**ЛАБОРАТОРНАЯ РАБОТА №7**

***Тема: Работа с файлами на языке Python***

***Цель: Рассмотрение способов работы с файлами***

***ОГЛАВЛЕНИЕ***

[***1 Работа с файлами***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1)[***1.1 Открытие файла***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.1)[***1.2 Методы работы с файлами***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.2)[***1.3 Передача содержимого скрипта клиенту***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.3)[***1.4 Использование модуля os для работы с файловой системой***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.4)[***1.5 Использование модуля pickle для сериализации и консервации объектов***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.5)[***2 Обработка исключений***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#2)[***2.1 Типы исключений***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#2.1)[***2.2 Использование оператора try***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#2.2)[***2.3 Использование оператора raise***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#2.3)[***2.4 Использование функции/оператора assert***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#2.4)[***Пример №1***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#prim1)[***Индивидуальные задания***](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#ind)

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***1 Работа с файлами***

В языке Python широко используются файлы для выполнения операций с данными, которые предполагается хранить длительное время. Для работы с файлами используется специальный объект файла, который содержит методы для чтения данных из файла и записи данных в файл.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***1.1 Открытие файла***

Открыть файл можно с помощью встроенной функции *open()*:

*open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)*

Функция открывает файл и возвращает соответствующий файловый объект для последующего оперирования с ним. Если файл не может быть открыт – вызывается исключение OSError:

* *file – единственный обязательный параметр, задает имя как текстового, так и двоичного файла (и путь к файлу, если файл не находится в текущем каталоге). Также этот параметр может указывать дескриптор файла в виде целого числа. Если задан дескрипртор файла, то он закрывается, когда закрывается файл (в том случае, если аргумент closefd не равен False);*
* *mode – задает режимы открытия файла:* 
  + *'w' – открывает файл только для записи, если такой файл уже существует – его содержимое будет перезаписано, т.е. размер текущего файлоа будет уменьшен до нуля (truncating), иначе – создается новый файл для записи;*
  + *'r' – открыть файл только для чтения (значение по умолчанию, синоним 'rt'), указатель устанавливается в начало файла, если файл не найден – возникает исключение FileNotFoundError;*
  + *'a' – открыть файл для добавления, если файл существует – предыдущее содержимое файла сохраняется и указатель устанавливается в конец файла (независимо от текущего положения указателя файла), иначе – файл создается для записи;*
  + *'b' – режим двоичных данных;*
  + *'t' – режим текстовых данных (значение по умолчанию);*
  + *'+' – открыть файл для чтения или записи, указатель устанавливается в начало файла. Различие между режимами 'w+' ('w+b') и 'r+' ('r+b') следующее – если указанный файл существует, то в первом случае его содержимое очищается (размер становится равным нулю), а во втором случае – нет;*
  + *'x' – при попытке открыть для записи существующий файл возникает исключение FileExistsError (параметр введен начиная с версии Python 3.3);*

*(значения 'w', 'r', 'a', 'b', 't', '+' и 'x' при совместном использовании могут указываться в произвольном порядке). Python поддерживает два вида файлов – двоичные, при открытии которых для аргумента mode указано значение 'b', и текстовые, при открытии которых аргумент mode не имеет значения 'b'. Текстовые файлы возвращают содержимое в виде строк, а двоичные файлы – в виде объектов типа bytes (см.* [*подраздел 3.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.1) *лаб. раб. №4);*

* *buffering – устанавливает размер буфера при работе с файлом:* 
  + *если размер равен 0 (только для двоичных файлов), то режим буферизации не используется и операции чтения и записи выполняются напрямую с диска на диск;*
  + *если размер равен 1 (только для текстовых файлов), то используется построковая буферизация (значение по умолчанию);*
  + *если размер больше 1, то для двоичных файлов устанавливается размер буфера, т.е. размер блоков, на которые делится файл. Размер буфера выбирается исходя из параметров устройств обмена с целью оптимизации выполнения файловых операций. По умолчанию его размер равен значению io.DEFAULT\_BUFFER\_SIZE:   
      
    >>> import io  
    io.DEFAULT\_BUFFER\_SIZE  
    8192*
* *encoding – указывает тип кодировки (используется только для текстовых файлов). Значение по умолчанию можно получить, используя метод locale.getpreferredencoding():  
    
