# 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 О. И. Чижик

Шифр 14

# 2021

1. **Опишите синтаксис создания функций в языке Python.**

Для определения функции нужно всего лишь написать ключевое слово def перед ее именем, а после — поставить двоеточие. Следом идет блок инструкций.

def compute\_surface(radius):

from math import pi

return pi \* radius \* radius

Последняя строка в блоке инструкций может начинаться с return, если нужно вернуть какое-то значение. Если инструкции return нет, тогда по умолчанию функция будет возвращать объект None. Как в этом примере:

i = 0

def increment():

global i

i += 1

Функция инкрементирует глобальную переменную i и возвращает None (по умолчанию).

Для вызова функции, которая возвращает переменную, нужно ввести:

surface = compute\_surface(1.)

Для вызова функции, которая ничего не возвращает:

increment()

Функцию можно записать в одну строку, если блок инструкций представляет собой простое выражение:

def sum(a, b): return a + b

Функции могут быть вложенными:

def func1(a, b):

def inner\_func(x):

return x\*x\*x

return inner\_func(a) + inner\_func(b)

Функции — это объекты, поэтому их можно присваивать переменным.

Аргументы функции можно использовать для изменения ввода и таким образом получать вывод функции. Но куда удобнее использовать инструкцию return, примеры которой уже встречались ранее. Если ее не написать, функция вернет значение None.

Пока что функция возвращала только одно значение или не возвращала ничего (объект None). А как насчет нескольких значений? Этого можно добиться с помощью массива. Технически, это все еще один объект. Например:

def stats(data):

"""данные должны быть списком"""

\_sum = sum(data) # обратите внимание на подчеркивание, чтобы избежать переименования встроенной функции sum

mean = \_sum / float(len(data)) # обратите внимание на использование функции float, чтобы избежать деления на целое число

variance = sum([(x-mean)\*\*2/len(data) for x in data])

return mean,variance # возвращаем x,y — кортеж!

m, v = stats([1, 2, 1])

В функции можно использовать неограниченное количество параметров, но число аргументов должно точно соответствовать параметрам. Эти параметры представляют собой позиционные аргументы. Также Python предоставляет возможность определять значения по умолчанию, которые можно задавать с помощью аргументов-ключевых слов.

Параметр — это имя в списке параметров в первой строке определения функции. Он получает свое значение при вызове. Аргумент — это реальное значение или ссылка на него, переданное функции при вызове. В этой функции:

def sum(x, y):

return x + y

x и y — это параметры, а в этой:

sum(1, 2)

1 и 2 — аргументы.

При определении функции параметры со значениями по умолчанию нужно указывать до позиционных аргументов:

def compute\_surface(radius, pi=3.14159):

return pi \* radius \* radius

Если использовать необязательный параметр, тогда все, что указаны справа, должны быть параметрами по умолчанию.

В следующем примере допущена ошибка:

def compute\_surface(radius=1, pi):

return pi \* radius \* radius

Для вызовов это работает похожим образом. Сначала нужно указывать все позиционные аргументы, а только потом необязательные:

S = compute\_surface(10, pi=3.14)

На самом деле, следующий вызов корректен (можно конкретно указывать имя позиционного аргумента), но этот способ не пользуется популярностью:

S = compute\_surface(radius=10, pi=3.14)

А этот вызов некорректен:

S = compute\_surface(pi=3.14, 10)

При вызове функции с аргументами по умолчанию можно указать один или несколько, и порядок не будет иметь значения:

def compute\_surface2(radius=1, pi=3.14159):

return pi \* radius \* radius

S = compute\_surface2(radius=1, pi=3.14)

S = compute\_surface2(pi=3.14, radius=10.)

S = compute\_surface2(radius=10.)

Можно не указывать ключевые слова, но тогда порядок имеет значение. Он должен соответствовать порядку параметров в определении:

S = compute\_surface2(10., 3.14)

S = compute\_surface2(10.)

Если ключевые слова не используются, тогда нужно указывать все аргументы:

def f(a=1,b=2, c=3):

return a + b + c

Второй аргумент можно пропустить:

f(1,,3)

Чтобы обойти эту проблему, можно использовать словарь:

params = {'a':10, 'b':20}

S = f(\*\*params)

**51 Опишите методы обработки множеств в языке Python.**

Множество — это «мешок», содержащий неупорядоченные уникальные значения. Одно множество может содержать значения любых типов. Если у вас есть два множества, можно совершать над ними любые стандартные операции, например, объединение, пересечение и разность.

Создание множества

>>> a\_set = {1}

>>> a\_set

{1}

>>> type(a\_set)

<class 'set'>

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

>>> a\_set

{1, 2}

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

Множества, вообще-то, реализуются как классы, но пока не беспокойтесь об этом.

Чтобы создать множество с несколькими значениями, отделите их друг от друга запятыми и поместите внутрь фигурных скобок.

