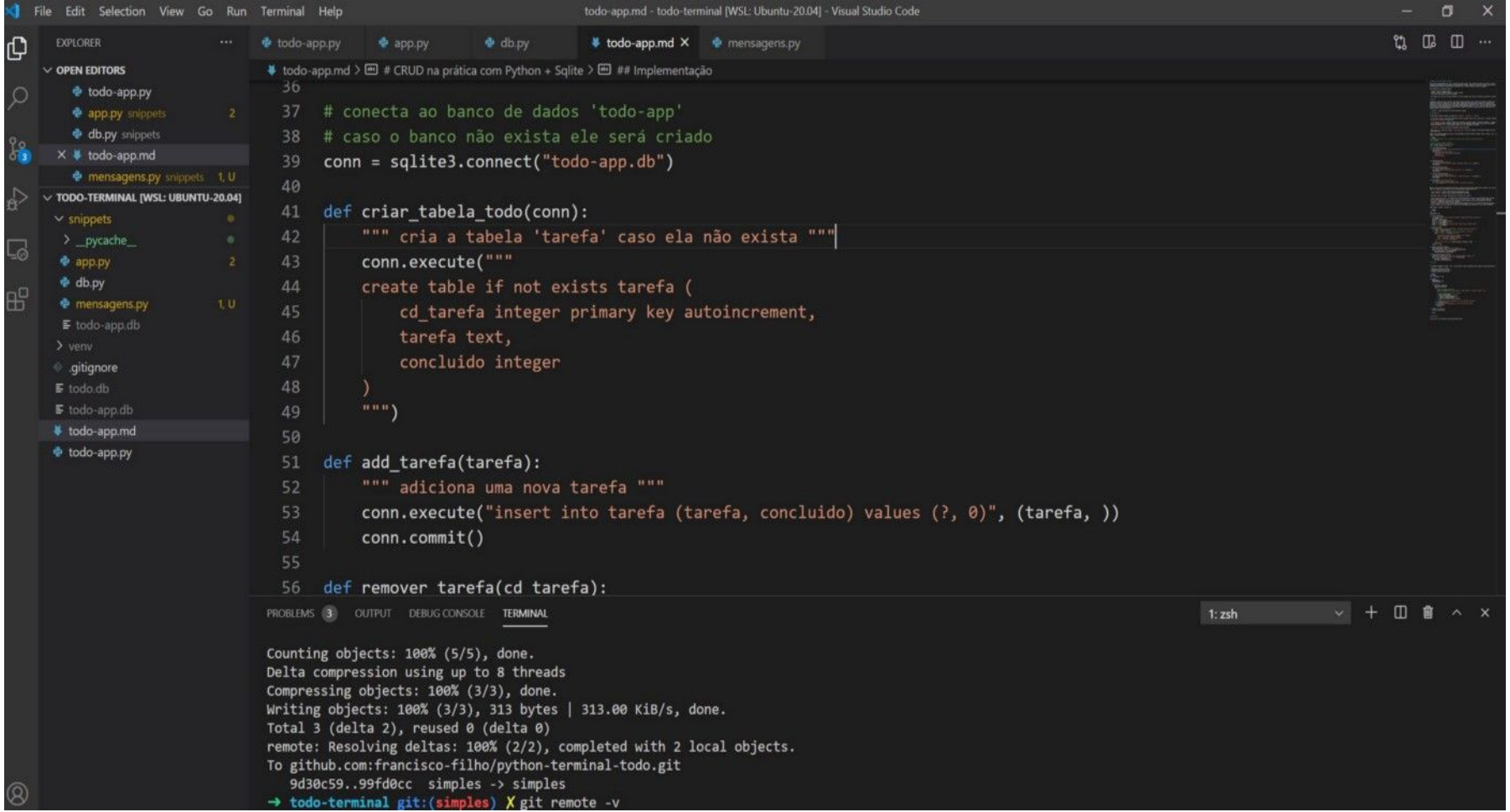
CRUD na prática com Python + Sqlite

Prof. MSc Cloves Rocha
car@etepd.com

07

Tópicos da aula

1. **Sqlite;**
    i. Conexão ao banco de dados;
2. **Implementação;**
    i. Inserir, alterar e remover registros no banco de dados;
    ii. Solicitação de input do usuário no terminal.
        1.
3. **Desafio #03;**
4. **Dúvidas;**
5. **Referências Bibliográficas.**



