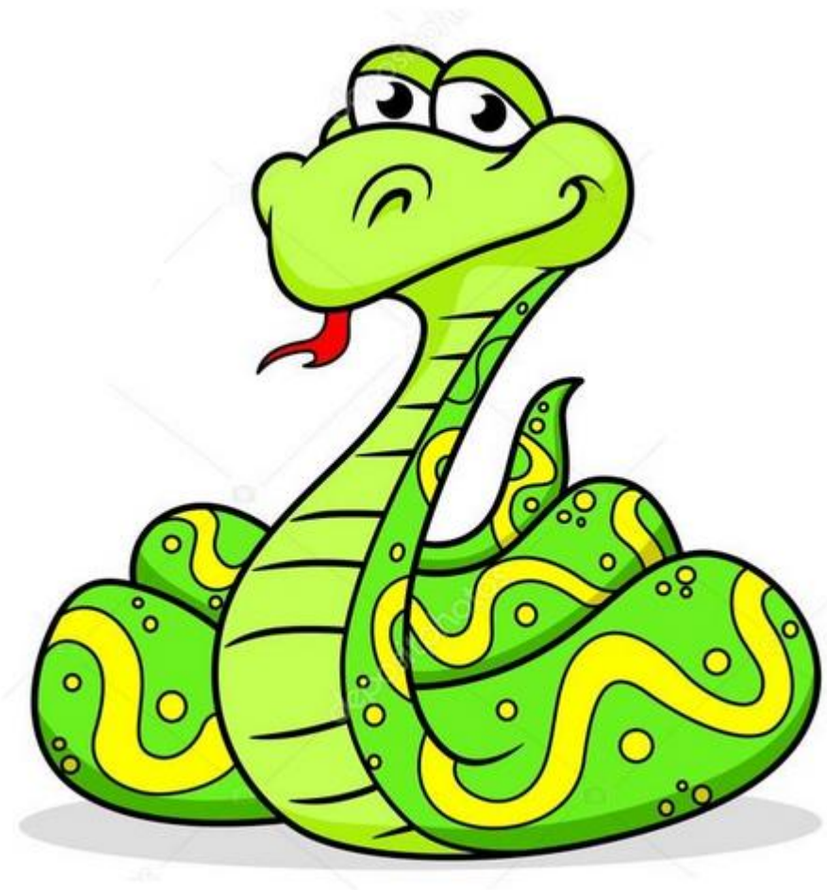
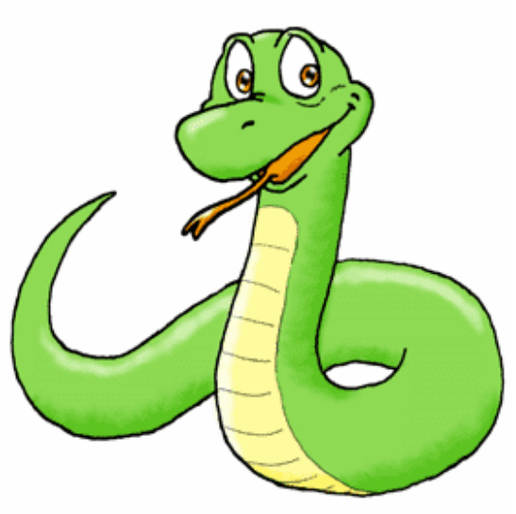
Python: Persistência de Dados



Prof. Dr. Dilermando Piva Jr.

Onde por meus dados?

... Arquivo neles!



... Clássico arquivos ...

- Suporte nativo para leitura e escrita de arquivos
- file =open('filePath', 'r') Mode: 'r', 'w', 'a' , file.close()
- file.read(), readlines(), readline()
- file.write(), writelines()
- Método mais simples para armazenamento de dados quando os mesmos não precisam de uma estrutura ou a recuperação dos mesmos não for um fator crítico!

... Clássico arquivos ...

```
>>>dir(file)
<tudo de arquivo>
>>>f = open("teste.tmp","w+")
>>>f.write("teste de arquivo!\n")
>>>f.close()
>>>f = open("teste.tmp", "a")
>>>f.writelines([ "linha1\n","linha2\n","linha3\n"])
>>>f.close()
>>>f = open("teste.tmp", "r")
>>>f.readlines()
['teste de arquivo!\n', 'linha1\n', 'linha2\n', 'linha3\n']
>>>f.readlines()
[]
>>>
```

Persistência de Objetos

- Pode-se escrever objetos em arquivos, basta usar o módulo **pickle**
- Dump
  - Transforma objetos em strings
- Load
  - Transforma objetos em strings

```
>>>import pickle
>>>pickle.dump(obj,file)
>>>obj = pickle.load(file)
```

