FICHA TRABALHO 7
PROJETO

MODALIDADE:	Aprendizagem +	Escolha um item.	
CURSO:	Técnico/a Especialista em Gestão de Informação e Ciência dos Dados		
UC:	Desenvolver projeto de tecnologias e aplicações informáticas de gestão	CÓDIGO UC:	00891
FORMADOR/A:	Bruno Silva	DATA:	

OBJETIVOS

- Criação de um projeto de gestão com Python

Para esta ficha de trabalho, vamos fazer a criação de um pequeno projeto para gerir uma tabela na base de dados MySQL. O objetivo será fazer a listagem, inserção, alteração e eliminação de dados, sem a necessidade de fazer a intervenção direta na base de dados. Como tal, vamos utilizar a linguagem Python e o conetor da base de dados para MySQL para ajudar a gerir a tabela em causa.

Parte 1 – Base de Dados e Configuração do projeto

Passo 1 – No MySQL, vamos **criar a base de dados** com o nome "**projeto**" e construir a **tabela** com o nome "**Produtos**" com os seguintes campos:

Estrutura da tabela | Visão de relação(ões)

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Comentários	Extra	Acções
1	ID_Produto	int(5)			Não	*Nenhum*		AUTO_INCREMENT	Muda Eliminar
2	Nome	varchar(75)	utf8mb4_general_ci		Não	*Nenhum*			Muda Eliminar
3	Descricao	varchar(150)	utf8mb4_general_ci		Não	*Nenhum*			Muda Eliminar
4	Preco	decimal(5,2)			Não	*Nenhum*			Muda Eliminar
5	IVA	int(2)			Não	*Nenhum*			Muda Eliminar
6	Ativo	tinyint(1)			Não	*Nenhum*			Muda Eliminar

Resumo dos dados da tabela Produtos:

- Os campos **ID_Produto** e **IVA** são campos numéricos inteiros e como tal, tem o tipo de dados **int** (inteiros);
- Os campos **Nome** e **Descricao** são campos textuais e como tal, tem o tipo de dados **varchar** (este tipo de dados pode suportar até 65535 caracteres);
- O campo **Preco** é um campo para colocar valores monetários (números decimais) e como tal, tem o tipo de dados **decimal** que é **composto por 2 partes**: número de casas inteiras e número de casas decimais separados por vírgula (5,2);
- O campo **ID_Produto** é chave primária da tabela e ao mesmo tempo é Auto_Increment (o valor deste campo vai ser incrementado automaticamente pela base de dados a cada nova informação);

Passo 2 – Crie uma pasta no Visual Studio Code com o nome "**Projeto**";

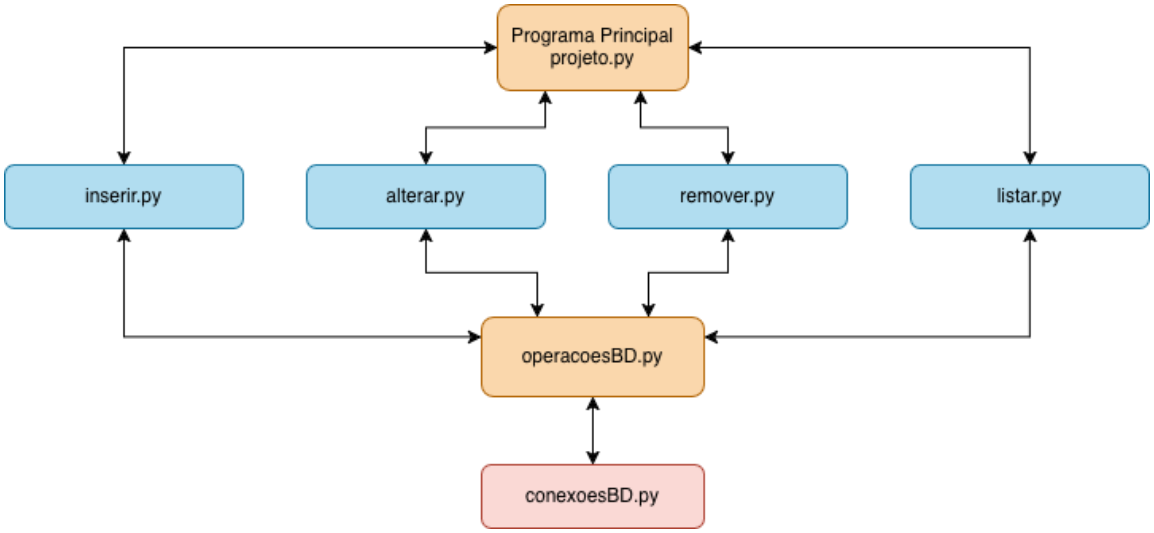
Passo 3 – No terminal do Visual Studio, vamos instalar o conector da base de dados Mysql.conector. Como tal, introduza a seguinte instarução: **python -m pip install mysql-connector-python**

```
python -m pip install mysql-connector-python
```