  >>> import locale  
  locale.getpreferredencoding()  
  'cp1251'*
* *errors – указывает, как будут обрабатываться ошибки кодирования и декодирования (только для текстового режима):* 
  + *если параметр не указан или указано значение 'strict' – при ошибке вырабатывается исключение ValueError;*
  + *'ignore' – игнорирует ошибку;*
  + *'replace' – вызывает установку маркера (например, "?") в том месте, где произошла ошибка;*
  + *'xmlcharrefreplace' – заменяет соответствующим XML-символом;*
  + *backslashreplace' – заменяет ESC-последовательностью с обратным слэшем;*
* *newline – определяет, как будут обрабатываться символы конца строки:* 
  + *если параметр не указан или указано значение None – осуществляется следующая трансляция символов конца строки: при записи в файл – символ \n преобразуется в форму, специфичную для платформы (\n для Unix и \r\n для Windows), при чтении из файла: из формы, специфичной для платформы в символ \n;*
  + *если указан параметр '', трансляция символов конца строки не выполняется;*
* *closefd – определяет, будет ли закрыт дескриптор файла после закрытия файла:* 
  + *если closefd = True (значение по умолчанию) – будет;*
  + *если closefd = False – нет;*
* *opener – указывает объект ABC-класса callable (например, функцию), который может быть использован для открытия файла.*

Тип файла устанавливается при его открытии в соответствии с указанными значениями аргументов *mode* и *buffering* функции *open()*:

* *текстовый файл (класс \_io.TextIOWrapper) – все режимы, в которых аргумент mode не содержит признак 'b' (binary);*
* *двоичный файл с буферизацией для записи (класс \_io.BufferedWriter) – mode='wb' или mode='ab' и buffering<>0;*
* *двоичный файл с буферизацией для чтения (класс \_io.BufferedReader) – mode='rb' и buffering<>0;*
* *двоичный файл с буферизацией для записи и чтения (класс \_io.BufferedRandom) – все режимы, в которых параметр mode содержит признаки 'b+' и buffering<>0;*
* *двоичный файл без буферизации (класс \_io.FileIO) – все режимы, в которых аргумент mode содержит признак 'b' и buffering=0.*

Текстовые файлы отличаются тем, что содержат строки, осуществляют кодирование и декодирование символов и обработку символов новой строки.  
Пример открытия текстового файла для записи:  
  
>>> f= open ( 'text.txt' , 'w' )  
  
Указав ссылку на файл, можно определить его параметры:  
  
>>> f  
< \_io.TextIOWrapper name='text.txt' mode='w' encoding='cp1251'> ,  
  
т.е. это:

* *текстовый файл класса io.TextIOWrapper;*
* *режим работы – запись;*
* *кодировка символов – 'cp1251'.*

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***1.2 Методы работы с файлами***

После создания текстового файла для работы с ним могут быть использованы следующие методы:

* *write(строка) пишет в файл, открытый для записи, указанную строку, возвращая число записанных символов:  
    
  >>> f.write( 'First line\n' )  
  11  
  >>> f.write( 'Second line\n' )  
  12 .  
    
  Записывать в текстовый файл можно только данные класса строки. Попытка записать в этот файл данные другого типа, например, последовательности байтов (см. описание класса bytes в* [*подразделе 3.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.1) *лаб. раб. №4) приведет к исключению:  
    
  >>> b=b'0123456789abcdef' # тип bytes  
  >>> f.write(b)  
  TypeError: must be str, not bytes*
* *read([число\_байт]) – читает в одну строку из файла, открытого для чтения, число байтов, указанное в качестве необязательного аргумента. Если метод вызывается без аргументов или значением аргумента является -1, то читается весь файл. Если файл уже был прочитан до конца (встретился символ EOF), то метод read() возвращает пустую строку:  
    
  >>> f= open ( 'text.txt' )  
  >>> f.read()  
  'First line\nSecond line\n'  
  >>> f= open ( 'text.txt' )  
  >>> f.read(15)  
  'First line\nSeco' ;*
* *readline() – читает из файла, открытого для чтения, строку:  
    
