# 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

**15 Перечислите приоритет операций**

Для правильного выполнения вычислений каждая операция (или оператор в языке программирования) имеет свой приоритет, в соответствии с которым выполнение операций производится в строгом соответствии. Вычисления выполняются слева направо, то есть, если в выражении встретятся операторы одинаковых приоритетов, первым будет выполнен тот, что слева. Изменить приоритет по умолчанию можно использованием скобок ().В одном выражении могут встретиться как арифметические, так и логические операции. В таблице 1 перечислены основные операции в порядке их приоритета. Самые приоритетные операции вверху, снизу — с низким приоритетом.

Оператор возведения в степень исключение из этого правила. Из двух операторов \*\* сначала выполнится правый, а потом левый.

Таблица 1. Приоритет операций.

|  |  |
| --- | --- |
| ( ) | Скобки |
| \*\* | Возведение в степень |
| +x, -x, ~x | Унарные плюс, минус и битовое отрицание |
| \*, /, //, % | Умножение, деления, взятие остатка |
| +, - | Сложение и вычитание |
| <<, >> | Битовые сдвиги |
| & | Битовое И |
| ^ | Битовое исключающее ИЛИ (XOR) |
| | | Битовое ИЛИ |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Сравнение, проверка идентичности, проверка вхождения |
| not | Логическое НЕ |
| and | Логическое И |
| or | Логическое ИЛИ |

**39 Опишите структуру протокола HTTP**

HTTP (англ. HyperText Transfer Protocol) - протокол прикладного уровня передачи данных (изначально в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). Определяется международным стандартом RFC 2616.

Основой HTTP является технология «клиент-сервер» - предполагается, что существует множество потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

Одно из ключевых понятий в протоколе - это адрес ресурса, в качестве которого выступает URI. URI (англ. **U**niform **R**esource **I**dentifier) - унифицированный (единообразный) идентификатор ресурса.

Пример URI приведен на рисунке 1. Обязательными атрибутами являются: имя, схема, путь, и Userinfo в случае авторизации.

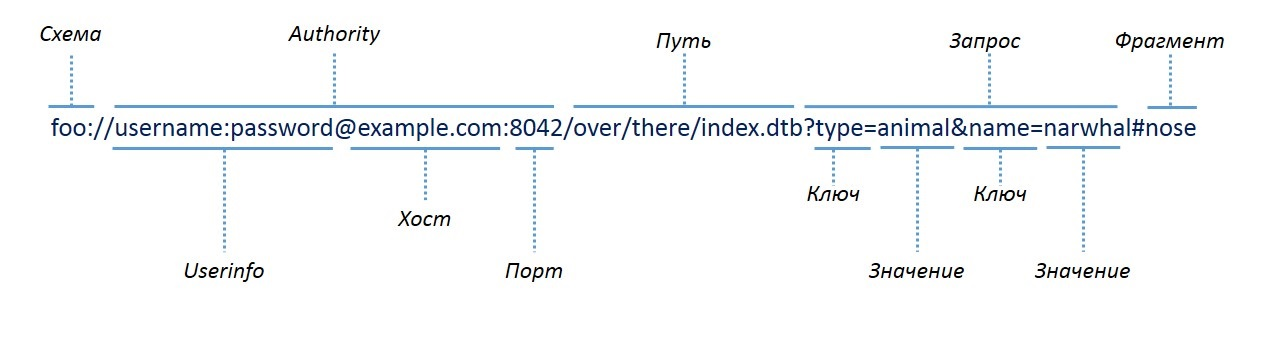
[](https://www.yuripetrov.ru/edu/python/_images/13_01_08.png)

Рисунок 1- Пример URI

Как HTTP-запрос (*англ.* Request), так и HTTP-ответ (*англ.* Response) имеют следующий формат:

1. Стартовая строка (*англ.* Starting Line) — определяет тип сообщения (обязательна);
2. Заголовки (*англ.* Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения (могут отсутствовать);
3. Пустая строка;
4. Тело сообщения (*англ.* Message Body) — непосредственно данные сообщения (например, код HTML-страницы или файл).
5. Стартовая строка.