Passo 4 – Faça a criação dos seguintes ficheiros: inserir.py, alterar.py, remover.py, listar.py, projeto.py, conexoesBD.py e operacoesBD.py;

O objetivo do nosso trabalho prático é construir a conexão entre os vários ficheiros para, mais especificamente:

- **projeto.py**: ficheiro responsável por ter apenas o menu principal do programa e fazer a comunicação com as funções nos ficheiros inserir.py, alterar.py, remover.py, listar.py;
- **inserir.py**: vai conter a função para pedir dados para inserir na base dados;
- **alterar.py**: vai conter a função para pedir os dados a alterar e enviar para a base dados;
- **remover.py**: vai conter a função para remover dados e enviar para a base dados;
- **listar.py**: vai conter a função para listar a tabela da base de dados;
- **conexoesBD.py**: vai conter os dados da ligação para a base de dados e funções de conexão;
- **operacoesBD.py**: ficheiro para receber os dados dos ficheiros inserir, alterar, eliminar e listar dados que são enviados pelos ficheiros inserir.py, alterar.py, remover.py, listar.py;



Parte 2 – Programa Principal (projeto.py)

No programa principal, vamos realizar as seguintes operações:

Passo 1 – Criar uma estrutura de repetição com um menu com 5 opções:

```
#Estrutura de repetição
while True:
    print("***** Gestão Produtos V 1.00 *****")
    print("Menu: \n")
    print("1 - Listar Produtos")
    print("2 - Inserir Produto")
    print("3 - Alterar Produto")
    print("4 - Remover Produto")
    print("5 - Sair")
```

Passo 2 – Devemos pedir qual a opção ao utilizador e devemos aceitar apenas números inteiros. Utilizamos uma estrutura de repetição para que enquanto o utilizador tentar inserir a opção e este não for válido aparecer a mensagem “Deve introduzir apenas números inteiros”:

```
#Pedir a opção ao utilizador
while True:
    try:
        opcao = int(input("Insira a opção: "))
        break
    except ValueError:
        print("Deve introduzir apenas números inteiros")
```

Passo 3 – Após a validação do passo anterior, vamos construir a **estrutura de decisão seletiva** (e por enquanto, apresentar valores textuais). Na opção 5 vamos pedir para encerrar o programa e no caso por defeito (case_): irá mostrar a mensagem “Opção inválida” (caso o utilizador coloque valores fora dos intervalos de valores do menu):

```
#estrutura de decisao seletiva
match opcao:
    case 1: print("selecionou a opção 1")
    case 2: print("selecionou a opção 2")
    case 3: print("selecionou a opção 3")
    case 4: print("selecionou a opção 4")
    case 5:
        print("Encerrar programa... ")
        break
    case _: print("Opção inválida")
```

Parte 3 – Ficheiro configuração BD (conexaoBD.py)

Este ficheiro vai servir para colocar as instruções **para fazer a conexão com a base de dados**, do qual temos de informar os dados para interligar com a base de dados e como é feita a abertura da conexão.

Passo 1 – No início do ficheiro, vamos começar por importar bibliotecas essenciais para o processo da conexão com a base de dados:

- **A 1ª importação** serve para carregar o driver da conexão da base de dados (que instalaram na parte 1 – passo 3);
- **A 2ª importação** serve para carregar o tratamento de erros que podem ocorrer na base de dados (para depois informar o utilizador qual foi o erro gerado);

```
import mysql.connector
from mysql.connector import errorcode
```