  >>> f= open ( 'text.txt' )  
  >>> f.readline()  
  'First line\n'  
  >>> f.readline()  
  'Second line\n'  
  >>> f.readline()  
  '' ;*
* *readlines([число\_байт]) – читает из файла, открытого для чтения, все строки файла, возвращая список строк. Если указан необязательный параметр, то читается заданное число байт, затем дополнительно читаются байты до конца текущей строки (т.е. строки не обрываются):  
    
  >>> f= open ( 'text.txt' )  
  >>> f.readlines()  
  ['First line\n', 'Second line\n']  
  >>> f= open ( 'text.txt' )  
  >>> f.readlines(5)  
  ['First line\n']  
    
  Текстовый файл можно прочитать в цикле построчно:  
    
  >>> f= open ( 'text.txt' )  
  >>> for line in f:  
  print (line)  
  First line  
  Second line ;*
* *writelines(список) – записывает в файл строки, заданные в виде списка:  
    
  >>> dig=[ '1' , '2' , '3' ]   
  >>> f= open ( 'text.txt' , 'a' )  
  >>> f.writelines(dig)  
  >>> f= open ('text.txt')  
  >>> for line in f:  
  print (line)  
  First line  
  Second line  
  123 ;*
* *close() – закрывает файл, перенося все сделанные в нем изменения на диск, возвращая ресурсы операционной системе. Необходимо закрывать все файлы, которые больше не используются, т.к. не закрытый файл может привести к потере данных. После закрытия обращение к файлу автоматически вызывает ошибку:  
    
  >>> f.close()  
  >>> f.writelines(dig)  
  ValueError: I/O operation on closed file  
  >>> f= open ( 'text.txt' )  
  >>> f.readlines()  
  ['First line\n', 'Second line\n', '123']*
* *readable() – возвращает значение True, если файл открыт для чтения, и False в противном случае:  
    
  >>> f= open ( 'text.txt' )  
  >>> f.readable()  
  True ;*
* *writable() – возвращает значение True, если файл открыт для записи, и False в противном случае:  
    
  >>> f.writable()  
  False ;*
* *seek(на\_сколько\_байт[, откуда]) – перемещает указатель текущего положения файла на заданное количество байт от позиции, указанной вторым аргументом:* 
  + *0 – начало файла (значение по умолчанию);*
  + *1 – текущая позиция в файле;*
  + *2 – конец файла;*

*При этом, число байт для перемещения может быть как положительным (вперед), так и отрицательным (назад). Для проверки этого метода создаем двоичный файл для записи и чтения:  
  
>>> f= open ( 'text2.txt' , 'w+b' )  
  
Записываем в него последовательность байтов из переменной b (b'0123456789abcdef'):  
  
>>> f.write(b)  
16,  
  
перемещаем указатель файла на 5 байтов вперед (относительно начала файла):  
  
>>> f.seek(5)  
5  
  
и читаем один байт из файла:  
  
>>> f.read(1)  
b'5'*

* *tell() – возвращает значение текущей позиции в файле:  
    
  >>> f.tell()  
  6  
    
  Перемещаем указатель файла на 3 позиции относительного текущего положения:  
    
  >>> f.seek(3,1)  
  9  
    
  и читаем 4 байта:  
    
  >>> f.read(4)  
  b'9abc'  
    
  И наконец, перемещаем указатель файла на 9 позиций назад относительно конца файла:  
    
  >>> f.seek(-9,2)  
  7  
  и читаем байты из этой позиции до конца файла:  
    
  >>> f.read()  
  b'789abcdef'*
* *seekable – возвращает значение True, если объект поддерживает метод seek(), и False в противном случае:  
    
  >>> f.seekable()  
  True*
* *truncate([размер]) – урезает файл до указанного размера, если аргумент не задан – до указателя текущего положения. Метод не работает для файла, открытого в режиме "только чтение":   
    
  >>> f.seek(8)  
  8  
  >>> f.truncate()  
  8  
  >>> f.read()  
  b'01234567'  
  >>> f.truncate(5)  
  5  
  >>> f.read()  
  b'01234'*

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***1.3 Передача содержимого скрипта клиенту***