Стартовая строка записывается по-разному в зависимости от типа сообщения:

* 1. запрос:

Типичная стартовая строка запроса выглядит следующим образом:

GET /path/to/file/index.html HTTP/1.0

где (по порядку, через пробел):

* + 1. GET - наиболее часто используемый HTTP-метод, обозначающий «дай мне вот этот ресурс»;
    2. /path/to/file/index.html - часть URI после имени сервера;
    3. HTTP/1.0 - HTTP-версия, строка формата «HTTP/x.x», в верхнем регистре.

Кроме метода GET часто используются методы HEAD и POST:

* + 1. HEAD: аналогичен GET, но запрашивает у сервера только заголовки (без тела сообщения); полезен для проверки соединения с сервером без нагрузки на канал;
    2. POST: предназначен для отправки данных на сервер и их дальнейшей обработки:
       1. в качестве тела сообщения отправляются какие-либо данные; обычно их формат определяется дополнительными заголовками, определяющими тип данных, размер и т.д.;
       2. POST-запросы часто используются в HTML-формах для отправки каких-либо данных на сервер.

HTTP-запрос также может содержать параметры: в URI они в виде пар ключ-значение через знак амперсанда:

http://example.com/enter?login=admin&password=qwerty

* 1. ответ:

Типичная стартовая строка ответа выглядит следующим образом:

HTTP/1.0 200 OK

где (по порядку, через пробел):

* + 1. HTTP/1.0 - HTTP-версия, строка формата «HTTP/x.x», в верхнем регистре;
    2. 200 - код ответа (англ. HTTP status code) (машиночитаемый);
    3. OK - сообщение (человекочитаемое).

Код ответа и сообщения образуют код состояния, по которому можно определить содержимое ответа сервера. Например:

* + 1. 200 OK: запрос выполнен успешно, ресурс содержится в теле сообщения;
    2. 404 Not Found: запрошенный ресурс не существует.

1. Заголовки

Заголовки HTTP - строки в HTTP-сообщении, содержащие разделенную двоеточием пару параметр-значение (регистр параметра не важен). Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений (RFC 822).

Как правило, клиент в запросах указывает следующие заголовки:

* 1. From: e-mail человека или программы, выполняющей запрос;
  2. User-Agent: наименование программы, выполняющей запрос в формате 'Program-name/x.xx', где 'x.xx' - версия приложения; пример: 'User-Agent: Mozilla/20.1'.

В свою очередь, в ответе, сервер, обычно указывает следующие заголовки:

* 1. Server: аналог User-Agent в запросе; пример: 'Server: Apache/1.2b3-dev'
  2. Last-Modified: дата/время по Гринвичу последнего обновления запрошенного ресурса (может использоваться клиентом для кеширования ресурса); пример: 'Last-Modified: Fri, 31 Dec 1999 23:59:59 GMT'

1. Тело сообщения

Тело HTTP сообщения, если присутствует, используется для передачи данных, связанных с запросом или ответом.

Если тело сообщение не пустое, сообщение содержит дополнительные заголовки:

* 1. Content-Type: MIME-тип тела сообщения (англ. **M**ultipurpose **I**nternet **M**ail **E**xtensions — многоцелевые расширения интернет-почты) (например, "text/html", "application/pdf");
  2. Content-Length: размер тела сообщения в байтах.

Когда клиент хочет взаимодействовать с сервером, являясь конечным сервером или промежуточным прокси, он выполняет следующие шаги:

**1)** Открытие TCP соединения: TCP-соединение будет использоваться для отправки запроса или запросов, и получения ответа. Клиент может открыть новое соединение, переиспользовать существующее, или открыть несколько TCP-соединений к серверу.

**2)** Отправка HTTP-сообщения: HTTP-сообщения (до HTTP/2) -- человеко-читаемо. Начиная с HTTP/2, простые сообщения инкапсулируются во фреймы, делая невозможным их чтения напрямую, но принципиально остаются такими же.

GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr

**3)** Читает ответ от сервера:

HTTP/1.1 200 OK

Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

ETag: "51142bc1-7449-479b075b2891b"

Accept-Ranges: bytes

Content-Length: 29769

Content-Type: text/html

<!DOCTYPE html... (далее идут 29769 байт запрошенной страницы)

**4)** Закрывает или переиспользует соединение для дальнейших запросов.

Если активирован HTTP-конвейер, несколько запросов могут быть отправлены без ожидания получения первого ответа целиком. HTTP-конвейер тяжело внедряется в существующие сети, где старые куски ПО сосуществуют с современными версиями. HTTP-конвейер был заменён в HTTP/2 на более надёжные мультиплексные запросы во фрейме.