Passo 2 – De seguida, vamos criar um objeto para armazenar os dados da conexão (para não estar sempre a escrever os mesmos dados em vários locais. Como tal, vamos criar o objeto com nome dados_BD e dentro das chavetas curvas vamos indicar um conjunto de par chave-valor para representar o campo e o valor que vão ser enviados na conexão:

```
#Dados da conexão ao MySQL
dados_BD = {
    'user':'root',
    'password': 'root',
    'host': '127.0.0.1', #mesma coisa que localhost
    'port': '3306',
    'database': '10805_projeto',
    'raise_on_warnings': True
}
```

Passo 3 – Para finalizar, vamos **criar a função** com o nome "conexao_MySQL" e dentro desta, vamos criar o **bloco try** para tentar fazer a conexão com a base de dados, caso contrário, o **bloco do catch** irá entrar no erro mais adequado, mais especificamente:

```
def conexao_MySQL():
```

- **Bloco try**: neste bloco vamos fazer a conexão com a base de dados e retorno da conexão para o programa (para não estar sempre a chamar pela mesma instrução e de forma repetida)

```
def conexao_MySQL():
    try:
        #tentar conectar com a base de dados
        return mysql.connector.connect(**dados_BD)
```

- **Bloco catch (tratamento dos potenciais erros):**
  - **Erro**: errorcode.ER_ACCESS_DENIED_ERROR
    - Será gerado caso o utilizador ou Password estejam errados;
  - **Erro**: ER_BAD_DB_ERROR
    - Será gerado caso a base de dados não existir;
  - **Outros erros**
    - Será gerado outros erros que não sejam os que estão mais acima;

```
except mysql.connector.Error as erro:
    if erro.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("ERRO Conexão MySQL 001: Utilizador ou Password errados!")
    elif erro.errno == errorcode.ER_BAD_DB_ERROR:
        print("ERRO Conexão MySQL 002: Base de dados não existe!")
    else:
        print(f"ERRO Conexão MySQL 003: {erro}")
```

Parte 4 – Ficheiro inserir dados (inserir.py) + (operacoesBD.py)

Este ficheiro vai servir para colocar as instruções para inserir dados na tabela da base de dados.

Passo 1 – No início do ficheiro, vamos criar a função com o nome **inserir_produto()**. Este por sua vez vai ser invocado mais tarde no programa principal por forma a fazer interligações entre menus:

```
def inserir_produto():

    #Construção do Menu
    print("*****Inserir Produto ***** \n")
```

Passo 2 – Numa primeira perspetiva podíamos pedir ao programa para receber os dados de forma simples com a função input e, quando necessário, a conversão dos dados quando pedimos valores inteiros ou decimais:

```
#Construção do Menu
print("*****Inserir Produto ***** \n")

nome = input("Nome: ")
descricao = input("Descrição: ")
preco = float(input("Preco: "))
iva = float(input("IVA: "))
ativo = bool(input("Ativo: "))
```

No entanto, podemos aperfeiçoar os dados a serem pedidos (para tentar minimizar os erros por parte do utilizador. Como tal, vamos fazer a construção de cada campo que pedimos de forma mais cuidada:

Passo 2.1 – Construção do campo Nome

Neste exemplo, vamos pedir a introdução de texto, mas com a restrição de ter mais que 3 caracteres. Como tal, vamos utilizar a seguinte estrutura:

- **Estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha escrito mais do que 3 caracteres;
    - **Receber os dados** textuais com a **função input()**;

- **Estrutura de decisão dupla** para verificar o comprimento do valor (função len()) e:
  - **Se este for inferior ou igual que 3**, dar a mensagem de erro "Nome deve ter mais do que 3 caracteres";
  - **Senão, quebrar** o ciclo da **estrutura de repetição** com a **instrução break** e avançar para o próximo passo;

```
while True:
    nome = input("Nome: ")

    if len(nome) <= 3:
        print("Nome deve ter mais do que 3 caracteres")
    else:
        break
```

Passo 2.2 – Construção do campo Descrição

Neste exemplo, vamos pedir a introdução de texto, mas com a restrição de ter mais que 3 caracteres. Como tal, vamos utilizar a seguinte estrutura:

- **Estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha escrito mais do que 3 caracteres;
  - **Receber os dados** textuais com a **função input()**;
  - **Estrutura de decisão dupla** para verificar o comprimento do valor (função len()) e:
    - **Se este for inferior ou igual que 3**, dar a mensagem de erro "Nome deve ter mais do que 3 caracteres";
    - **Senão, quebrar** o ciclo da **estrutura de repetição** com a **instrução break** e avançar para o próximo passo;

```
while True:
    descricao = input("Descrição: ")

    if len(descricao) <= 3:
        print("Descrição deve ter mais do que 3 caracteres")
    else:
        break
```