Работа с текстовыми файлами на языке Python показана в небольшом клиент-серверном приложении, в котором сервер предоставляет пользователю тексты запрашиваемых скриптов.   
Клиентская часть реализована в виде [примера №1](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#prim1), в котором HTML-форма содержит поле ввода имени файла ("filename") и кнопки "Сброс" и "Получить файл":

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

[*Пример №1*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/prim1.htm)  
  
*<!DOCTYPE html>*   
*<html>  
<head>  
<title>Передача файла клиенту</title>  
<link rel="stylesheet" href="file:///D:/!PYTHON/lab.css">  
</head>  
<body>  
<h1>Передача файла клиенту</h1>  
<form method="get" action="http://zykov/PYTHON/get\_file\_form.py">  
Имя файла:  
<input type="text" class="form" name="filename"><br><br>  
<input type="reset" value="Сброс" class="kn">&nbsp;&nbsp;  
<input type="submit" value="Получить файл" class="kn">  
</form>  
</body>  
</html>*

После ввода пользователем имени требуемого файла и нажатия кнопки "Получить файл" браузер передает введенное имя файла серверному приложению *get\_file\_form.py*:   
  
#!c: . . . /Python35-32/python.exe  
print ( "Content-Type: text/html\n" )  
import cgi, cgitb, my\_cgi  
file=cgi.parse()[ "filename" ][0]  
try :  
f= open (file, encoding= "utf-8" )  
print (my\_cgi.H, "CGI: передача содержимого скрипта " ,file)  
for el in f: my\_cgi.D,el),  
except FileNotFoundError : print (my\_cgi.D, "Файл " , file, " не найден" ),  
  
которое:

* *с помощью метода cgi.parse() получает имя файла, содержащего требуемый скрипт;*
* *пробует с использованием оператора try/except (см.* [*подраздел 2.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#2.2) *) открыть этот файл для чтения;*
* *если файл с таким именем не найден – возникает исключение "FileNotFoundError", которое обрабатывается секцией except – клиенту сообщается, что такой файл не найден. На этом выполнение скрипта завершается;*
* *если файл найден, открывает его и передает клиенту сообщение "CGI: передача файла" с указанием имени файла;*
* *построчно считывает и передает клиенту содержимое файла.*

Если указать имя скрипта, который осуществляет приведенные выше действия, т.е. серверного приложения *get\_file\_form.py*, то клиент получит следующую Web-страницу, содержащую текст требуемого скрипта:

**CGI: передача содержимого скрипта get\_file\_form.py**

#!c:/ . . . Python35-32/python.exe  
  
print("Content-Type: text/html\n")  
  
import cgi, cgitb, my\_cgi   
  
file=cgi.parse()["filename"][0]   
  
try :   
  
f= open (file, encoding= "utf-8" )   
  
print (my\_cgi.H, "CGI: передача содержимого скрипта " ,file)   
  
for el in f: my\_cgi.D,el)   
  
except FileNotFoundError : print (my\_cgi.D, "Файл " , file, " не найден" ).

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***1.4 Использование модуля os для работы с файловой системой***

Модуль *os* имеет ряд методов, которые можно использовать для работы с файлами:

* *rename(имя\_файла, новое\_имя\_файла) – переименовует файл:  
    
  >>> f= open ( 'text2.txt' )  
  >>> f.readlines()  
  'First line\nSecond line\nThrid line\n'  
  >>> f= open ( 'text3.txt' )  
  FileNotFoundError: [Errno 2] No such file or directory: 'text3.txt' :  
    
  Фрагмент кода показывает, что файл text2.txt содержит данные, а файла text3.txt нет. Теперь с помощью метода rename() модуля os переименуем файл text2.txt в файл text3.txt  
    
  >>> f.close()  
  >>> import os  
  >>> os.rename( 'text2.txt' , 'text3.txt' )  
    
  и попытается открыть файлы text2.txt и text3.txt и прочитать из них данные:  
    
  >>> f= open ( 'text2.txt' )  
  FileNotFoundError: [Errno 2] No such file or directory: 'text2.txt'   
  >>> open ( 'text3.txt' )  
  >>> f.readlines()  
  'First line\nSecond line\nThrid line\n'  
  >>> f.close()  
    