Также вы можете 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 вы добавляете 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'>

С множествами можно выполнять множество операций: находить объединение, пересечение...

len(s) - число элементов в множестве (размер множества).

x in s - принадлежит ли x множеству s.

set.isdisjoint(other) - истина, если set и other не имеют общих элементов.

set == other - все элементы set принадлежат other, все элементы other принадлежат set.

set.issubset(other) или set <= other - все элементы set принадлежат other.

set.issuperset(other) или set >= other - аналогично.

set.union(other, ...) или set | other | ... - объединение нескольких множеств.

set.intersection(other, ...) или set & other & ... - пересечение.

set.difference(other, ...) или set - other - ... - множество из всех элементов set, не принадлежащие ни одному из other.

set.symmetric\_difference(other); set ^ other - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.

set.copy() - копия множества.

И операции, непосредственно изменяющие множество:

set.update(other, ...); set |= other | ... - объединение.

set.intersection\_update(other, ...); set &= other & ... - пересечение.

set.difference\_update(other, ...); set -= other | ... - вычитание.

set.symmetric\_difference\_update(other); set ^= other - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.

set.add(elem) - добавляет элемент в множество.

set.remove(elem) - удаляет элемент из множества. KeyError, если такого элемента не существует.

set.discard(elem) - удаляет элемент, если он находится в множестве.

set.pop() - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.

set.clear() - очистка множества.

**67 Создать список заданной длины и записать в него число**

Блок-схема алгоритма представлена на рисунке 2

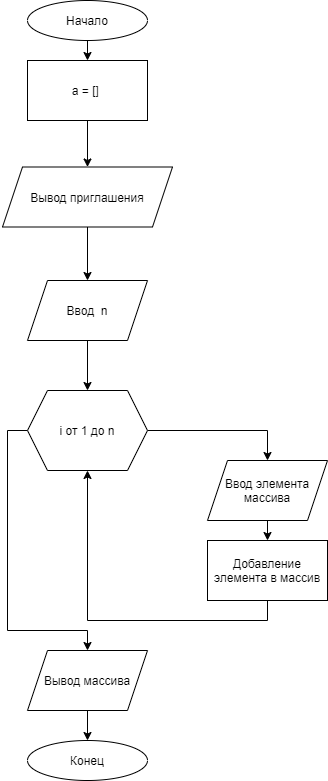


Рисунок 2 – Блок схема алгоритма программы

**Текст программы**

# 67 Создать список заданной длины и записать в него заданное число

# Чижик О. И.

a = [] # заводим пустой список

print('Введите количество')

n = int(input()) # считываем количество элементов в списке

print("Нужно ввести элементов: ",n)

for i in range(n): # в цикле считываем элемент и добавляем его

print('Элемент ', i+1, ':')

new\_element = int(input()) # считываем очередной элемент

a.append(new\_element) # добавляем его в список

print('Список:')

print(a) # выводим список на экран

В результате работы программы мы создали пустой список, и добавили в него указанное количество элементов. По окончании работы результирующий список был выведен на экран.

**91 Решите задачу. Дана матрица A(3,4). Найти наименьший элемент в каждой строке матрицы. Вывести исходную матрицу и результаты вычислений**

Блок-схема алгоритма представлена на рисунке 3

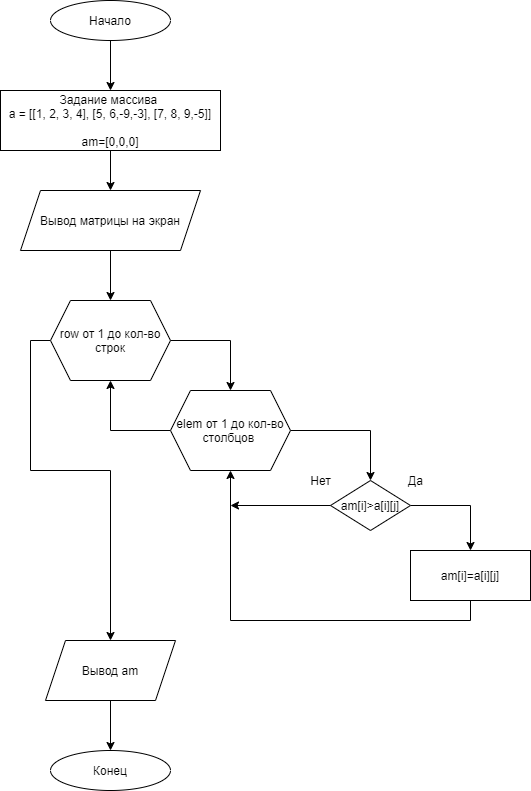


Рисунок 3 – Блок схема алгоритма программы.

**Текст программы**

# 91 Решите задачу. Дана матрица A(3,4). Найти наименьший

# элемент в каждой строке матрицы.

# Вывести исходную матрицу и результаты вычислений

# Чижик О. И.

a = [[1, 2, 3, 4], [5, 6, -9, -3], [7, 8, 9, -5]]

am = [0, 0, 0]

print(a) # выводим массив на экран

for i in range(3): # по каждой строке

am[i] = a[0][0]

for j in range(4): # по каждому элементу строки

if am[i] > a[i][j]:

am[i] = a[i][j]

print('Наименьшие элементы:')

print(am)

Программа по заданному в тексте массиву производит поиск минимального элемента для каждой строки.

**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.