Persistência de Objetos

```
>>> import pickle
>>> f = open("teste.tmp", "w")
>>> pickle.dump([ 1,2,3,4,5,6,7,8,9] ,f)
>>> f.close()
>>> f = open("teste.tmp", "r")
>>> f.read()
'(lp0\nI1\naI2\naI3\naI4\naI5\naI6\naI7\naI8\naI9\na.'
>>> f.close()
>>> f = open("teste.tmp", "r")
>>> pickle.load(f)
[ 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

Arquivos .CSV

- Comma Separated Values.
- Dados armazenados por algum delimitador (, ; \t)
- Suporte Nativo de Python (import csv)
- open('file.csv', 'rb') reader =csv.reader(file) , csv.writer(outfile,delimiter=',')
- csv.writerow(row) , file.close() , for row in reader (leitura)
- Arquivo que permite já armazenar dados com uma não tão sofisticada estrutura, mas permite ser aberto em aplicativos como Microsoft Excel!

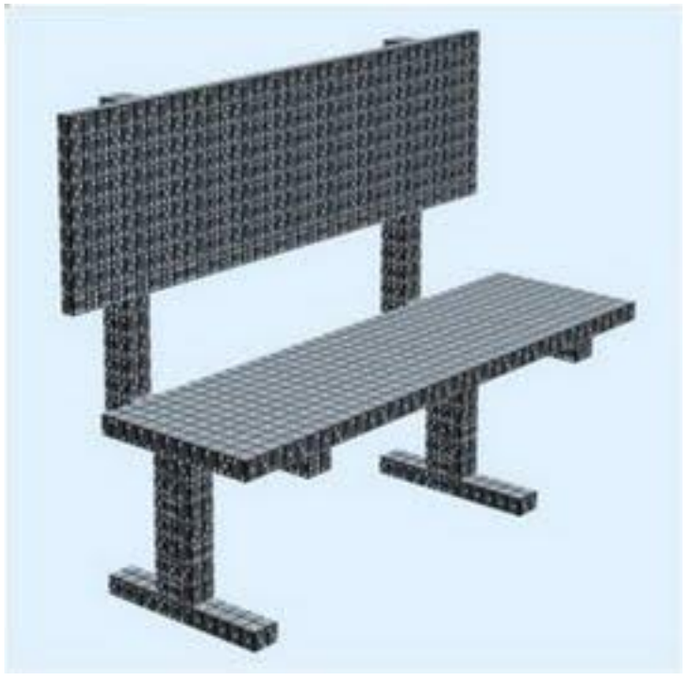
Arquivos .CSV

```
#To Read
import csv
reader = csv.reader(open("some.csv", "rb"))
for row in reader:
    print row

#To Write
import csv
writer = csv.writer(open("some.csv", "wb"))
writer.writerows(someiterable)
```

E Persistir em DBs ?

Persistência em Banco de dados!



Sqlite

- Sqlite é uma biblioteca em C que provê a construção de bancos de dados sem a necessidade de serviços de servidores

- Bastante utilizado para construção de pequenas bases de dados

- Para acesso ao banco, utiliza-se de uma variante da linguagem de consulta SQL.

- Você pode prototipar sua aplicação usando SQLite e depois portar o mesmo facilmente para bancos maiores como Oracle ou PostGreSQL

Abrindo uma conexão

- con = sqlite3.connect(*“mydb.db”*)
- Se o banco não existir, um banco vazio é criado

- con = sqlite3.connect(*“:memory:”*)
- Banco armazenado na mémoria e é sempre vazio quando criado.

Criando um cursor

- cur = con.cursor()
- Responsável pela instância de persistência (o cursor que irá realizar as transações sobre a conexão do banco corrente)

Executando uma consulta

```
cur.execute("select firstname, lastname from person")
row = cur.fetchone()
if row is None:
    # Error, no result
else:
    firstname, lastname = row[0], row[1]
```

Retorna linhas como tuplas !

Executando uma consulta

```
cur.execute("select firstname, lastname from person")
for row in cur:
    print "firstname: %s, lastname: %s" % (row[ 0] , row[ 1] )
```

O cursor é iterável, apenas fazer um loop sobre o cursor!

print cur.fetchall() *

*Degradação de performance! (Retorna todos os resultados)

Consultas com parâmetros