  Результаты подтверждают правильность работы функции rename() по переименованию файлов;*
* *remove(имя\_файла) – удаляет указанный файл:  
    
  >>> os.remove( 'text3.txt' )  
  >>> open ( 'text3.txt' )  
  FileNotFoundError: [Errno 2] No such file or directory: 'text3.txt' ;*
* *getcwd() – возвращает путь к текущему каталогу, например:  
    
  >>> os.getcwd()  
  'F:\\DISK\_D\\PYTHON\\LAB' ;*
* *chdir(путь) – сделать каталог, указанный аргументом, текущим каталогом, например:  
    
  >>> os.chdir('F:\\DISK\_D\\PYTHON')  
  >>> os.getcwd()  
  'F:\\DISK\_D\\PYTHON' ;*
* *listdir(путь) – возвращает в виде списка содержимое указанного каталога (если путь не задан – текущего):  
    
  >>> os.listdir()  
  ['LAB', 'LAB.rar', 'python', 'Python\_book', 'SAMPLES', 'THEORY', 'ТЕОРИЯ'] ;*
* *mkdir(имя\_нового\_каталога) – создает новый каталог:  
    
  >>> os.chdir('TEMP')  
  >>> os.listdir()  
  ['LAB', 'LAB.rar', 'python', 'Python\_book', 'SAMPLES', 'TEMP', 'THEORY', 'ТЕОРИЯ'] ;*
* *rmdir(путь) – удаляет указанный каталог:  
    
  >>> os.chdir('TEMP')  
  >>> os.listdir()  
  ['LAB', 'LAB.rar', 'python', 'Python\_book', 'SAMPLES', 'THEORY', 'ТЕОРИЯ'] ;*

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***1.5 Использование модуля pickle для сериализации и консервации объектов***

Файлы в языке Python могут содержать или строки (текстовые файлы), или последовательности байтов (двоичные файлы). Чтобы сохранить в файле данные других типов, необходимо преобразовать их в строки (класс str) или в последовательности байтов (классы bytes и bytesarray). Это называется сериализацией объектов.  
Модуль *pickle* содержит методы, позволяющие выполнить преобразование объектов разных типов и запись их в файлы:

* *dumps(объект) – преобразует объект в последовательность байтов типа bytes:  
    
  >>> a\_dict={ 'name' : 'Tom' , 'sport' : 'tennis' }  
  >>> import pickle  
  >>> s\_dump=pickle.dumps(a\_dict)  
  >>> s\_dump  
  b'\x80\x03}q\x00(X\x05\x00\x00\x00sportq\x01X\x06\x00\x00\x00tennisq\x02X\x04\x00\x00\x00name q\x03X\x03\x00\x00\x00Tomq\x04u.'*
* *loads(объект bytes) – выполняет обратную операцию – возвращает исходную форму сериализованного методом dumps() объекта:  
    
  >>> b\_dict=pickle.loads(s\_dump)  
  >>> b\_dict  
  {'sport': 'tennis', 'name': 'Tom'}*
* *dump(объект, ссылка\_на\_файл) – преобразует объект в последовательность байтов и в таком виде записывает в двоичный файл, т.е. выполняет операции сериализации и консервации объекта:   
    
  >>> f2= open ( 'dump1.dat' , 'w+b' )  
  >>> pickle.dump(a\_dict,f2)  
  >>> f2.seek(0)  
  0*
* *load(ссылка\_на\_файл) – читает из файла данные, преобразует их в исходную форму и возращает:  
    
  >>> c\_dict=pickle.load(f2)  
  >>> c\_dict  
  {'sport': 'tennis', 'name': 'Tom'}*

Методы консервации *pickle.dump()* и деконсервации *pickle.load()* данных были использованы для сохранения и чтения набора тестов и словаря данных пользоваетеля при его тестировании (см. подразделы [2.2.2](https://py-khpi.github.io/!nfo/doc/lab/LAB10/lab.htm#2.2.2) и [2.2.3](https://py-khpi.github.io/!nfo/doc/lab/LAB10/lab.htm#2.2.3) лаб. раб. №10).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***2 Обработка исключений***

В программах на языке Python могут возникнуть ошибки двух типов:

* *синтаксические ошибки, которые возникают при грамматическом разборе кода (парсинге от англ. parsing);*
* *исключительные ситуации или просто иключения, которые возникают при работе кода.*

Если синтаксические ошибки могут обрабатываться только интерпретатором, то исключения могут быть обработаны также пользователем.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***2.1 Типы исключений***

Python имеет следующие встроенные типы исключений:

* *BaseException – базовый класс для всех встроенных исключений;*
* *Exception – все встроенные исключения, а также все пользовательские исключения должны наследоваться из этого класса;*
* *SystemExit – исключение, порождаемое функцией sys.exit при выходе из программы;*
* *SystemExit – исключение, порождаемое функцией sys.exit при выходе из программы.*
* *KeyboardInterrupt – возникает при прерывании программы пользователем (обычно комбинацией клавиш Ctrl+C;*
* *GeneratorExit – возникает при вызове метода close объекта generator;*
* *StopIteration – возникает встроенной функцией next, если в итераторе больше нет элементов;*
* *ArithmeticError – арифметическая ошибка;*
* *FloatingPointError – возникает при неудачном выполнении операции с плавающей запятой;*
* *OverflowError – возникает, когда результат арифметической операции слишком велик для представления. Не появляется при обычной работе с числами (так как Python поддерживает длинные числа), но может возникать в некоторых других случаях;*
* *ZeroDivisionError – деление на ноль;*
* *AssertionError – выражение в функции assert ложно;*
* *AttributeError – объект не имеет данного атрибута (значения или метода)*
* *BufferError – операция, связанная с буфером, не может быть выполнена;*
* *EOFError – функция input() наткнулась на конец файла и не смогла при этом ничего прочитать;*
* *ImportError – не удалось импортирование модуля или его атрибута;*
* *LookupError – некорректный индекс или ключ;*
* *IndexError – индекс не входит в диапазон элементов;*
* *KeyError – несуществующий ключ (в словаре, множестве или другом объекте);*
* *MemoryError – недостаточно памяти;*
* *NameError – не найдено переменной с таким именем;*
* *UnboundLocalError – сделана ссылка на локальную переменную в функции, но переменная не определена ранее;*
* *OSError – ошибка, связанная с системой;*
* *io.BlockingIOError – возникает, когда используется блоковая операция ввода/вывода при неблокированном потоковом вводе/выводе;*
* *ChildProcessError – неудача при операции с дочерним процессом;*
* *ConnectionError – базовый класс для исключений, связанных с подключениями;*
* *BrokenPipeError – возникает, когда запись в контейнер невозможна;*
* *ConnectionAbortedError – возникает, когда подключение прекращино устройством;*
* *ConnectionRefusedError – возникает, когда попытка подключиться отвергнута устройством;*
* *ConnectionResetError – возникает, когда попытка подключиться сброшена устройством;*
* *FileExistsError – попытка создания файла или директории, которые уже существуют;*
* *InterruptedError – системный вызов прерван входящим сигналом;*
* *IsADirectoryError – ожидался файл, а это директория;*
* *NotADirectoryError – ожидалась директория, а это файл;*
* *PermissionError – не достаточно прав доступа;*
* *ProcessLookupError – указанного процесса не существует;*
* *TimeoutError – возникает, когда заканчивается интервал времени работы системной функции;*
* *ReferenceError – попытка доступа к атрибуту со слабой ссылкой;*
* *RuntimeError – возникает, когда исключение не попадает ни под одну из других категорий;*
* *SyntaxError – синтаксическая ошибка;*
* *– неправильные отступы;*
* *TabError – смешивание в отступах табуляции и пробелов;*
* *SystemError – внутренняя ошибка;*
* *ValueError – функция получает аргумент правильного типа, но некорректного значения;*
* *UnicodeError – ошибка, связанная с кодированием / раскодированием Unicode в строках;*
* *UnicodeEncodeError – исключение, связанное с кодированием Unicode;*
* *UnicodeDecodeError – исключение, связанное с декодированием Unicode;*
* *UnicodeTranslateError – исключение, связанное с переводом Unicode;*
* *Warning – предупреждение.*

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***2.2 Использование оператора try***

