Лекция 5

Обработка исключений

Исключения - это извещения интерпретатора, возбуждаемые в случае возникновения ошибки в программном коде или при наступлении какого-либо события. Если в коде не предусмотрена обработка исключения, то программа прерывается и выводится сообщение об ошибке.

Существуют три типа ошибок в программе:

- синтаксические - это ошибки в имени оператора или функции, отсутствие закрывающей или открывающей кавычек и т. д., т. е. ошибки в синтаксисе языка. Как правило, интерпретатор предупредит о наличии ошибки, а программа не будет выполняться совсем. Пример синтаксической ошибки:

In [1]:
```
print("Нет  завершающей  кавычки!)
```

```
  File "<ipython-input-1-f572aab57a7b>", line 1
    print("Нет  завершающей  кавычки!)
                                       ^
SyntaxError: EOL while scanning string literal
```

- логические - это ошибки в логике работы программы, которые можно выявить только по результатам работы скрипта. Как правило, интерпретатор не предупреждает о наличии ошибки. А программа будет выполняться, т. к. не содержит синтаксических ошибок. Такие ошибки достаточно трудно выявить и исправить;
- ошибки времени выполнепия - это ошибки, которые возникают во время работы скрипта. Причиной являются события, не предусмотренные программистом. Классическим примером служит деление на ноль

In [2]:
```
print(10/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-14fd4c3343b4> in <module>()
----> 1 print(10/0)

ZeroDivisionError: division by zero
```

Необходимо заметить, что в языке Python исключения возбуждаются не только при ошибке, но и как уведомление о наступлении каких-либо событий. Например, метод index () возбуждает исключение ValueError, если искомый фрагмент не входит в строку:

In [3]:
```
"Строка".index("текст")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-0a329fdc99ca> in <module>()
----> 1 "Строка".index("текст")

ValueError: substring not found
```

Инструкция try ... except ... else ... finally

Для обработки исключений предназначена инструкция try. Формат инструкции:

```
try:
    <Блок, в котором перехватьшаются исключения>
[ except [ <Исключение1> [ as <Объект исключения>] ] :
    <Блок, вьтолняемьм при возникновении исключения>
[ ...
except [<ИсключениеN>[ as <Объект исключения>]]:
    <Блок, вьтолняемьм при возникновении исключения>]]
[else:
    <Блок, вьтолняемьм, если исключение не возникло>]
[finally:
    <Блок, вьтолняемьм в любом случае>]
```

Инструкции, в которых перехватываются исключения, должны быть расположены внутри блока try. В блоке except в параметре <Исключение1> указывается класс обрабатываемого исключения. Например, обработать исключение, возникающее при делении на ноль, можно так:

In [4]:
```
x = 0
try: # Перехватыаем исключения
    x = 1/0 # Ошибка: деление на 0
except ZeroDivisionError: # Указыаем класс исключения
    print("Обработали деление на 0")
    # x = 0
print(x)
```

```
Обработали деление на 0
0
```

Если в блоке try возникло исключение, то управление передается блоку except. В случае, если исключение не соответствует указанному классу, управление передается следующему блоку except. Если ни один блок except не соответствует исключению, то исключение "всплывает" к обработчику более высокого уровня. Если исключение нигде не обрабатывается в программе, то управление передается обработчику по умолчанию, который останавливает выполнение программы и выводит стандартную информацию об ошибке. Таким образом, в обработчике может быть несколько блоков except с разными классами исключений.

In [5]:
```
x = 0
try: # Обрабатыаем исключения
    try: # Вложенным обработчик
        x = 1/0 # Ошибка: деление на 0
    except NameError:
        print("Неопределенный идентификатор")
    except IndexError:
        print("Несуществующий индекс")
    print("Выражение после вложенного обработчика")
except  ZeroDivisionError:
    print("Обработка деления на 0")
    x = 0
print(x) # Выведет: 0
```

```
Обработка деления на 0
0
```

В инструкции except можно указать сразу несколько исключений, перечислив их через запятую внутри круглых скобок:

In [6]:
```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except (NameError, IndexError, ZeroDivisionError):
    x = 0
print(x) # Выведет: 0
```

```
0
```

Получить информацию об обрабатываемом исключении можно через второй параметр в инструкции except:

In [7]:
```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except (NameError, IndexError, ZeroDivisionError) as err:
    print(err.__class__.__name__) #Название класса исключения
    print(err) # Текст сообщения об ошибке
```

```
ZeroDivisionError
division by zero
```

Для получения информации об исключении можно воспользоваться функцией exc_info() из модуля sys, которая возвращает кортеж из трех элементов: типа исключения, значения и объекта с трассировочной информацией. Преобразовать эти значения в удобочитаемый вид позволяет модуль traceback.