Fonte: https://medium.com/@francisco_51376/crud-na-pr%C3%A1tica-com-python-sqlite-8e15d11bbac9

Sqlite

- **Python já tem suporte nativo ao mesmo e não é necessário instalar uma biblioteca** sequer, para começar a utilizar-lo.
- De **bônus** todo o conhecimento que você obter vai poder ser utilizado com qualquer outro banco que tenha drivers para python desde que eles implementem a DBAPI de banco de dados para python.
- O **Sqlite** também **possui as principais funcionalidades dos grandes bancos**, só não é indicado para aplicações que recebam diversas conexões simultâneas como aplicações web, mas para nosso pequeno app (exemplo) ele é mais que perfeito.

Implementação

- Vamos iniciar com a conexão da base de dados.
    - Os principais métodos do **Sqlite** são:
        - connect()
        - execute()
        - commit()
- connect(path_arquivo) retorna uma conexão com o banco de dados sqlite que no nosso caso é o arquivo "todo-app.db" que será criado no diretório da aplicação.
- conn.execute(sql, tupla) executa comandos sql utilizando a conexão ao banco. O primeiro argumento é o código sql e o segundo parametro (opcional)
    - é uma tupla com as variáveis que serão usadas na consulta. Esse comando **retorna um objeto cursor** o qual podemos interar e ler os resultados da consulta.

Implementação

- Vamos iniciar com a conexão da base de dados.
    - `conn.commit()` comita (salva definitivamente) as alterações realizadas no banco de dados.
    - Nosso módulodb ainda tem o método `criar_tabela_todo()` que cria a tabela caso ela ainda não exista.
    - Vamos criar um módulo python que conterá a funcionalidade de acesso ao banco de dados, nomeie o arquivo `db.py` e digite o seguinte conteudo:

Implementação

main ETEPDPI2 / db.ipynb

clovesrocha Criado usando o Colaboratory

1 contributor

82 lines (82 sloc) | 2.6 KB

Open in Colab

In [1]:

```
# ETE PD
# Prof. Cloves
# Exemplo aula CRUD

# arquivo db.py

import sqlite3

# conecta ao banco de dados 'todo-app'
# caso o banco não exista ele será criado
conn = sqlite3.connect("todo-app.db")

def criar_tabela_todo():
    """ cria a tabela 'tarefa' caso ela não exista
    cursor = conn.cursor()
    conn.execute("""
    create table if not exists tarefa (
        cd_tarefa integer primary key autoincrement
        tarefa text,
        concluido integer
    )
    """)

def add_tarefa(tarefa):
    """ adiciona uma nova tarefa """
    conn.execute("insert into tarefa (tarefa, concl
    conn.commit()

def remover_tarefa(cd_tarefa):
    """ remove a tarefa da tabela """
    conn.execute("delete from tarefa where cd_taref
    conn.commit()

def concluir_tarefa(cd_tarefa):
    """ marca a tarefa como concluida """
    conn.execute("update tarefa set concluido = 1 w
    conn.commit()

def get_tarefas(): # retorna um cursor
    """ retorna a lista de tarefas cadastras """
    return conn.execute("select cd_tarefa, tarefa,
```