Для обработки исключений используется оператор *try*, который в общем случае имеет следующую структуру:  
  
try:  
<блок>  
except <тип исключения>:  
<блок>  
[else:  
<блок>]  
[finally:  
<блок>]  
  
Отдельные части (секции) оператора *try* имеют следующее назначение:

* *в секции try указывается фрагмент кода, подлежащий проверке на правильность выполнения. Если во время выполнения возникает исключение, выполнение передается секции except и оставшаяся часть кода секции try не выполняется. Если исключение не возникает, управление секции except вообще не передается и эта секция не выполняется;*
* *в секции except задается тип обрабатываемого исключения одним из следующих способов:* 
  + *если тип исключения не указан – будет обрабатываться любое из возникших в секции try исключений;*
  + *если указан тип одного исключения – будет обрабатываться именно это исключение. Если возникнет любое другое исключение, оно будет передано на обработку вышестоящему оператору try. Если ни один из операторов программы не обрабатывает возникшее исключение, его обработает интерпретатор стандартным образом;*
  + *если указано в виде кортежа несколько типов исключений – будет обрабатываться любое из них;*
* *в отличие от секций try и except секция else не является обязательной. Она выполняется тогда, когда в секции try не возникло исключений;*
* *секция finally, если указана в операторе try, выполняется всегда после завершения работы секции try, независимо от того, произошло исключение, или нет. Используется в случаях, когда нужно выполнить определенные действия всегда, например, при операциях с файлами в любом случае необходимо закрыть файл.*

В операторе *try* может быть указано несколько секций *except*, что позволяет обрабатывать несколько исключений независимо.  
Приведем пример, в котором вводятся числа с клавиатуры и вычисляются их обратные значения. Поскольку при ошибке ввода или по незнанию может быть введено число 0 и возникнуть исключение деления на 0 (*ZeroDivisionError*), то для обработки этой ситуации используется оператор *try*:  
  
>>> while True :  
try :  
c=1/int(input ( 'Введите число ' ))  
print('c= ',c)  
break   
except ZeroDivisionError:  
print ( 'Ошибка: деление на 0' )  
Введите число 0  
Ошибка: деление на 0  
Введите число 5  
c=0.2  
  
После ввода нуля возникает и обрабатывается исключение *ZeroDivisionError*, сообщая пользователю об ошибке ввода. При повторном вводе числа на экран выводится его обратное значение – 0.2.   
Использование оператора *try* для обнаружения и обработки исключения *FileNotFoundError* при открытии файла показано в [подразделе 1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.3) .

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***2.3 Использование оператора raise***

В языке Python имеется возможность вырабатывать исключения программно (в основном при отладке скриптов). Это делается с помощью оператора *raise*. Пвриведем пример его использования для генерации ислючения *ZeroDivisionError*, изменив предыдущий пример:  
  
>>> try :  
raise ZeroDivisionError  
except ZeroDivisionError :  
print ( 'Ошибка: деление на 0' )  
Ошибка: деление на 0

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***2.4 Использование функции/оператора assert***

Для проверки правильности выполнения отдельных выражений может быть также использована функция *assert()* (для аналогичных целей может быть также применен оператор *assert*). Аргументом этой функции является выражение на языке Python, которое выражает некоторое условие. Функция проверяет это условие и, если оно не выполняется, генерирует исключение *AssertionError*.   
В качестве примера использования функции *assert* рассмотрим тот же случай ввода чисел с клавиатуры и их анализ. Только на этот раз, чтобы соответствовать требованиям их дальнейшей обработки, числа должны быть больше нуля:  
  
>>> try :  
c= int ( input ( 'Введите число ' ))  
assert (c>=0)  
print ( 'c= ' ,c)  
except AssertionError :  
print ( 'Ошибка: введено отрицательное число' )  
Введите число -4  
Ошибка: введено отрицательное число

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#beg)

***Индивидуальные задания***

Разработать приложение на базе технологии "клиент-сервер".   
В задачи клиентской части входит: разработать HTML-документ, содержащий форму, обязательными элементами которой являются:

* *три текстовых поля для задания данных, выбранных согласно колонке "Данные" табл. 1:* 
  + *1 – Фамилия, инициалы;*
  + *2 – Наименование университета;*
  + *3 – Наименование факультета;*
  + *4 – Наименование кафедры;*
  + *5 – Наименование специальности;*
  + *6 – Курс;*
  + *7 – Наименование группы;*
  + *8 – Телефон;*
  + *9 – Мобильный телефон;*
  + *10 – Электронная почта;*
* *кнопка для передачи данных на сервер.*

В задачи серверной части входит:

* *принять данные из формы;*
* *создать каталог "CGI" и сделать его текущим;*
* *записать имена и значения полученных данных соответственно в файлы "names.txt" и "values.txt" с помощью методов, заданных колонкой "Методы"/"Запись" табл. 1 (первая цифра – для файла "names.txt", вторая – для файла "values.txt"):* 
  + *1 – write();*
  + *2 – writelines();*
* *определить длину файлов "names.txt" и "values.txt";*
* *прочитать и передать клиенту содержимое файла "values.txt" с помощью метода, заданного колонкой "Методы"/"Чтение" табл. 1:* 
  + *1 – read();*
  + *2 – readline();*
  + *3 – readlines();*
* *Создать с помощью встроенных функций bytes() range() (см. разделы* [*3*](file:///D:\!PYTHON\LAB4\lab.htm#3) *и* [*2*](file:///D:\!PYTHON\LAB4\lab.htm#2) *) лаб. раб. №4) последовательность чисел длиной, заданной колонкой "Длина" табл. 1 и записать ее в двоичный файл "binary\_data.dat";*
* *Прочитать значение байта файла "binary\_data.dat", номер которого задан колонкой "Номер" табл. 1;*
* *Переместить указатель файла соответственно значениям колонок "Смещение" и "Откуда" табл. 1 и прочитать значения трех байтов;*
* *Проверить с использованием оператора try и встроенной функции assert(), выполняется ли условие, заданное колонкой "Условие" табл. 1, для целого числа, введенного функцией input() (заканчивается ли последовательность чисел в файле на цифру, указанную в колонке «Условие»)*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 1 – Параметры индивидуальных заданий | | | | | | | | |
| **Номер п/п** | **Данные** | **Методы** | | **Длина** | **Номер** | **Смещение** | **Откуда** | **Условие** |
| **записи** | **чтения** |
| 1 | 1, 2, 6 | 1 | 3 | 20 | 11 | 17 | 0 | 1 |
| 2 | 1, 2, 7 | 2 | 2 | 16 | 5 | 7 | 1 | 2 |
| 3 | 1, 2, 8 | 1 | 1 | 25 | 20 | -4 | 2 | 3 |
| 4 | 1, 2, 9 | 2 | 3 | 10 | 4 | 6 | 0 | 4 |
| 5 | 1, 2, 10 | 1 | 2 | 12 | 6 | -1 | 1 | 5 |
| 6 | 1, 3, 6 | 2 | 1 | 18 | 14 | -10 | 2 | 1 |
| 7 | 1, 3, 7 | 1 | 3 | 15 | 13 | 6 | 0 | 2 |
| 8 | 1, 3, 8 | 2 | 2 | 24 | 12 | 8 | 1 | 3 |
| 9 | 1, 3, 9 | 1 | 1 | 8 | 2 | -2 | 2 | 4 |
| 10 | 1, 3, 10 | 2 | 3 | 14 | 8 | 10 | 0 | 5 |
| 11 | 1, 4, 6 | 1 | 2 | 26 | 20 | -12 | 1 | 1 |
| 12 | 1, 4, 7 | 2 | 1 | 9 | 8 | -5 | 2 | 2 |
| 13 | 1, 4, 8 | 1 | 3 | 19 | 17 | 14 | 0 | 3 |
| 14 | 1, 4, 9 | 2 | 2 | 13 | 9 | 3 | 1 | 4 |
| 15 | 1, 4, 10 | 1 | 1 | 30 | 15 | -20 | 2 | 5 |
| 16 | 1, 5, 6 | 2 | 3 | 11 | 1 | 10 | 0 | 1 |
| 17 | 1, 5, 7 | 1 | 2 | 17 | 5 | -9 | 1 | 2 |
| 18 | 1, 5, 8 | 2 | 1 | 21 | 20 | -20 | 2 | 3 |