In [8]:
```
import sys, traceback
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except ZeroDivisionError as err:
    etype, value, trace = sys.exc_info()
    print("Type: {}, Value: {}, Trace: {} ".format(etype, value, trace))
    print("\n", "print_exception() ".center(40,  "-"))
    traceback.print_exception(etype, value, trace, limit=5, file=sys.stdout)
    print("\n",  "print_tb()".center(40,  "-"))
    traceback.print_tb(trace, limit=1, file=sys.stdout)
    print("\n",  "format_exception()".center(40,  "-"))
    print(traceback.format_exception(etype, value, trace, limit=5))
    print("\n",  "format_exception_only()".center(40,"-"))
    print(traceback.format_exception_only(etype, value))
```

```
Type: <class 'ZeroDivisionError'>, Value: division by zero, Trace: <traceback object at 0x000000A73CA4
A388>

 -----------print_exception() -----------
Traceback (most recent call last):
  File "<ipython-input-8-bdbf3e19a700>", line 4, in <module>
    x = 1/0 # Ошибка: деление на 0
ZeroDivisionError: division by zero

 ---------------print_tb()---------------
  File "<ipython-input-8-bdbf3e19a700>", line 4, in <module>
    x = 1/0 # Ошибка: деление на 0

 -----------format_exception()-----------
['Traceback (most recent call last):\n', '  File "<ipython-input-8-bdbf3e19a700>", line 4, in <module>
\n    x = 1/0 # Ошибка: деление на 0\n', 'ZeroDivisionError: division by zero\n']

 --------format_exception_only()---------
['ZeroDivisionError: division by zero\n']
```

Если в инструкции except не указан класс исключения, то такой блок перехватывает все исключения. На практике следует избегать пустых инструкций except, т. к. можно перехватить исключение, которое является лишь сигналом системе, а не ошибкой.

In [9]:
```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except:
    x = 0
print(x) # Выведет: 0
```

```
0
```

Если в обработчике присутствует блок else, то инструкции внутри этого блока будут выполнены только при отсутствии ошибок. При необходимости выполнить какие-либо завершающие действия вне зависимости от того, возникло исключение или нет, следует воспользоваться блоком finally.

In [10]:
```
x = 0
try: # Обрабатыаем исключения
#     x = 1/0 # Ошибка: деление на 0
    x = 1/10
except ZeroDivisionError:
    print('Деление на 0')
    x = 0
else:
    print('Блок else')
finally:
    print('Блок finally')
print(x) # Выведет: 0
```

```
Блок else
Блок finally
0
```

Необходимо заметить, что при наличии исключения и отсутствии блока except инструкции внутри блока finally будут выполнены, но исключение не будет обработано. Оно продолжит "всплывание" к обработчику более высокого уровня. Если пользовательский обработчик отсутствует, то управление передается обработчику по умолчанию, который прерывает выполнение программы и выводит сообщение об ошибке.

In [11]:
```
x = 0
try: # Обрабатыаем исключения
#     x = 1/0 # Ошибка: деление на 0
    x = 1/10
finally:
    print('Блок finally')
```

```
Блок finally
```

Классы встроенных исключений

- BaseException
  - GeneratorExit
  - Keyboardinterrupt
  - SystemExit
  - Exception
    - Stopiteration
    - Warning
      - BytesWarning, ResourceWarning,
      - DeprecationWarning, FutureWarning, ImportWarning,
      - PendingDeprecationWarning, RuntimeWarning, SyntaxWarning,
      - UnicodeWarning, UserWarning
    - ArithmeticError
      - FloatingPointError, OverflowError, ZeroDivisionError
    - AssertionError
    - AttributeError
    - BufferError
    - EnvironmentError
      - IOError
      - OSError
        - WindowsError
    - EOFError
    - ImportError
    - LookupError
      - IndexError, KeyError
    - MemoryError
    - NameError
      - UnboundLocalError
    - ReferenceError
    - RuntimeError
      - NotimplementedError
    - SyntaxError
      - IndentationError
        - TabError
    - SystemError
    - TypeError
    - ValueError
      - UnicodeError
        - UnicodeDecodeError, UnicodeEncodeError
        - UnicodeTranslateError

Основное преимущество использования классов для обработки исключений заключается в возможности указания базового класса для перехвата всех исключений соответствующих классов-потомков. Например, для перехвата деления на ноль мы использовали класс ZeroDivisionError. Если вместо этого класса указать базовый класс ArithrneticError, то будут перехватываться исключения классов FloatingPointError, OverflowError и ZeroDivisionError.

