



Técnicas de Programação II

Profº Ramon Trigo

Utilizando Mongo DB



1º Passo Instalar a biblioteca pymongo digitar o comando no cmd do Windows.

Comando -> `pip install pymongo`

2º Digitação do Código

```
from tkinter import *  
from tkinter import ttk  
import tkinter as tk  
import pymongo
```

**Biblioteca para
utilização do
combobox**

Biblioteca do mongodb

Configuração da tela

```
tela = Tk()  
tela.title("Exemplo Mongo DB")  
tela.geometry("800x600")  
tela.resizable(True, True)  
tela.configure(background="#ffffff")
```

Configuração do mongodb

```
exemplo = pymongo.MongoClient("mongodb://localhost:27017/")  
db = exemplo["exemplo"]  
collection = db["clientes"]
```

Exemplo Mongo DB

Cadastro de Clientes



```
lbl_titulo = Label(tela, text="Cadastro de Clientes",  
font=("Arial", 30, "bold"), bg="#ffffff").place(x=200, y=50)
```

Cadastro de Clientes

Código:
 Nome: CPF:

 Idade: Rua:


```
lbl_codigo = Label(tela, text="Código:", bg="#ffffff").place(x=130, y=140)
txt_codigo = Entry(tela, width=20, borderwidth=2, fg="black", bg="white")
txt_codigo.place(x=190, y=140)

lbl_nome = Label(tela, text="Nome:", bg="#ffffff").place(x=130, y=170)
txt_nome = Entry(tela, width=40, borderwidth=2, fg="black", bg="white")
txt_nome.place(x=190, y=170)
txt_nome.insert(0, "")

lbl_cpf = Label(tela, text="CPF:", bg="#ffffff").place(x=450, y=170)
txt_cpf = Entry(tela, width=20, borderwidth=2, fg="black", bg="white")
txt_cpf.place(x=480, y=170)
txt_cpf.insert(0, "")
```

Exemplo Mongo DB

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:



```
lbl_idade = Label(tela, text="Idade:", bg="#ffffff").place(x=130, y=200)
txt_idade = Entry(tela, width=20, borderwidth=2, fg="black", bg="white")
txt_idade.place(x=190, y=200)
txt_idade.insert(0, "")
```

```
lbl_end = Label(tela, text="Rua:", bg="#ffffff").place(x=450, y=200)
txt_end = Entry(tela, width=25, borderwidth=2, fg="black", bg="white")
txt_end.place(x=480, y=200)
txt_end.insert(0, "")
```

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:

```
lbl_bairro = Label(tela, text="Bairro:", bg="#ffffff").place(x= 130, y= 230)
txt_bairro = Entry(tela, width=20, borderwidth=2, fg="black", bg="white")
txt_bairro.place(x= 190, y=230)
txt_bairro.insert(0, " ")
```


Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:

```
lbl_estado = Label(tela, text="Estado:", bg="#ffffff").place(x=330, y=230)
comboestado = ttk.Combobox(tela,
                             values=[
                                 "São Paulo",
                                 "Rio de Janeiro",
                                 "Minas Gerais",
                                 "Espírito Santo"],)

comboestado.grid(column=0, row=1)
comboestado.place(x=370, y=230)
```

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:

```
lbl_cidade = Label(tela, text="Cidade:", bg="#ffffff").place(x= 520, y= 230)
txt_cidade = Entry(tela, width=20, borderwidth=2, fg="black", bg="white")
txt_cidade.place(x= 570, y=230)
txt_cidade.insert(0, " ")
```

Até o momento criamos parcialmente a tela de cadastro de clientes
Neste momento começaremos criar as funções.

```
def salvar():  
    codigo = txt_codigo.get()  
    nome = txt_nome.get()  
    idade = int(txt_idade.get())  
    end = txt_end.get()  
    cpf = txt_cpf.get()  
  
    bairro = txt_bairro.get()  
    cidade = txt_cidade.get()  
    estado = comboestado.get()
```



Cada variavel ligada a
uma caixa de texto

continua

```
txt_codigo.delete(0, tk.END)
txt_nome.delete(0, tk.END)
txt_idade.delete(0, tk.END)

txt_end.delete(0, tk.END)
txt_bairro.delete(0, tk.END)
txt_cidade.delete(0, tk.END)
comboestado.set("")
txt_cpf.delete(0, tk.END)
```