```
cur.execute(SQL, parameters)
```

SQL:

String em Python, que deve ser codificada em UTF-8 se conter caracteres não ASCII, ou em Unicode

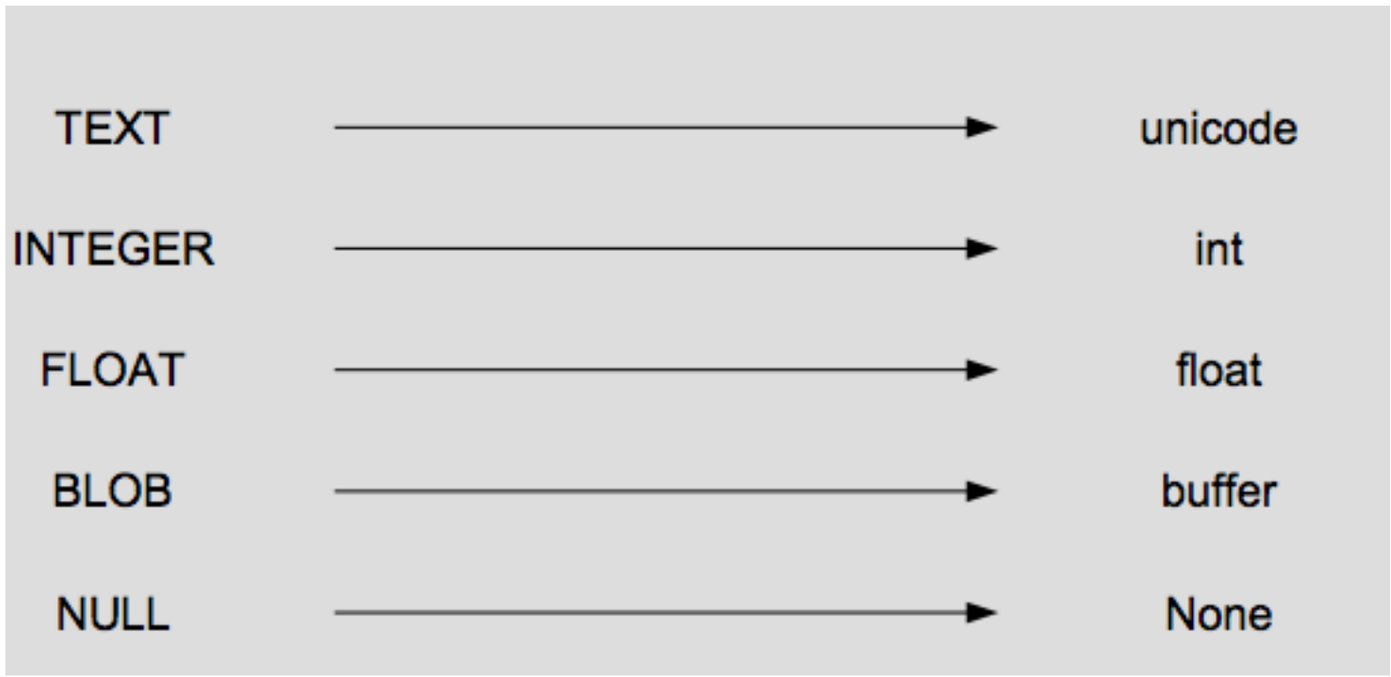
Parameters:

Sequência (lista, tupla) ou até um dicionário

Executando uma consulta

```
c.execute(“‘create table person  firstname text, secondname text, age integer””)
```

Criando Tabelas, mesma maneira que Sql comum, apenas não esquecer os tipos possíveis!



Executando uma consulta

```
cur.execute("""
      insert into person(firstname, lastname)
       values (?, ?)",
   ("Gerhard", "Haering")
)
```

Use ? como caracteres de posição e a sequência como parâmetros

Executando uma consulta

```
item = { "firstname": "Gerhard", "lastname": "Haering"}
cur.execute("""
      insert into person(firstname, lastname)
       values (:firstname, :lastname)",
  item
)
```

Use :nome como caracteres de posição e o dicionário como parâmetro

Oops ? Cadê meus dados?

sqlite3 usa **transações** para que o banco de dados sempre fique consistente. Para que as mudanças sejam permantentes você deve realizar uma operação de **commit**!

Commit neles!

```
cur = con.cursor()
cur.execute("insert into table1 ...")
cur.execute("insert into table2 ...")
con.commit()
```

Após modificações no banco, faça um commit sobre suas mudanças para que os dados sejam gravados de forma consistente.

Algum problema? Roll back!

```
cur = con.cursor()
try:
    cur.execute("delete from ...")
    cur.execute("delete from ...")
    con.commit()
except sqlite.DatabaseError:
    con.rollback()
```

Roll back permite desfazer algum commit quando algum erro acontece durante a transação (Objetivo: Manter os dados consistentes!)

Seja legal e feche tudo!