In [80]:
```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except ArithmeticError: # Указываем базовый класс
    print('Обработка деления на 0')
    x = 0
print(x) # Выведет: 0
```

```
Обработка деления на 0
0
```

Рассмотрим основные классы встроенных исключений:

- baseException - является классом самого верхнего уровня;
- Exception - именно этот класс, а не BaseException, необходимо наследовать при создании пользовательских классов' исключений;
- AssertionError - возбуждается инструкцией assert;
- AttributeError - попытка обращения к несуществующему атрибуту объекта;
- EOFError - возбуждается функцией input() при достижении конца файла;
- IOError - ошибка доступа к файлу;
- ImportError - невозможно подключить модуль или пакет;
- IndentationError - неправильно расставлены отступы в программ е;
- IndexError - указанный индекс не существует в последовательности;
- KeyError - указанный ключ не существует в словаре;
- KeyboardInterrupt - нажата комбинация клавиш +;
- NameError - попытка обращения к идентификатору до его определения;
- Stoprteration - возбуждается методом _ next _ () как сигнал об окончании итераций;
- SyntaxError - синтаксическая ошибка;
- TypeError - тип объекта не соответствует ожидаемому;
- UnboundLocalError - внутри функции переменной присваивается значение после обращения к одноименной глобальной переменной;
- UnicodeDecodeError - ошибка преобразования последовательности байтов в строку;

- UnicodeEncodeError - ошибка преобразования строки в последовательность байтов;
- ValueError - переданный параметр не соответствует ожидаемому значению;
- ZeroDivisionError - попьпка· деления на ноль.

Пользовательские исключения

Инструкция raise возбуждает указанное исключение. Она имеет несколько форматов:

- raise <Экземпляр класса>
- raise <Название класса>
- raise <Экземпляр или название класса> from <Объект исключения>
- raise

В первом формате инструкции raise указывается экземпляр класса возбуждаемого исключения. При создании экземпляра можно передать данные конструктору класса. Эти данные будут доступны через второй параметр в инструкции except.

In [81]:
```
try:
    raise ValueError("Описание исключения")
except ValueError  as  msg:
    print(msg) # Выведет: Описание исключения
```

```
Описание исключения
```

In [12]:
```
try:
    raise ValueError
except ValueError:
    print('Сообщение об ошибке')
```

```
Сообщение об ошибке
```

Инструкция assert

Инструкция assert возбуждает исключение AssertionError, если логическое выражение возвращает значение False. Инструкция имеет следующий формат:

assert <Логическое выражение> [, <Сообщение>]

Инструкция assert эквивалентна следующему коду:

```
if __debug__:
    if not <Логическое  выражение>:
        raise  AssertionError(<Сообщение>)
```

Если при запуске программы используется флаг -o, то переменная __debug__ будет иметь ложное значение. Таким образом можно удалить все инструкции assert из байт-кода.

In [84]:
```
try:
    x = -3
    assert x >= 0, "Сообщение об ошибке"
except AssertionError as err:
    print(err)  # Выдает: Сообщение об ошибке
```

```
Сообщение об ошибке
```

In [18]:
```
def factorial(n):
    """Возвращает Факториал числа n.
    Аргумент n - не отрицательное целое число."""
    assert n >= 0, 'Аргумент n должен быть больше 0!'
    assert n % 1 == 0, 'Аргумент n должен быть целым!'
    f = 1
    for i in range(2, n+1):
        f *= i
    return f
```

In [19]:
```
factorial(-1)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-19-456c1b35b1cf> in <module>()
----> 1 factorial(-1)

<ipython-input-18-8ec99d38a0cf> in factorial(n)
      2     """Возвращает Факториал числа n.
      3     Аргумент n - не отрицательное целое число."""
----> 4     assert n >= 0, 'Аргумент n должен быть больше 0!'
      5     assert n % 1 == 0, 'Аргумент n должен быть целым!'
      6     f = 1

AssertionError: Аргумент n должен быть больше 0!
```

In [20]:
```
factorial(5.5)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-20-347bc4de69b9> in <module>()
----> 1 factorial(5.5)

<ipython-input-18-8ec99d38a0cf> in factorial(n)
      3     Аргумент n - не отрицательное целое число."""
      4     assert n >= 0, 'Аргумент n должен быть больше 0!'
----> 5     assert n % 1 == 0, 'Аргумент n должен быть целым!'
      6     f = 1
      7     for i in range(2, n+1):

AssertionError: Аргумент n должен быть целым!
```

In [21]:
```
factorial(0)
```

Out[21]: 1

In [23]:
```
factorial(1)
```

Out[23]: 1

In [24]:
```
factorial(2)
```

Out[24]: 2

In [25]:
```
factorial(5)
```

Out[25]: 120

Работа с файлами

Открытие файла

Прежде чем работать с файлом, необходимо создать объект файла с помощью функции open(). Функция имеет следующий формат:

open(<Путь к файлу>[, mode='r'] [, buffering=-1] [, encoding=None] [,errors=None] [, newline=None] [, closefd=True])

В первом параметре указывается путь к файлу. Путь может быть абсолютным или относительным. В Windows следует учитывать, что слэш является специальным символом. По этой причине слэш необходимо удваивать или вместо обычных строк использовать неформатированные строки.

