат вызова pop в новую переменную: res = a.pop().
* a \* 3 — это массив, полученный приписыванием массива a самого к себе три раза. Например, [1, 2, 3] \* 3 — это [1, 2, 3, 1, 2, 3, 1, 2, 3]. Конечно, на месте тройки тут может быть любое арифметическое выражение. Самое частое применение этой конструкции — если нужен массив длины n, заполненный, например, нулями, то вы пишете [0] \* n.
* b = a — присваивание массивов. Теперь в b записан тот же массив, что и в a. Тот же — в прямом смысле слова: теперь и a, и b соответствуют **одному и тому же массиву**, и **изменения в b отразятся в a и наоборот.** Еще раз, потому что это очень важно. Присваивание массивов (и вообще любых сложных объектов) в питоне **не копирует массив**, а просто обе переменные начинают ссылаться на один и тот же массив, и изменения массива через любую из них меняет один и тот же массив.
* b = a[1:4] ("срез") — делает новый массив, состоящий из элементов старого массива начиная со первого (помните про нумерацию с нуля!) и заканчивая третьим (т.е. до четвертого, но не включительно, аналогично тому, как работает range); этот массив сохраняется в b. Для примера выше получится [5, -3, 12]. Конечно, на месте 1 и 4 может быть любое арифметическое выражение. Более того, эти индексы можно вообще не писать, при этом автоматически подразумевается начало и конец массива. Например, a[:3] — это первые три элемента массива (нулевой, первый и второй), a[1:] — все элементы кроме нулевого, a[:-1] — все элементы кроме последнего (!), а a[:] — это копия всего массива. И это именно **копия**, т.е. запись b = a[:] именно копирует массив, получающиеся массивы никак не связаны, и изменения в b не влияют на a (в отличие от b = a).

**57 Опишите синтаксис создания множеств.**

Множество — это «мешок», содержащий неупорядоченные уникальные значения. Одно множество может содержать значения любых типов. Над любыми множествами можно совершать любые стандартные операции, например, объединение, пересечение и разность.

Создание множества

>>> a\_set = {1}

>>> type(a\_set)

<class 'set'>

>>> a\_set = {1, 2}

>>> a\_set

{1, 2}

Чтобы создать множество с одним значением, нужно поместить его в фигурные скобки ({}).

Чтобы создать множество с несколькими значениями, нужно отделить их друг от друга запятыми и поместить внутрь фигурных скобок.

Также можно создать множество из списка.

>>> a\_list = ['a', 'b', 'mpilgrim', True, False, 42]

>>> a\_set = set(a\_list)

>>> a\_set

{'a', False, 'b', True, 'mpilgrim', 42}

Если добавить элементы в множество, оно не запоминает, в каком порядке они добавлялись.

Можно создать пустое множество.

>>> a\_set = set()

>>> a\_set

set()

>>> type(a\_set)

<class 'set'>

>>> len(a\_set)

0

>>> not\_sure = {}

>>> type(not\_sure)

<class 'dict'>

Чтобы создать пустое множество, нужно вызвать set() без аргументов.

Нельзя создать пустое множество с помощью двух фигурных скобок. На самом деле, они создают пустой словарь, а не множество.

Изменение множества

Есть два способа добавить элементы в существующее множество: метод add() и метод update().

>>> a\_set = {1, 2}

>>> a\_set.add(4)

>>> a\_set

{1, 2, 4}

>>> len(a\_set)

3

>>> a\_set.add(1)

>>> a\_set

{1, 2, 4}

>>> len(a\_set)

3

Метод add() принимает один аргумент, который может быть любого типа, и добавляет данное значение в множество.

Теперь множество содержит 3 элемента.

Множества — мешки уникальных значений. Если попытаться добавить значение, которое уже присутствует в множестве, ничего не произойдет. Это не приведет в возникновению ошибки; просто нулевое действие.

Это множество все ещё состоит из 3 элементов.

>>> a\_set = {1, 2, 3}

>>> a\_set

{1, 2, 3}

>>> a\_set.update({2, 4, 6})

>>> a\_set

{1, 2, 3, 4, 6}

>>> a\_set.update({3, 6, 9}, {1, 2, 3, 5, 8, 13})

>>> a\_set

{1, 2, 3, 4, 5, 6, 8, 9, 13}

>>> a\_set.update([10, 20, 30])

>>> a\_set

{1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 20, 30}

Метод update() принимает один аргумент — множество, и добавляет все его элементы к исходному множеству. Так, как если бы вы вызывали метод add() и по очереди передавали ему все элементы множества.

Повторяющиеся значения игнорируются, поскольку множество не может содержать дубликаты.

Вообще-то, можно вызвать метод update() с любым количеством параметров. Когда он вызывается с двумя множествами, метод update() добавляет все элементы обоих множеств в исходное множество (пропуская повторяющиеся).

Метод update() может принимать объекты различных типов, включая списки. Когда ему передается список, он добавляет все его элементы в исходное множество.

**73 Напишите программу, которая выводит на экран таблицу значений функции**  **в диапазоне от -2 до 2. Шаг изменения аргумента 0,1.**

Для красивого оформления результатов работы программы применим округление значений функции до 2 знаков после запятой. Для избавления от модуля math заменим операцию взятия квадратного корня операцией возведения в степень 0,5.

Блок-схема алгоритма решения задачи представлена на рисунке 1.

