Python

Работа с StringIO,json,xml, Sqlite3.
Файлы и файловая система.

Библиотеки для работы с буфером

from io import StringIO, BytesIO # Python3
from StringIO import StringIO # Python2 – Unicode или 8-bit strings
from cStringIO import StringIO # Python2 – 8-bit strings

```
# Find the best implementation available on this platform
try:
   from cStringIO import StringIO
except:
   from StringIO  import StringIO

# Writing to a buffer
output = StringIO()
output.write('This goes into the buffer. ')
print >>output, 'And so does this.'

# Retrieve the value written
print output.getvalue()
output.close() # discard buffer memory

# Initialize a read buffer
input = StringIO('Inital value for read buffer')

# Read from the buffer
print input.read()
```

Библиотека JSON

```
#Получение данных из строки с JSON:
json_string = '{"first_name": "Guido", "last_name":"Rossum"}'
import json
parsed_json = json.loads(json_string)
#Получаем обычный словарь:
print(parsed_json['first_name'])
>>>"Guido"

#Сериализуем в JSON
d = { 'first_name': 'Guido',
      'second_name': 'Rossum',
      'titles': ['BDFL', 'Developer'],
}

print(json.dumps(d))
>>>'{"first_name": "Guido", "last_name": "Rossum", "titles": ["BDFL", "Developer"]}'
```

Библиотеки xml.dom.minidom и lxml

```
<document>
    <name>Mikhail Ovsiannikov</name>
    <hobby>Teaching Students</hobby>
</document>
```

```
from xml.dom.minidom import parse
xml = parse('document.xml')
name = xml.getElementsByTagName('name')
for node in name:
    print node.childNodes[0].nodeValue
```

```
from lxml import etree
tree = etree.parse('document.xml')
print tree.xpath("/document/name/text()")
print tree.xpath("/document/hobby/text()")
```

Файловая система

1. **Открыть файл** можно с помощью функции **open**:
open(name[, mode[, buffering]])

Функция возвращает файловый объект.
Таблица режимов (mode) функции open:

'r' – чтение.
'w' – запись.
'a' – добавление.
'b' – бинарный режим.
'+' – чтение/запись.

Режим '+' может быть добавлен к остальным режимам.

- По умолчанию питон открывает файлы в текстовом режиме.
- Для открытия файла в бинарном режиме на чтение можно добавить 'rb'.

Третий параметр устанавливает размер буферизации при работе с файлом.

- По умолчанию он выключен, и чтение/запись идет напрямую с диска на диск.
- Для включения буфера третий параметр должен быть отличным от нуля.

2. Базовые файловые методы

В питоне многие объекты являются файлами: стандартный ввод **sys.stdin**, стандартный вывод **sys.stdout**, объекты, открываемые функцией **urllib.urlopen** и т.д.

Запись в файл:

```
>>> f = open('my_file', 'w')
>>> f.write('Hello, ')
>>> f.write('World!')
>>> f.close()
```

Чтение:

```
>>> f = open('my_file', 'r')
>>> f.read(5)
'Hello'
>>> f.read()
', World!'
```

Файловая система

3. Произвольный доступ

По умолчанию метод read() читает данные последовательно по порядку, от начала и до конца файла. Для произвольного доступа к файлу есть функция **seek**:
seek(offset[, whence])

offset – смещение в байтах относительно начала файла;

whence – по умолчанию равен нулю, указывает на то, что смещение берется относительно начала файла.
Пример:

```
>>> f = open(r'my_file', 'w')
>>> f.write('01234567890123456789')
>>> f.seek(5)
>>> f.write('Hello, World!')
>>> f.close()
>>> f = open(r'my_file')
>>> f.read()
'01234Hello, World!89'
```

Функция tell() возвращает текущую позицию файла.

Файловая система

4. Построчная работа с файлами

Обычно мы имеем дело с текстовыми файлами. Прочитать одну строку:
file.readline()

Функция **readline**() без параметра читает всю строку, наличие параметра указывает функции максимальное число символов строки, которое будет прочитано. Прочитать все строки и вернуть список строк:
file.readlines()

Записать строки в файл:
file.writelines()

Пример. Прочитать файл и записать его содержимое в другой

файл:

```
>>> f = open(r'my_file')
>>> lines = f.readlines()
>>> f.close()
>>> lines[0] = "This is a my_file2 \n" # изменяем 1-ю строку
>>> f = open(r'my_file2','w')
>>> f.writelines(lines)
>>> f.close()
```

Файловая система

5. Закрытие файла

Для закрытия файла есть метод close(). Обычно файл закрывается сам после того, как вы выходите из программы, но иногда файлы нужно закрывать вручную.

Для полной уверенности в закрытии файла можно использовать блок try/finally:

```
>>> try:
>>>     # Тут идет запись в файл
>>> finally:
>>>     file.close()
```

Можно также использовать менеджер контекста, который в любом случае закроет файл:

```
>>> with open("my_file") as somefile:
>>>     do_something(somefile)
```

Если вы все же не хотите закрывать файл, то синхронизировать многопользовательский доступ к файлу на чтение/запись можно с помощью функции **flush()**, которая актуализирует все операции записи на диск. При этом возможна блокировка файла на чтение.