In [33]:
```
# Правильно
"C:\\temp\\new\\file.txt"
```

Out[33]: 'C:\\temp\\new\\file.txt'

In [34]:
```
# Правильно
r"C:\temp\new\file.txt"
```

Out[34]: 'C:\\temp\\new\\file.txt'

In [32]:
```
# Неправильно!
# В этом пути присутствуют сразу три специальных символа: \t, \n и \f. После преобразования специальных символов пут
# будет выглядеть следующим образом: C:<Табуляция>emp<Перевод строки>ew<Перевод формата>ile.txt
"C:\temp\new\file.txt"
```

Out[32]: 'C:\temp\new\x0cile.txt'

Если открываемый файл находится в текущем рабочем каталоге, то можно указать толь ко название файла. Пример:

In [35]:
```
ls
```

```
 '®¬ ў гбва®©бвўЃ Е Ё¬ЃЃв ¬ЃвЄг Data
 'ЃаЁ©л© ®¬Ґа в®¬ : 1C8C-28F1

 '®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5

06.11.2016  20:32    <DIR>          .
06.11.2016  20:32    <DIR>          ..
06.11.2016  19:33               616 lec5_v2016s2_v1.ipynb
06.11.2016  20:32           80я779 lec5_v2016s2_v2.ipynb
               2 д ©«®ў          81я395 Ў ©в
               2 Ї Ї®Є    3я263я270я912 Ў ©в бў®Ў®¤®
```

In [36]:
```
f1 = open('new_file.txt', mode='w')
f1.close()
```

In [37]:
```
ls
```

```
 '®¬ ў гбва®©бвўҐ Е Ё¬ҐҐв ¬ҐвЄг Data
 'ҐаЁ©л© ®¬Ґа в®¬ : 1C8C-28F1

 '®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5

06.11.2016  20:51    <DIR>          .
06.11.2016  20:51    <DIR>          ..
06.11.2016  19:33               616 lec5_v2016s2_v1.ipynb
06.11.2016  20:32            80я779 lec5_v2016s2_v2.ipynb
06.11.2016  20:51                 0 new_file.txt
               3 д ©«®ў          81я395 Ў ©в
               2 Ї Ї®Є   3я263я270я912 Ў ©в бў®Ў®¤®
```

Если открываемый файл расположен во вложенной папке, то перед названием файла перечисляются названия вложенных папок через слэш:

In [38]:
```
f2 = open('folder1\\new_file2.txt', mode='w')
f2.close()
```

In [39]:
```
ls
```

```
 '®¬ ў гбва®©бвўҐ Е Ё¬ҐҐв ¬ҐвЄг Data
 'ҐаЁ©л© ®¬Ґа в®¬ : 1C8C-28F1

 '®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5

06.11.2016  20:52    <DIR>          .
06.11.2016  20:52    <DIR>          ..
06.11.2016  20:53    <DIR>          folder1
06.11.2016  19:33               616 lec5_v2016s2_v1.ipynb
06.11.2016  20:32            80я779 lec5_v2016s2_v2.ipynb
06.11.2016  20:51                 0 new_file.txt
               3 д ©«®ў          81я395 Ў ©в
               3 Ї Ї®Є   3я263я270я912 Ў ©в бў®Ў®¤®
```

In [40]:
```
cd folder1
```

```
E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5\folder1
```

In [41]:
```
ls
```

```
 '®¬ ў гбва®©бвўҐ Е Ё¬ҐҐв ¬ҐвЄг Data
 'ҐаЁ©л© ®¬Ґа в®¬ : 1C8C-28F1

 '®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5\folder1

06.11.2016  20:53    <DIR>          .
06.11.2016  20:53    <DIR>          ..
06.11.2016  20:53                 0 new_file2.txt
               1 д ©«®ў             0 Ў ©в
               2 Ї Ї®Є   3я263я270я912 Ў ©в бў®Ў®¤®
```

Если папка с файлом расположена выше уровнем, то перед названием файла указываются две точки и слэш " .. /":

In [42]:
```
f3 = open('..\\new_file3.txt', mode='w')
f3.close()
```

In [43]:
```
cd ..
```

```
E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5
```

In [44]:
```
ls
```

```
 '®¬ ў гбва®©бвўҐ Е Ё¬ҐҐв ¬ҐвЄг Data
 'ҐаЁ©л© ®¬Ґа в®¬ : 1C8C-28F1

 '®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5

06.11.2016  20:55    <DIR>          .
06.11.2016  20:55    <DIR>          ..
06.11.2016  20:53    <DIR>          folder1
06.11.2016  19:33               616 lec5_v2016s2_v1.ipynb
06.11.2016  20:54            88я604 lec5_v2016s2_v2.ipynb
06.11.2016  20:51                 0 new_file.txt
06.11.2016  20:55                 0 new_file3.txt
               4 д ©«®ў          89я220 Ў ©в
               3 Ї Ї®Є   3я274я436я608 Ў ©в бў®Ў®¤®
```

Если путь к файлу начинаетсяя со слеша или с имени диска, то файл находится по абсолютному пути и текущий путь не имеет значения.