Após a inserção dos dados as caixas de textos, será apagada automaticamente

continua

```
cliente = {"código":codigo, "nome": nome, "idade":idade, "endereço": end, "cpf":cpf,  
           "bairro":bairro, "cidade":cidade, "estado": estado}  
collection.insert_one(cliente)
```




Dicionário chamado cliente está sendo criado. Este dicionário é usado para representar um conjunto de informações relacionadas a um cliente. Cada campo do cliente é representado como uma chave (por exemplo, "código", "nome", "idade", etc.) e o valor associado a cada chave é obtido de variáveis ou valores existentes no código, como codigo, nome, idade, etc.

collection é uma instância de um objeto que representa uma coleção de um banco de dados NoSQL (como o MongoDB). A função insert_one é chamada para inserir o dicionário cliente na coleção.

Função Atualizar


```
def atualizar():  
    codigo = txt_codigo.get()  
    nome = txt_nome.get()  
    idade = int(txt_idade.get())  
  
    end = txt_end.get()  
    cpf = txt_cpf.get()  
  
    bairro = txt_bairro.get()  
    cidade = txt_cidade.get()  
    estado = comboestado.get()  
  
    collection.update_one({"código":codigo}, {"$set": {"código":codigo, "nome": nome,  
                                                         "idade":idade, "endereço": end, "cpf":cpf,  
                                                         "bairro":bairro,  
                                                         "cidade":cidade, "estado": estado}}})
```



O método `update_one` é comumente usado em bancos de dados NoSQL, como o MongoDB, para atualizar um único documento que atende a um critério específico.

Função Atualizar

```
def atualizar():  
    codigo = txt_codigo.get()  
    nome = txt_nome.get()  
    idade = int(txt_idade.get())  
  
    end = txt_end.get()  
    cpf = txt_cpf.get()  
  
    bairro = txt_bairro.get()  
    cidade = txt_cidade.get()  
    estado = comboestado.get()  
  
    collection.update_one({"código":codigo}, {"$set": {"código":codigo, "nome": nome,  
                                                         "idade":idade, "endereço": end, "cpf":cpf,  
                                                         "bairro":bairro,  
                                                         "cidade":cidade, "estado": estado}}})
```



O primeiro argumento do `update_one` é um dicionário que especifica o critério de pesquisa. Neste caso, o critério é que o campo "código" no banco de dados corresponda ao valor da variável `codigo`.


Função Atualizar

```
def atualizar():
    codigo = txt_codigo.get()
    nome = txt_nome.get()
    idade = int(txt_idade.get())

    end = txt_end.get()
    cpf = txt_cpf.get()

    bairro = txt_bairro.get()
    cidade = txt_cidade.get()
    estado = comboestado.get()

    collection.update_one({"código":codigo}, {"$set": {"código":codigo, "nome": nome,
                                                         "idade":idade, "endereço": end, "cpf":cpf,
                                                         "bairro":bairro,
                                                         "cidade":cidade, "estado": estado}}})
```



O segundo argumento é outro dicionário que usa a operação \$set para especificar quais campos e valores devem ser atualizados no registro correspondente, todos os campos estão sendo atualizados com os valores das variáveis correspondentes, como "código", "nome", "idade", "endereço", "cpf", "bairro", "cidade" e "estado".

Função Apagar

```
def apagar():  
    codigo = txt_codigo.get()  
    collection.delete_one({"código": codigo})
```



Função utiliza a função `delete_one` para excluir um registro no banco de dados.

A exclusão é feita com base na correspondência do campo "código" do registro com o valor armazenado na variável `codigo`.

O único argumento para a função `delete_one` é um dicionário que especifica o critério de pesquisa. Neste caso, o critério é que o campo "código" no banco de dados corresponda ao valor da variável `codigo`.