~/Documentos/ETE PD/CICLO 2022/crud_python_app_sqlite.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

crud_python_app_sqlite.py

```
1  # ETE PD
2  # Prof. Cloves
3  # Exemplo aula CRUD
4
5  # arquivo db.py
6
7  import sqlite3
8
9  # conecta ao banco de dados 'todo-app'
10 # caso o banco não exista ele será criado
11 conn = sqlite3.connect("todo-app.db")
12
13 def criar_tabela_todo():
14     """ cria a tabela 'tarefa' caso ela não exista """
15     cursor = conn.cursor()
16     conn.execute("""
17     create table if not exists tarefa (
18         cd_tarefa integer primary key autoincrement,
19         tarefa text,
20         concluido integer
21     )
22     """)
23
24 def add_tarefa(tarefa):
25     """ adiciona uma nova tarefa """
26     conn.execute("insert into tarefa (tarefa, concluido) values (?, 0)", (tarefa, ))
27     conn.commit()
28
29 def remover_tarefa(cd_tarefa):
30     """ remove a tarefa da tabela """
31     conn.execute("delete from tarefa where cd_tarefa = ?", (cd_tarefa, ))
32     conn.commit()
33
34 def concluir_tarefa(cd_tarefa):
35     """ marca a tarefa como concluida """
36     conn.execute("update tarefa set concluido = 1 where cd_tarefa = ?", (cd_tarefa, ))
37     conn.commit()
38
39 def get_tarefas(): # retorna um cursor
40     """ retorna a lista de tarefas cadastras """
41     return conn.execute("select cd_tarefa, tarefa, concluido from tarefa")
```

replit

Implementação

Agora é hora de criarmos os métodos que exibirão as mensagens da nossa aplicação, ou a interface de linha de comando. A primeira vista este código parece complexo, mas o que ele faz é simplesmente imprimir informações na tela de maneira tabulada. Utilizamos dois métodos para isso:

`exibir_cabecalho()` exibe o nome do app com informações básicas
`exibir_tarefas()` exibe a lista de tarefas cadastradas no sistema

Logo depois temos os métodos utilizados para interação com o usuário:

`mostrar_opcao_nova_tarefa()` O sistema mostra a opção perguntando ao usuário o que ele deseja fazer, as duas opções disponíveis são incluir uma nova tarefa ou concluir uma tarefa. Caso o usuário selecione incluir o app perguntará a descrição da tarefa que ele quer cadastrar. Mas se ele selecionar a opção de conclusão, o método explicado abaixo será chamado.

`mostrar_opcao_concluir_tarefa()` Nesta opção o sistema pergunta ao usuário o código da tarefa que ele deseja concluir, quando o usuário informa o valor, o sistema marca a atividade como concluída e o ciclo recomeça novamente.
Segue abaixo o arquivo `mensagens.py`

Leia o QR CODE
MiniCurso de Introdução à **Python** no Google **Colab** + **GitHub**

Implementação

Segue abaixo o arquivo `mensagens.py`

```
# ETE PD
# Prof. Cloves
# Exemplo aula CRUD
# arquivo mensagens.py

import db
MENU_INICIAL = 99


def exibir_cabecalho():
    """ imprimi o cabeçalho no terminal utilizando o tamanho maximo de 60
caracteres """
    QTD_COLUNAS = 60
    print ("-" * QTD_COLUNAS)
    print ("{:^60}".format("TAREFAS"))
    print ("-" * QTD_COLUNAS)
    print ("{:^60}".format("tecle 99 volta para o menu inicial, [CTRL+C] sair"))
    print ("-" * QTD_COLUNAS)
```

Leia o QR CODE
MiniCurso de Introdução à **Python** no Google **Colab** + **GitHub**

Implementação

```
def exibir_tarefas():
    """ exibe a lista de tarefas cadastradas, com algumas formatações básicas """
    for tarefa in db.get_tarefas():
        # check = \u2713 é o caracter unicode que representa o concluido
        check = u'\u2713' if tarefa[2] == 1 else ""
        """
            os parametros passados para esse format() são o seguinte
            {:>4}  = 4 posições, alinhado a direita
            {:<47} = 47 posições, alinhado a esquerda
            {:^3}  = 3 posições, centralizado
        """
        t = "- [{:>4}] {:<47} {:^3}".format(tarefa[0], tarefa[1], check)
        print (t)
    print ("-" * 60)

def mostrar_opcao_nova_tarefa():
    texto_nova_tarefa = input("Descreva a Tarefa => ")
    print ("adicionando tarefa -> " + str(texto_nova_tarefa))
    if texto_nova_tarefa != str(MENU_INICIAL):
        db.add_tarefa(texto_nova_tarefa)
```