Необязательный параметр mode в функции open() может принимать следующие значения:

- r - только чтение (значение по умолчанию). После открытия файла указатель устанавливается на начало файла. Если файл не существует, то возбуждается исключение IOError;
- r+ - чтение и запись. После открытия файла указатель устанавливается на начало файла. Если файл не существует, то возбуждается исключение IOError;
- w - запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла;
- w+ - чтение и запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла;
- a - запись. Если файл не существует, то он будет создан. Запись осуществляется в конец файла. Содержимое файла не удаляется;
- a+ - чтение и запись. Если файл не существует, то он будет создан. Запись осуществляется в конец файла. Содержимое файла не удаляется.

Кроме того, после режима может следовать модификатор:

- b - файл будет открыт в бинарном режиме. Файловые методы принимают и возвращают объекты типа bytes;
- t - файл будет открыт в текстовом режиме (значение по умолчанию в Windows). Файловые методы принимают и возвращают объекты типа str. В этом режиме в Windows при чтении символ \r будет удален, а при записи, наоборот, добавлен.

In [55]:
```
f4 = open('new_file4.txt', mode='w')
```

In [56]:
```
f4.write ( "String1\nString2")
```

Out[56]: 15

In [57]:
```
f4.close()
```

In [58]:
```
ls
```

```
 '®¬ ў гбва®©бвўЃ Е Ё¬ГЃв ¬ГвЄг Data
 'ГаЁ©л© ®¬Ґа в®¬ : 1C8C-28F1

 '®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\fa1_2016\lec_5

06.11.2016  21:04    <DIR>          .
06.11.2016  21:04    <DIR>          ..
06.11.2016  20:53    <DIR>          folder1
06.11.2016  19:33               616 lec5_v2016s2_v1.ipynb
06.11.2016  20:54            88я604 lec5_v2016s2_v2.ipynb
06.11.2016  20:51                 0 new_file.txt
06.11.2016  20:55                 0 new_file3.txt
06.11.2016  21:04                16 new_file4.txt
               5 д ©«®ў          89я236 Ў ©в
               3 Ї Ї®Є    3я274я387я456 Ў ©в бў®Ў®¤®
```

Инструкция with ... as

Python поддерживает протокол менеджеров контекста. Этот протокол гарантирует выполнение завершающих действий (например, закрытие файла) вне зависимости от того, произошло исключение внутри блока кода или нет.

Для работы с протоколом предназначена инструкция with ... as. Инструкция имеет следующий формат:

```
with <Выражение1>[ as <Переменная>] [, ... ,
    <ВыражениеN>[ as <Переменная>]]:
    <Блок, в котором перехватываем исключения>
```

Вначале вычисляется <Выражение1>, которое должно возвращать объект, поддерживающий протокол. Этот объект должен иметь два метода: __enter__() и __exit__(). Метод __enter__() вызывается после создания объекта. Значение, возвращаемое этим методом, присваивается переменной, указанной после ключевого слова as. Далее выполняются инструкции внутри тела инструкции with. Если при выполнении возникло исключение, то управление передается методу __exit__(). Если при выполнении инструкций, расположенных внутри тела инструкции with, исключение не возникло, то управление все равно передается методу __exit__(). В функции __exit__() решаются все задачи освобождения ресурсов, если это необходимо.

Некоторые встроенные объекты по умолчанию поддерживают протокол, например, файлы.

In [1]:
```
with open("new_file4.txt", "rb") as f:
    for line in f:
        print(repr(line))
```

```
b'String1\r\n'
b'String2'
```

Для ускорения работы производится буферизация записываемых данных. Информация из буфера записывается в файл по.rтностью только в момент закрытия файла. В необязательном параметре buffering можно указать размер буфера. Если в качестве значения указан 0, то данные будут сразу записываться в файл (значение допустимо только в бинарном режиме).

Значение 1 используется при построчной записи в файл (значение допустимо только в текстовом режиме), другое положительное число задает примерный размер буфера, а отрицательное значение (или отсутствие значения) означает установку размера, применяемого в системе по умолчанию.