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

Блок-схема алгоритма представлена на рисунке 2

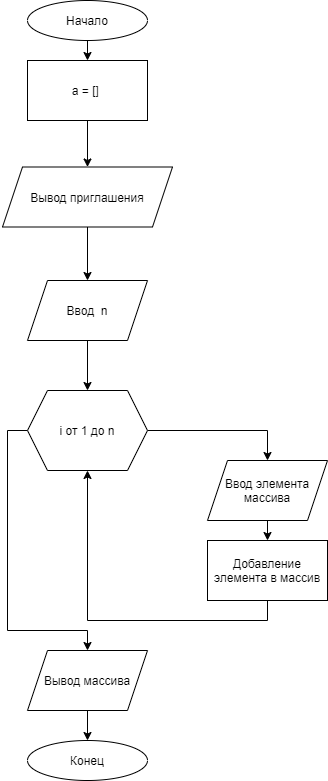


Рисунок 2 – Блок схема алгоритма программы

**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.

**79 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите: количество и сумму отрицательных элементов; максимальное из чисел, в заданном столбце массива**

Блок-схема алгоритма представлена на рисунке 3

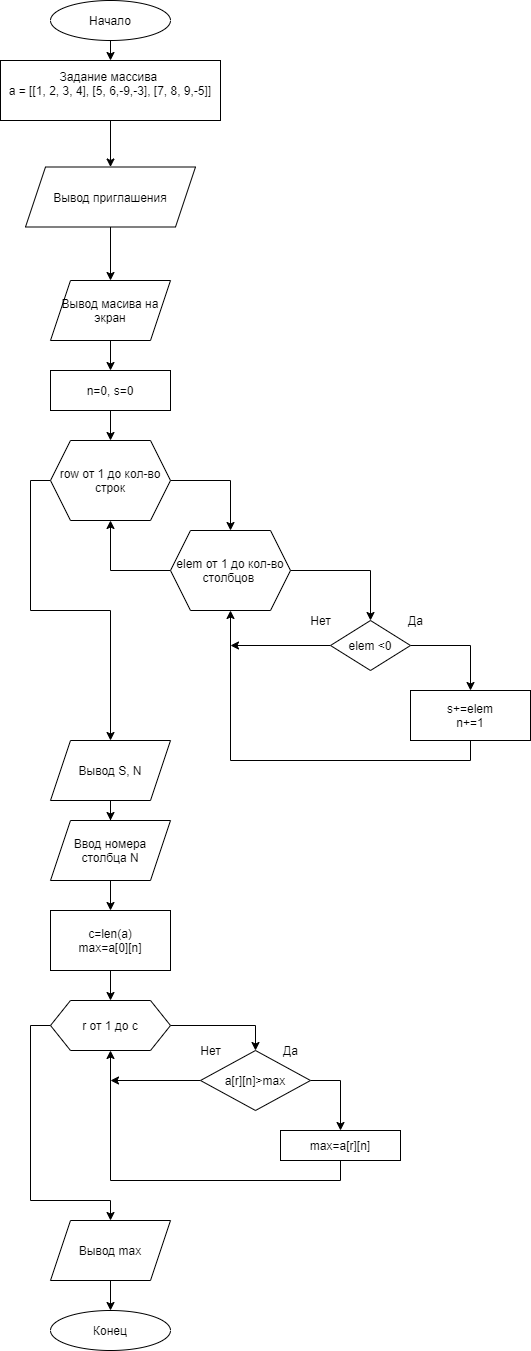


Рисунок 3 – Блок схема алгоритма программы.

**Текст программы**

# 79 Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

# количество и сумму отрицательных элементов;

# максимальное из чисел, в заданном столбце массива

# Чижик О. И.

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

print(a) # выводим массив на экран

n=0

s=0

for row in a:# по каждой строке

for elem in row:# по каждому элементу строки

if elem <0:

s+=elem

n+=1

print('Сумма отрицательных: ',s)

print('Количество отрицательных: ',n)

print('Номер столбца: ')

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

c=len(a)

max=a[0][n]

for r in range(c):

if a[r][n]>max:

max=a[r][n]

print('Максимальный элемент столбца: ',max)

Программа по заданному в тексте массиву производит подсчет количества и суммы отрицательных элементов, а по указанному столбцу – поиск максимального элемента.

**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.