Leia o QR CODE
MiniCurso de Introdução à **Python** no Google **Colab** + **GitHub**

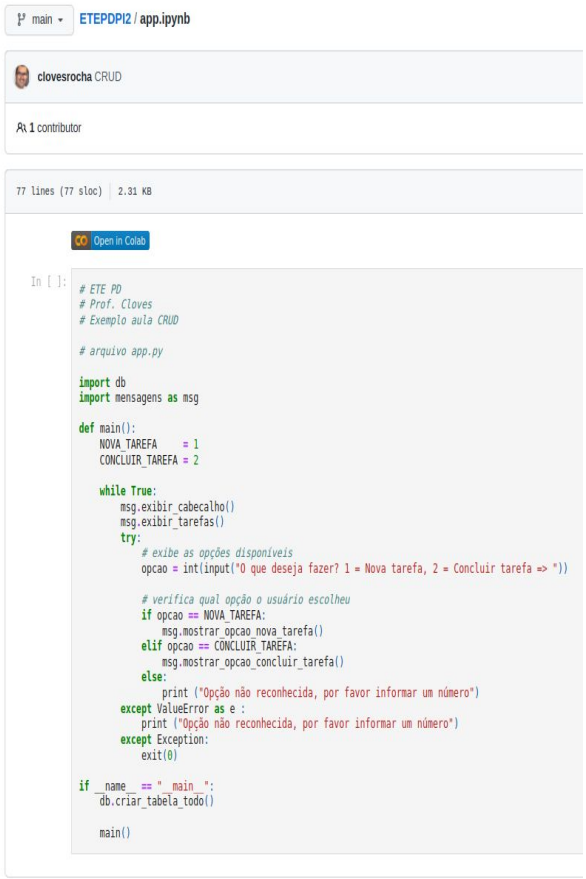
Implementação

Finalmente chegamos ao método `main()` que controla o fluxo do programa.
Nele criamos um loop infinito onde:

- exibimos o cabeçalho e tarefas;
- solicitamos a interação do usuário;
- e depois repetimos tudo de novo.

Leia o QR CODE
MiniCurso de Introdução à **Python** no Google **Colab** + **GitHub**

Implementação




```
# arquivo app.py

import db
import mensagens as msg

def main():
    NOVA_TAREFA     = 1
    CONCLUIR_TAREFA = 2

    while True:
        msg.exibir_cabecalho()
        msg.exibir_tarefas()
        try:
            # exibe as opções disponíveis
            opcao = int(input("O que deseja fazer? 1 = Nova tarefa, 2 = Concluir tarefa => "))
```

NOTA:
Você deve ter notado que apesar de nosso módulo **db** possuir um **método para remover tarefas**, nós **não demos esta opção para o usuário**, isto ficará como DESAFIO #03 para você.

Leia o QR CODE
MiniCurso de Introdução à **Python** no Google **Colab** + **GitHub**

Agora que vocês conheceram melhor **CRUD** na prática com **Python** +**Sqlite**.

Vamos praticar um pouco!

Desafio #03

Conforme classroom da disciplina.

INSTRUÇÕES

- Faça um **CRUD** com Python +Sqlite para ONG Casa Menina Mulher.
- No classroom da disciplina.
- Recomendo usar o **replit.**

DÚVIDAS?

Referências Bibliográficas

- CRUD na prática com Python + Sqlite
- (https://medium.com/@francisco_51376/crud-na-pr%C3%A1tica-com-python-sqlite-8e15d11bbac9)
- Documentação Sqlite
- (https://www.sqlite.org/docs.html)