При использовании текстового режима (используется по умолчанию) при чтении производится попытка преобразовать данные в

кодировку Unicode, а при записи выполняется об ратная операция: строка преобразуется в последовательность байтов в определенной кодировке. По умолчанию используется· кодировка, исш;шьзуем'ая в системе. Если преобразова ние невозможно, то возбуждается исключение. Указать кодировку, которая будет использоваться при записи и чтении файла, позволяет параметр encoding. В качестве при мера запишем данные в кодировке UTF-8:

In [62]:
```
with open("new_file5.txt", "w", encoding="utf-8") as f: #указываем кодировку файла
    f.write("Строка") # Записываем строку в файл
```

In [64]:
```
with open("new_file5.txt", "r", encoding="utf-8") as f:
    for line in f:
        print(line)
```

```
Строка
```

Методы для работы с файлами

close() - закрывает файл. Так как интерпретатор автоматически удаляет объект, когда отсутствуют ссылки на него, можно явно не закрывать файл в небольших программах. Тем не менее явное закрытие файла является признаком хорошего стиля программирования. Протокол менеджеров контекста гарантирует закрытие файла вне зависимости от того, произошло исключение внутри блока кода или нет.

write (<Данные>) - записывает строку или последовательность байтов в файл. Если в качестве параметра указана строка, то файл должен быть открыт в текстовом режиме. Для записи последовательности байтов необходимо открыть файл в бинарном режиме.

Помните, что нельзя записывать строку в бинарном режиме и последовательность байтов в текстовом режиме. Метод возвращает количество записанных символов или байтов.

In [66]:
```
# бинарный режим:
with open("new_file6.txt", "wb") as f:
    f.write(bytes("Строка1\nСтрока2", "cp1251"))
    f.write(bytearray("\nСтрока3", "cp1251"))
```

In [68]:
```
bytes("Строка1\nСтрока2", "cp1251")
```

Out[68]:
```
b'\xd1\xf2\xf0\xee\xea\xe01\n\xd1\xf2\xf0\xee\xea\xe02'
```

In [69]:
```
bytes("Строка1\nСтрока2", "utf-8")
```

Out[69]:
```
b'\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb01\n\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb0
2'
```

writelines(<Последовательность>) - записывает последовательность в файл. Если все элементы последовательности являются строками, то файл должен быть открыт в текстовом режиме. Если все элементы являются последовательностями байтов, то файл должен быть открыт в бинарном режиме.

Запись в файл можно призводить при помощи функции print(), передав файл в необязательный атрибут file:

In [86]:
```
with open("new_file8.txt", "w", encoding="cp1251") as f:
    print('Строка1\nСтрока2', file=f)
    print('Строка3', end='', file=f)
    print(' продолжение строки3', file=f)
    print('Строка{} с использованием format'.format(4), file=f)
```

In [87]:
```
with open("new_file8.txt", "r", encoding="cp1251") as f:
    print(f.read())
```

```
Строка1
Строка2
Строка3 продолжение строки3
Строка4 с использованием format
```

In [70]:
```
with open("new_file7.txt", "w", encoding="utf-8") as f:
    f.writelines(["Строка1\n", "Строка2"])
```

read([<Количество>])- считывает данные из файла. Если файл открыт в текстовом режиме, то возвращается строка, а если в бинарном - последовательность байтов. Если параметр не указан, то возвращается содержимое файла от текущей позиции указателя до конца файла:

In [71]:
```
with open("new_file7.txt", "r", encoding="utf-8") as f:
    print(f.read())
```

```
Строка1
Строка2
```

In [72]:
```
with open("new_file7.txt", "r", encoding="utf-8") as f:
    print(f.read(4))
    print(f.read(4))
```

```
Стро
ка1
```

In [73]:
```
with open("new_file7.txt", "rb") as f:
    print(f.read())
```

```
b'\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb01\r\n\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb
02'
```

In [74]:
```
with open("new_file7.txt", "rb") as f:
    print(f.read(4))
```

```
b'\xd0\xa1\xd1\x82'
```

readline ([<Количество>]) -считывает из файла одну строку при каждом вызове. Если файл открыт в текстовом режиме, то возвращается строка, а если в бинарном - последовательность байтов. Возвращаемая строка включает символ перевода строки. Исключением является последняя строка. Если она не завершается символом перевода строки, то символ перевода строки добавлен не будет. При достижении конца файла возвращается пустая строка.

Если в необязательном параметре указано число, то считывание будет выполняться до тех пор, пока не встретится символ новой строки (\n), символ конца файла или из файла не будет прочитано указанное количество символов. Иными словами, если количество символов в строке меньше значения параметра, то будет считана одна строка, а не указанное количество символов. Если количество символов в строке больше, то возвращается указанное количество символов.

In [77]:
```
with open("new_file7.txt", "r", encoding="utf-8") as f:
    print(f.readline())
    print('----')
    print(f.readline())
    print('----')
    print(f.readline())
```

```
Строка1

----
Строка2
----
```

In [78]:
```
with open("new_file7.txt", "rb") as f:
    print(f.readline())
    print('----')
    print(f.readline())
    print('----')
    print(f.readline())
```

```
b'\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb01\r\n'
----
b'\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb02'
----
b''
```

readlines () - считывает все содержимое файла в список. Каждый элемент списка будет содержать одну строку, включая символ перевода строки. Исключением является последняя строка. Если она не завершается символом перевода строки, то символ перевода строки добавлен не будет. Если файл открыт в текстовом режиме, то возвращается список строк, а если в бинарном-список-объектов типа bytes.