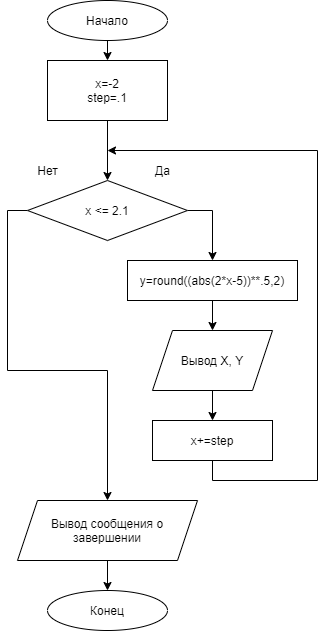


Рисунок 1 – Блок схема алгоритма решения задачи.

**Текст программы.**

# 73 Напишите программу, которая выводит на экран таблицу значений функции в диапазоне от -2 до 2. Шаг изменения аргумента 0,1.

х=-2

step = .1

while x <= 2.1: # в цикле перебираем значения функции

y=round((abs(2\*x-5))\*\*.5,2)

print('X= ', round(x,1), 'Y= ',y)

x+=step

print('Работа завершена.')

**90 Решите задачу. Дан двумерный массив nхn. Заполните его по строкам с клавиатуры. Определите максимальный элемент в побочной диагонали и поменяйте местами столбец содержащего его с первым столбцом.**

Блок-схема алгоритма решения задачи представлена на рисунке 2.

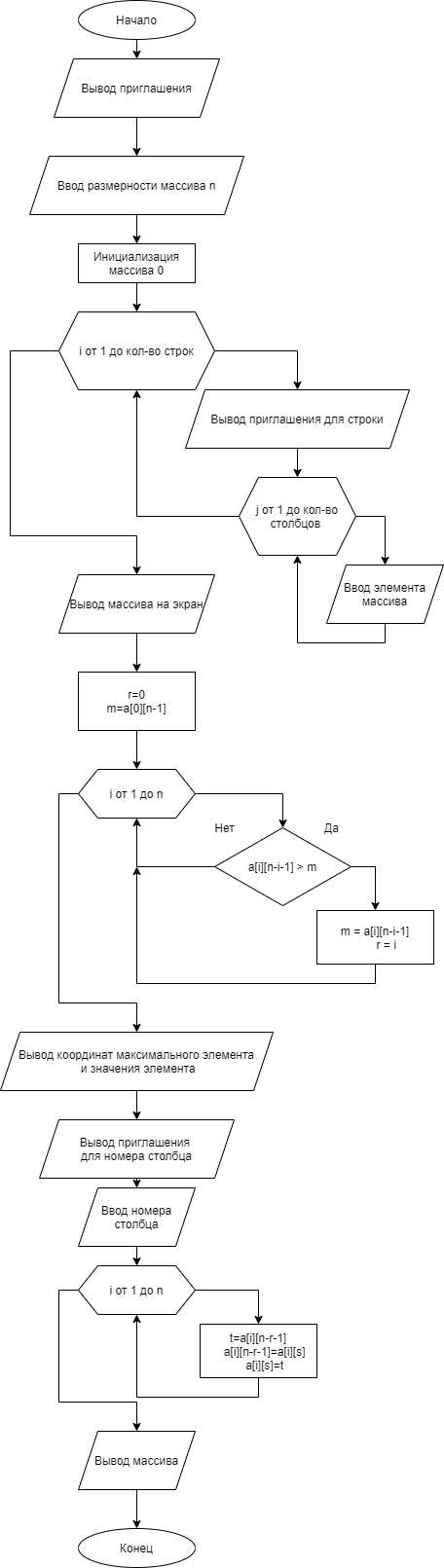


Рисунок 2- блок схема алгоритма решения задачи.

**Текст программы.**

# Дан двумерный массив n х n. Заполните его по строкам с клавиатуры. Определите

# максимальный элемент в побочной диагонали и поменяйте

# местами столбец содержащего его с первым столбцом.

print('Введите размерность массива N') # выводим на экран приглашение

n=int(input())

a=[[0] \* n for i in range(n)]

for i in range(n):

print('Вводите строку ',i+1)

for j in range(n):

a[i][j]=int(input())

print(a) # выводим массив на экран

r=0

m=a[0][n-1]

for i in range(n):

if a[i][n-i-1] > m:

m = a[i][n-i-1]

r = i

print('Координаты максимального элемента i: ',r+1, 'j: ',n-r)

print('Максимальный элемент: ',m)

print('Номер столбца c которым поменять столбец ',n-r,': ')

s= int(input())-1 # считываем номер столбца

t=0

for i in range(n):

t=a[i][n-r-1]

a[i][n-r-1]=a[i][s]

a[i][s]=t

print(a)

**Список использованных источников**

1 Прохоренок Н. А. Python 3. Самое необходимое / Н. А. Прохоренок, В. А. Дронов. — СПб. : БХВ-Петербург, 2016. — 464 с.: ил

2 Плас, Дж. В. Python для сложных задач: наука о данных и машинное обучение / Дж. В. Плас. — СПб.: Питер, 2018. — 576 с.: ил.

3 Саммерфилд, М. Программирование на Python 3. Подробное руководство / М. Саммерфилд. - М.: Символ, 2016. - 608 c.