Файловая система

6. Итерация

Использование функции read() для байтового чтения:

```
>>> f = open(filename)
>>> while True:
>>>     char = f.read(1)
>>>     if not char: break
>>>     process(char)
>>> f.close()
```

Построчное чтение текстовых файлов и функция readline():

```
>>> f = open(filename)
>>> while True:
>>>     line = f.readline()
>>>     if not line: break
>>>     process(line)
>>> f.close()
```

Файл сам может выступать в роли итератора:

```
>>> for line in open(filename):
>>>     process(line)
```

Файловая система

7. Закрытие файла

Для закрытия файла есть метод close(). Закрывать следует, т.к.:

- Питон может буферизировать запись в файл ваших данных, что может привести к неожиданным эффектам и возникновению ошибок.
- У операционной системы есть ограничение на число одновременно открытых файлов.
- При доступе к файлу из разных мест одновременно и на чтение, и на запись необходимо синхронизировать файловые операции. Буферизация записи может привести к тому, что запись уже произошла, а данных в файле еще нет.

Через блок try/finally:

```
try:
    # Тут идет запись в файл
finally:
    file.close()
```

Через менеджер контекста:

```
with open("my_file") as somefile:
   do_something(somefile)
```

Файловая система

8. Модуль os - интерфейс работы с файловой системой

Функция os.getcwd возвращает текущий каталог:

```
import os
cwd = os.getcwd()
print cwd
```

Проверить наличие файла в текущем каталоге:

```
os.path.exists('my_file')
```

Вывод списка файлов и подкаталогов для данного каталога:

```
os.listdir(path)
```

Рекурсивный вывод списока всех файлов и подкаталогов для данного каталога:

```
import os
def walk(dir):
  for name in os.listdir(dir):
    path = os.path.join(dir, name)
    if os.path.isfile(path):
        print path
    else:
        walk(path)
walk(path)
```

Файловая система

Создать каталог:

```
os.mkdir('TEXT')
```

Создать дерево каталогов:

```
os.makedirs('ONE/TWO/THREE')
os.listdir('ONE')
>>> ['TWO']
os.listdir('ONE/TWO')
>>>['THREE']
```

Список содержимого каталога (нерекурсивный):

```
os.listdir()
>>> ['text4.txt', 'text1.txt', 'text3.txt', 'try_except.py', 'files.py']
os.listdir('/home')
>>> ['pl']
```

Сведения о файле|каталоге:

```
os.stat('/home')
>>> posix.stat_result(st_mode=16877, st_ino=1048577, st_dev=2053,
st_nlink=3, st_uid=0, st_gid=0, st_size=4096, st_atime=1344368518,
st_mtime=1339316982, st_ctime=1339316982)
os.stat('text.txt')
>>> posix.stat_result(st_mode=33188, st_ino=1575740, st_dev=2053,
st_nlink=1, st_uid=1000, st_gid=1000, st_size=41, st_atime=1344124896,
st_mtime=1343895362, st_ctime=1343895362)
```

Файловая система

Переименовать файл|каталог:

```
os.rename('text2.txt', 'xett.txt')
os.rename('TEXT', 'ETXT')
```

Удаление каталога:

```
os.rmdir('ETXT')
```

Смена текущего каталога:

```
os.chdir('/home')
os.getcwd()
>>>'/home`
os.chdir('./pl/Documents/python')
os.getcwd()
>>>'/home/pl/Documents/python'
```

9. Модуль shutil, в отличие от os, позволяет выполнять операции над деревьями данных.

Скопировать дерево:

```
shutil.copytree('../python', '../copy-python')
```

Перемещение|переименование дерева:

```
shutil.move('../copy-python', 'python-copy')
```

Удаление дерева:

```
shutil.rmtree('python-copy')
```

Библиотека SQLite

```
-*- coding: utf-8 -*-
import sqlite3 as lite
import sys
con = None
try:
    con = lite.connect('test.db')
    cur = con.cursor()
    cur.execute('SELECT SQLITE_VERSION()')
    data = cur.fetchone()
    print "SQLite version: %s" % data
except lite.Error, e:
    print "Error %s:" % e.args[0]
    sys.exit(1)
finally:
    if con:
        con.close()
```

```
# -*- coding: utf-8 -*-
import sqlite3 as lite
import sys
con = lite.connect('test.db')
with con:
    cur = con.cursor()
    cur.execute("CREATE TABLE Cars(Id INT, Name TEXT, Price INT)")
    cur.execute("INSERT INTO Cars VALUES(1,'Audi',52642)")

lid = cur.lastrowid
```

```
# -*- coding: utf-8 -*-
import sqlite3 as lite
import sys
con = lite.connect('test.db')
con.row_factory = lite.Row
uId = 4
with con:
    cur = con.cursor()
    cur.execute("SELECT * FROM Cars WHERE Id=:Id", {"Id": uId})
    rows = cur.fetchall()
    for row in rows:
        print row
        print "%s %s %s" % (row["Id"], row["Name"], row["Price"])
```

Zen of Python

```
>>> import this
```