In [79]:
```
with open("new_file7.txt", "r", encoding="utf-8") as f:
    lines = f.readlines()
lines
```

Out[79]:
```
['Строка1\n', 'Строка2']
```

__next__() - считывает одну строку при каждом вызове. Если файл открыт в текстовом режиме, то возвращается строка, а если в бинарном - последовательность байтов. При достижении конца файла возбуждается исключение Stopiteration. Пример:

In [80]:
```
with open("new_file7.txt", "r", encoding="utf-8") as f:
    for l in f:
        print(l)
```

```
Строка1

Строка2
```

flush () - записывает данные из буфера на диск;

tell() - возвращает позицию указателя относительно начала файла в виде целого числа. Обратите внимание на то, что в Windows метод tell () считает символ \r как дополнительный байт, хотя этот символ удаляется при открытии файла в текстовом режиме,

чтобы избежать этого несоответствия, следует открывать файл в бинарном режиме, а не в текстовом.

seek (<СМещение> [, <Позиция>]) - устанавливает указатель в позицию, имеющую сме щение <Смещение> относительно позиции <Позиция>. В параметре <Поз~ция> могут быть указаны следующие атрибуты из модуля io или соответствующие им значения:

- io.SEEK_SET или 0 - начало файла (значение по умолчанию);
- io.SEEK_CUR или 1 - текущая позиция указателя;
- io.SEEK_END или 2 - конец файла.

Помимо методов объекты файлов поддерживают несколько атрибутов:

- name - содержит название файла;
- mode - режим, в котором был открыт файл;
- closed - возвращает True, если файл был закрыт, и False в противном случае.
- encoding - название кодировки, которая будет использоваться для преобразования строк перед записью в файл или при чтении. Обратите внимание на то, что изменить значение атрибута нельзя, т. к. атрибут доступен только мя чтения. Атрибут доступен только в текстовом режиме.
- buffer - позволяет получить доступ к буферу. Атрибут доступен только в текстовом режиме. С помощью этого объекта можно записать последовательность байтов в текстовый поток.

Сохранение объектов в файл

Сохранить объекты в файл и в дальнейшем восстановить объекты из файла позволяют модули pickle и shelve.

Модуль pickle предоставляет следующие функции:

- dump(<Объект>, <Файл> [, <Протокол>] [, fix _ imports=True]) - производит сериализацию объекта и записывает данные в указанный файл. В параметре <Файл> указывается файловый объект, открытый на запись в бинарном режиме.
- load() - читает данные из файла и преобразует их в объект. В параметре <Файл> указывается файловый объект. открытый на чтение в бинарном режиме. Формат функции: load (<Файл> [, fix _ imports=True] [, encoding="ASCII"] [, errors="strict"])

In [88]:
```
obj = ["Строка", (12, 3)] # объект для сохранения
```

In [89]:
```
import pickle # подключаем модуль pickle
```

In [90]:
```
with open('obj1.txt', 'wb') as f:
    pickle.dump(obj, f)
```

In [91]:
```
with open('obj1.txt', 'rb') as f:
    obj_l = pickle.load(f)
obj_l
```

Out[91]:
```
['Строка', (12, 3)]
```

В один файл можно сохранить сразу несколько объектов, последовательно вызывая функцию dump().

In [92]:
```
obj2 = (6, 7, 8, 9, 10)
```

In [93]:
```
with open('obj2.txt', 'wb') as f:
    pickle.dump(obj, f)
    pickle.dump(obj2, f)
```

In [95]:
```
with open('obj2.txt', 'rb') as f:
    obj_l2 = pickle.load(f)
    obj2_l = pickle.load(f)
obj_l2, obj2_l
```

Out[95]:
```
(['Строка', (12, 3)], (6, 7, 8, 9, 10))
```

Модуль pickle позволяет также преобразовать объект в строку байтов и восстановить объект из строки. Для этого предназначены две функции:

- dumps(<Объект> [, <Протокол>] [, fix_imports=True]) - производит сериализацию объекта и возвращает последовательность байтов специального формата.
- loads(<Последовательность байтов>[, fix_imports=True] [, errors="strict"])- преобразует последовательность байтов обратно в объект.

In [96]:
```
bs = pickle.dumps(obj)
bs
```

Out[96]:
```
b'\x80\x03]q\x00(X\x0c\x00\x00\x00\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb0q\x01K\x0cK\x03\x86q
\x02e.'
```

In [97]:
```
pickle.loads(bs)
```

Out[97]:
```
['Строка', (12, 3)]
```

Модуль shelve позволяет сохранять объекты под определенным ключом (задается в виде строки) и предоставляет интерфейс доступа, сходный со словарями. Для сериализации объекта используются возможности модуля pickle, а чтобы записать получившуюся строку по ключу в файл, применяется модуль dBпL Все эти действия модуль shelve производит незаметно для нас.

