# 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 Опишите синтаксис создания свойств и декораторов**

**классов в языке Python.**

Создание классов в Python:

class SomeClass(object):

# поля и методы класса SomeClass

Классы-родители перечисляются в скобках через запятую:

class SomeClass(ParentClass1, ParentClass2, …):

# поля и методы класса SomeClass

Свойства классов устанавливаются с помощью простого присваивания:

class SomeClass(object):

attr1 = 42

attr2 = "Hello, World"

Методы объявляются как простые функции:

class SomeClass(object):

def method1(self, x):

# код метода

Первый аргумент – self – общепринятое имя для ссылки на объект, в контексте которого вызывается метод. Этот параметр обязателен и отличает метод класса от обычной функции.

Все пользовательские атрибуты сохраняются в атрибуте \_\_dict\_\_, который является словарем.

Инстанцировать класс в Python:

class SomeClass(object):

attr1 = 42

def method1(self, x):

return 2\*x

obj = SomeClass()

obj.method1(6)

obj.attr1

Можно создавать разные инстансы одного класса с заранее заданными параметрами с помощью инициализатора (специальный метод \_\_init\_\_). Для примера возьмем класс Point (точка пространства), объекты которого должны иметь определенные координаты:

class Point(object):

def \_\_init\_\_(self, x, y, z):

self.coord = (x, y, z)

p = Point(13, 14, 15)

p.coord # (13, 14, 15)

**Динамическое изменение**

Можно обойтись без определения атрибутов и методов:

class SomeClass(object):

pass

Классы в Python могут динамически изменяться после определения:

class SomeClass(object):

pass

def squareMethod(self, x):

return x\*x

SomeClass.square = squareMethod

obj = SomeClass()

obj.square(5) # 25

**Статические и классовые методы**

Для создания статических методов в Python предназначен декоратор @staticmethod. У них нет обязательных параметров-ссылок вроде self. Доступ к таким методам можно получить как из экземпляра класса, так и из самого класса:

class SomeClass(object):

@staticmethod

def hello():

print("Hello, world")

SomeClass.hello() # Hello, world

obj = SomeClass()

obj.hello() # Hello, world

Еще есть так называемые методы классов. Они аналогичны методам экземпляров, но выполняются не в контексте объекта, а в контексте самого класса (классы – это тоже объекты). Такие методы создаются с помощью декоратора @classmethod и требуют обязательную ссылку на класс (cls).

class SomeClass(object):

@classmethod

def hello(cls):

print('Hello, класс {}'.format(cls.\_\_name\_\_))

SomeClass.hello() # Hello, класс SomeClass

Статические и классовые методы доступны без инстанцирования.

**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.

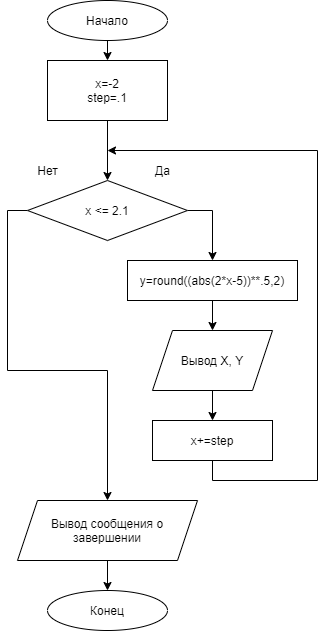


Рисунок 1 – Блок схема алгоритма решения задачи.

**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.

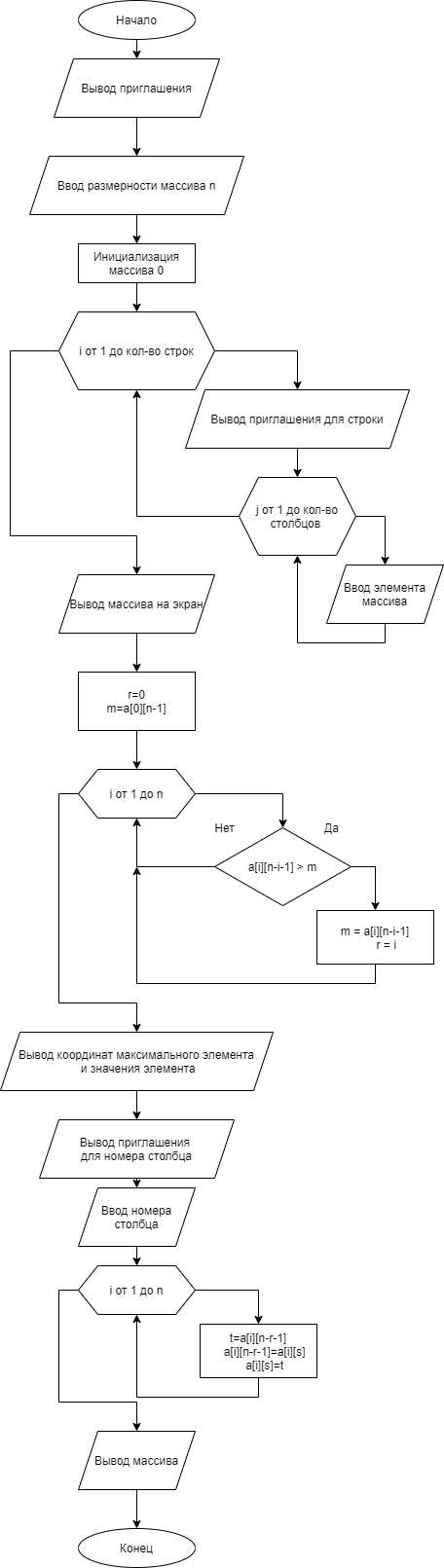


Рисунок 2- блок схема алгоритма решения задачи.

**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.