```
cur.close()
con.close()
```

Você deve fechar as conexões e cursores que não estão sendo mais usadas. Apenas feche a conexão quando os respectivos cursores estiverem fechados!

Vamos a um exemplo prático!

Date	Trans	Symbol	Qty	Price
2006-01-05	‘BUY’	IBM’	1000	45.00
2006-03-28	SELL’	MSOFT’	500	72.00
…	…	…	….	…

Vamos criar uma tabela Estoque de item e fazer algumas consultas!

Outros bancos de dados

- Várias bibliotecas disponíveis para diversos bancos:
  - Conector Oracle (cx_Oracle)
  - Conector PostGreSql (py-postgresql)
  - Conector Sql Server (pymssql)
  - Conector MySQL (MySQLdb)
  - Interbase/Firebird = (kinterbasdb)
- Python muito poderoso, não apenas fornece conectores para diversos bancos como também provê um mini-banco nativo de fácil acesso!

Outros bancos de dados

- Todos seguem a mesma linha, apenas o sqlite usa um arquivo local como banco ao invés de uma base de dados
- Existem pequenas diferenças entre as implementações
- Os comandos para obter conexões são os que mais variam

```
import MySQLdb
import sqlite
from pyPgSQL import PgSQL
import kinterbasdb
import pymssql
import cx_Oracle
```

```
#MySQL
con = MySQLdb.connect('servidor', 'usuario', 'senha')
con.select_db('banco de dados')

#SQLite
con = sqlite.connect('nome do arquivo', mode=775) # no futuro
                                                  # mode definirá o modo
                                                  # de trabalho
#PostgreSQL
con = PgSQL.connect(host='servidor',
                    database='banco de dados',
                    user='usuario',
                    password='senha')


#Interbase/Firebird
con = kinterbasdb.connect(dsn='servidor:/path/arquivo.fdb',
                          user='usuario',
                          password='senha')

#MSSQL
con = pymssql.connect(host = 'servidor',
                      user = 'usuario',
                      password = 'senha',
                      database = 'base de dados')

#Oracle
con = cx_Oracle.connect('usuario/senha@tnsname')
```

Outros bancos de dados

- Para obter resultado das requisições feitas através do comando execute devemos usar os comandos
  - >>> cursor.fetchone()
    - Uma linha
  - >>> cursor.fetchall()
    - Todas as linhas
  - >>> cursor.dictfetchall
    - Busca todas as linhas e retorna um dicionário com o nomes do campo e valor para cada linha
    - Os bancos Interbase/Firebird e MSSQL não dão suporte a este comando

Outros bancos de dados

- Obtendo o resultado das buscas pode-se verificar o resultado de duas maneiras

```
rs = cursor.fetchall()
print(rs[ 0] )  # valor da primeira coluna
```

```
for linha in cursor:
    print linha[ 0]  # valor da primeira coluna
```

Outros bancos de dados

- Será explorado agora um estudo de caso que usará o MySQL como banco de dados
- Devemos criar a conexão com o banco
- Selecionar o banco a ser usado
  - Difere um pouco dos outros por escolher o banco depois da conexão
  - Se o banco não existir devemos criá-lo e depois criar as tabelas do banco
- Agora é só usar os comandos fetchone, fetchall ou dictfechall para manipular
- Para confirmar as transações é importante usar o comando commit

Outros bancos de dados

```
import MySQLdb
import md5

try:
    dbConnection = MySQLdb.connect("localhost", "root", "123456")
except:
    print "Erro ao conectar ao banco de dados"
    exit()

cursor = dbConnection.cursor()

try:
    dbConnection.select_db("testdb")
except:
    cursor.execute("create database testdb")
    dbConnection.select_db("testdb")
    cursor.execute("create table usuarios (idusuario int not null auto_increment,"\
                   " nome varchar(15) not null, senha varchar(32) not null,"\
                   " primar\ key(idusuario));")

    print results[ 0], "--", results[ 1]
```

Outros bancos de dados

```
md5.update("123456")
cursor.execute("insert into usuarios (nome, senha) values ('usuario1', '"\
               + md5.hexdigest() + "')")

md5.update("abcdef")
cursor.execute("insert into usuarios (nome, senha) values ('usuario2', '"\
               + md5.hexdigest() + "')")

md5.update("123abc")
cursor.execute("insert into usuarios (nome, senha) values ('usuario3', '"\
               + md5.hexdigest() + "')")

cursor.execute("select nome from usuarios order by idusuario")
resultSet = cursor.fetchone()
print "-- O nome do primeiro usuario registrado e", resultSet[ 0]

print "-- Usuarios"
cursor.execute("select idusuario, nome from usuarios order by nome")
resultSet = cursor.fetchall()
for results in resultSet:
    print results[ 0] , "--", results[ 1]
```