Чтобы открыть файл с базой объектов, используется функция open(). Функция имеет следующий формат:

open (<Путь к файлу> [, flag="c"] [, protoco1=None] [, writeback=Fa1se])

В необязательном параметре flag можно указать один из режимов открытия файла:

- r - только чтение;
- w - чтение и запись;
- c - чтение и запись (значение по умолчанию). Если файл не существует, он будет создан;
- n - чтение и запись. Если файл не существует, он будет создан. Если файл существует, он будет перезаписан.

Функция open() возвращает объект, с помощью которого производится дальнейшая работа с базой данных. Этот объект имеет следующие методы:

- close() - закрывает файл с базой данных.
- keys() - возвращает объект с ключами;
- values() - возвращает объект to значениями;
- items() - возвращает объект, поддерживающий итерации. На каждой итерации возвращается кортеж, содержащий ключ и значение.
- get(<Ключ> [, <Значение по умолчанию>]) - если ключ присутствует; то метод возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то возвращается начение None или значение, указанное во втором параметре;
- setdefault(<Ключ> [, <Значение по умолчанию>]) ---: если ключ присутствует, то метод возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то вставляет новый элемент со значением, указанным во втором параметре, и возвращает это значение. Если второй параметр не указан, значением нового элемента будет None;
- pop (<Ключ> [, <Значение по умолчанию>]) - удаляет элемент с указанным ключом и возвращает его значение. Если ключ отсутствует, то возвращается значение из второго параметра. Если ключ отсутствует, и второй параметр не указан, то возбуждается исключение KeyError;
- popitem() - удаляет произвольный элемент и возвращает кортеж из ключа и значения. Если файл пустой, возбуждается исключение KeyError;
- clear () - удаляет все элементы. Метод ничего не возвращает в качестве значения;
- update () - добавляет элементы. Метод изменяет текущий объект и ничего не возвращает. Если элемент с указанным ключом уже присутствует, то его значение будет перезаписано.

Помимо этих методов можно воспользоваться функцией len() для получения количества элементов и оператором del для удаления определенного элемента, а также оператором in для проверки существования ключа.

In [106]:
```
import shelve
```

In [107]:
```
db = shelve.open("shl_1")
```

In [108]:
```
db["obj1"] = [1, 2, 3, 4, 5]
db["obj2"] = (6, 7, 8, 9, 10)
```

In [109]:
```
db["obj1"]
```

Out[109]:
```
[1, 2, 3, 4, 5]
```

In [110]:
```
db["obj2"]
```

Out[110]:
```
(6, 7, 8, 9, 10)
```

In [111]:
```
db.close()
```

In [112]:
```
db = shelve.open('shl_1')
```

In [114]:
```
db.keys()
```

Out[114]:
```
KeysView(<shelve.DbfilenameShelf object at 0x000000A73CA82630>)
```

In [115]:
```
list(db.keys())
```

Out[115]:
```
['obj2', 'obj1']
```

In [116]:
```
list(db.values())
```

Out[116]:
```
[(6, 7, 8, 9, 10), [1, 2, 3, 4, 5]]
```

In [117]:
```
for k, v in db.items():
    print('key: {}, value: {}'.format(k, v))
```

```
key: obj2, value: (6, 7, 8, 9, 10)
key: obj1, value: [1, 2, 3, 4, 5]
```

Модуль CSV

https://docs.python.org/2/library/csv.html (https://docs.python.org/2/library/csv.html)

https://pymotw.com/2/csv/ (https://pymotw.com/2/csv/)

In [119]:
```
import csv
```

In [120]:
```
with open('participants.csv') as f:
    f_csv = csv.reader(f, delimiter=';')
    header = next(f_csv)
    rows = [r for r in f_csv]
```

In [122]:
```
header
```

Out[122]:
```
['Last Name',
 'First Name',
 'Company Name',
 'Company Department',
 'Assigned Classifications',
 'City',
 'State',
 'Country']
```

In [121]:
```
rows
```

Out[121]:
```
[['AALTONEN',
  'Wanida',
  'University',
  'Graduate student',
  'School or university (student)',
  'London',
  '',
  'United Kingdom'],
 ['ABDELHAMID',
  'Mohamed',
  'alexandria university',
  'student',
  'School or university (student)',
  'alexandria',
  '',
  'Egypt'],
 ['ABDELMEGUID',
  'Sheref',
  'IBM EGYPT BRANCH',
```

Подготовка к следующей лекции:

Саммерфильд: гл. 5 Модули

Прохоренок: гл. 12 Модули и пакеты

Встроенные функции

Материалы по ссылкам:

https://docs.python.org/3/library/functions.html (https://docs.python.org/3/library/functions.html)

http://python-reference.readthedocs.io/en/latest/docs/functions/ (http://python-reference.readthedocs.io/en/latest/docs/functions/)

In [ ]:
```

```