Agora que já criamos as funções iremos criar os botões

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:

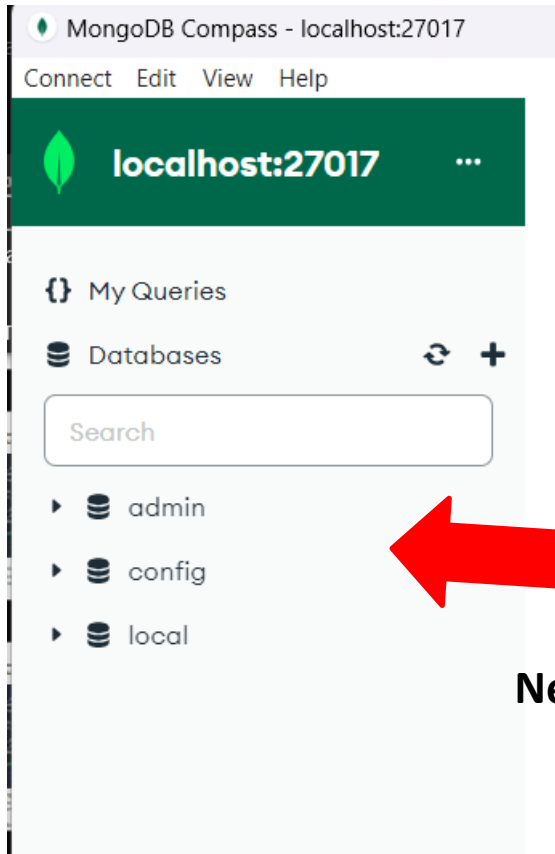
Salvar Alterar Excluir Sair

```
btn_salvar = Button(tela, text="Salvar", width=10, command=salvar).place(x=130, y=280)
btn_alterar = Button(tela, text="Alterar", width=10, command=atualizar).place(x=220, y=280)
btn_excluir = Button(tela, text="Excluir", width=10, command=apagar).place(x=310, y=280)
btn_sair = Button(tela, text="Sair", width=10, command=tela.quit).place(x=400, y=280)
tela.mainloop()
```



Cada botão está chamando uma função específica

Realizando o teste da aplicação - Salvar



Nenhuma base existente

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:


Preencha os campos

Click no botão Salvar



Atualize o mongo db

MongoDB Compass - localhost:27017





Connect Edit View Help

 **localhost:27017** ...

{ } My Queries

☰ Databases  

Search

- ▶  admin
- ▶  config
- ▶  exemplo
- ▶  local



Base

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:

Salvar

Alterar

Excluir

Sair

 **ADD DATA** ▾

 **EXPORT DATA** ▾

```

_id: ObjectId('654ac5f5e05d8af1616e400b')
código: "01"
nome: "Fatec Registro"
idade: 2
endereço: "Principal"
cpf: "123"
bairro: " Centro"
cidade: " Registro"
estado: "São Paulo"
    
```

Teste Alterar

Cadastro de Clientes

Código:

Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:

```
_id: ObjectId('654ac5f5e05d8af1616e400b')
código: "01"
nome: "Teste"
idade: 15
endereço: "Outra"
cpf: "3456"
bairro: "Centro"
cidade: "Belo Horizonte"
estado: "Minas Gerais"
```

Preencha todos os campos com os valores diferentes do 1º teste, mantendo apenas o mesmo código (01).

Atualizado no Mongo DB

Teste Excluir

Cadastro de Clientes

Código:

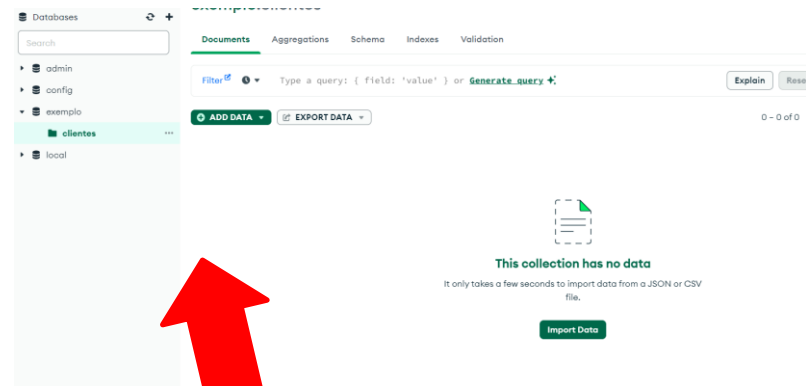
Nome: CPF:

Idade: Rua:

Bairro: Estado: Cidade:



Digite no campo código – 01
Click em Excluir



Nenhuma Collection
Existente