Passo 2.3 – Construção do campo Preço

Neste exemplo, vamos pedir a introdução do preço do produto, mas com a restrição de ter apenas números. Como tal, vamos utilizar a seguinte estrutura:

- **Estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha introduzido conteúdo numérico válido, mais especificamente:
  - **Construção do bloco Try-Except**
    - **Bloco do Try:**
      - **Deverá receber os dados** textuais com a **função input()**;
      - **Converter o valor recebido** para decimal com a **função float();**
    - **Bloco Except:**
      - **Caso ocorra exceção nesta zona (pois o utilizador pode ter metido valores diferentes de números),** devemos dar a mensagem de erro "Deve introduzir apenas números";

```
while True:
    try:
        preco = float(input("Preco: "))
        break
    except ValueError:
        print("Deve introduzir apenas números")
```

Passo 2.4 – Construção do campo IVA

Neste exemplo, vamos pedir a introdução do iva do produto, mas com a restrição de ter apenas números. Como tal, vamos utilizar a seguinte estrutura:

- **Estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha introduzido conteúdo numérico válido, mais especificamente:
  - **Construção do bloco Try-Except**
    - **Bloco do Try:**
      - **Deverá receber os dados** textuais com a **função input()**;
      - **Converter o valor recebido** para decimal com a **função float();**

- **Bloco Except:**
  - **Caso ocorra exceção nesta zona (pois o utilizador pode ter metido valores diferentes de números),** devemos dar a mensagem de erro "Deve introduzir apenas números";

```
while True:
    try:
        iva = int(input("IVA: "))
        break
    except ValueError:
        print("Deve introduzir apenas números")
```

Passo 2.5 – Construção do campo Ativo

Neste exemplo, vamos pedir qual o estado do produto, mas com a restrição de ter apenas os números 1 (Verdadeiro) ou 0 (Falso). Como tal, vamos utilizar a seguinte estrutura:

- **Estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha introduzido conteúdo numérico válido, mais especificamente:
  - **Construção do bloco Try-Except**
    - **Bloco do Try:**
      - **Deverá receber os dados** textuais com a **função input()**;
      - **Converter o valor recebido** para inteiro com a **função int();**
      - **Com uma estrutura de decisão seletiva:**
        - **Caso o valor seja 0 ou 1,** deve quebrar a estrutura de repetição com a instrução **break;**
        - **Todos os outros casos,** dar a mensagem de erro **"Deve introduzir apenas 1 ou 0"**
    - **Bloco Except:**
      - **Caso ocorra exceção nesta zona (pois o utilizador pode ter metido valores diferentes de números),** devemos dar a mensagem de erro "Deve introduzir apenas 1 ou 0";

```
while True:
    try:
        ativo = int(input("Ativo: "))

        match ativo:
            case 0 | 1: break
            case _ : print("Deve introduzir apenas 1 ou 0")
    except ValueError:
        print("Deve introduzir apenas 1 ou 0")
```

Passo 3 – Criar uma variável chamada **query** (nome sugestivo), do qual, vai conter a preparação da instrução para inserir os dados na base de dados. Na opção dos valores, vamos colocar a instrução %s para indicar que vamos receber um determinado valor e que será definido no próximo passo:

```
#Inserir dados na BD
query = "INSERT INTO Produtos(Nome, Descricao, Preco, IVA, Ativo) " \
"VALUES (%s, %s, %s, %s, %s)"
```

Passo 4 – Criar uma variável chamada **valores** (nome sugestivo), e dentro de parênteses, vamos colocar os valores que queremos associar a cada %s (que neste caso, será os valores das variáveis que guardaram os dados no passo 2 e com os valores separados por vírgulas):

```
#Associar os valores por cada %s da query
valores = (nome, descricao, preco, iva, ativo)
```

Passo 5 – Ainda falta fazer a transação na base de dados com os valores dos passos 3 e 4. No entanto, vamos ao ficheiro "operacoesBD.py" e vamos:

- Importar as bibliotecas conexaoDB para conseguir interligar com a base de dados e o driver de conexão mysql.connector:

```
from conexaoBD import conexao_MySQL
import mysql.connector
```

- Criar uma função específica para inserir dados com o nome **executar_inserir** que recebe dois parâmetros de entrada (neste caso a query com a preparação e os valores para associar a cada %s o valor correspondente:

```python
def executar_inserir(query, valores):
    try:
        #Abrir a conexao com a base de dados
        conexao = conexao_MySQL()

        # dictionary=True -> permite colocar os nomes dos campos
        # (em vez de ficar apenas os valores)
        cursor = conexao.cursor(dictionary=True)

        #Executar a query com os valores
        cursor.execute(query, valores)

        #Confirmar a operação da inserção no MySQL
        conexao.commit()

        #Devolver qual foi o ID atribuido
        return print("Registo Inserido com o ID: ", cursor.lastrowid)

    except mysql.connector.Error as erro:
        print("Erro ao inserir dados", erro)
    finally:
        #Se a conexão estiver ainda aberta, deverá fechar
        if conexao.is_connected():
            cursor.close()
            conexao.close()
```

Passo 6 – Retorne ao ficheiro "inserir.py" e depois do código feito no passo 4, deve colocar a instrução para invocar a função **executar_inserir()** e passar dos parâmetros de entrada (neste caso a query com a preparação e os valores para associar a cada %s o valor correspondente):

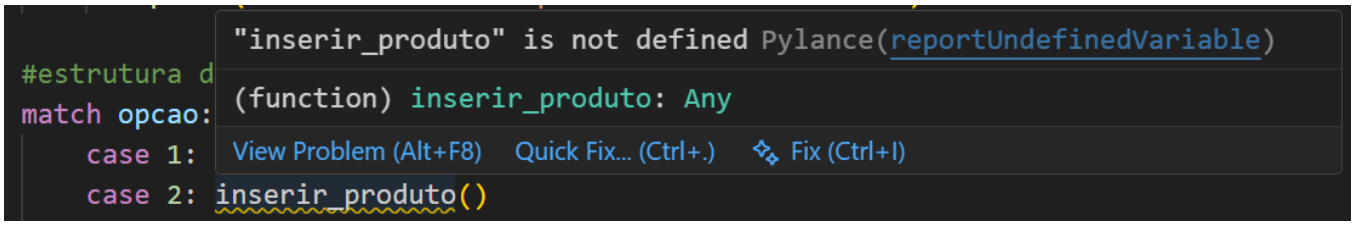
```python
#Executar a query com os valores
executar_inserir(query, valores)
```

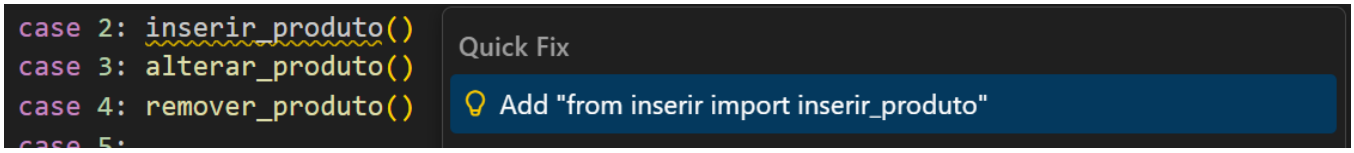
Passo 7 – Concluído os passos anteriores, vamos agora associar a execução da função inserir_produto **no menu principal do programa**. Como tal, deve ir a estrutura de decisão seletiva que avalia as opções do menu e **colocar no caso 2 a invocação do nome da função**:

```python
case 2: inserir_produto()
```

No entanto, a função pode estar com o sublinhado amarelo, pois falta importar a localização do ficheiro no cabeçalho. Como tal, pode usar as ferramentas de autoajuda, bastando colocar o rato em cima do

nome da função e será exibida uma nova janela. Clique nas opções "**Quick Fix…**" e selecione a opção para importar o ficheiro com a função que pretende associar nas escolhas do menu:





Resultado:

Parte 5 – Listar dados + (operacoesBD.py)

O objetivo desta funcionalidade será listar os dados da base de dados com a ajuda de uma biblioteca do python conhecida por Pretty Table (página do projeto: https://pypi.org/project/prettytable/). O objetivo será fazer query na base de dados e criar uma tabela com os dados obtidos. Exemplo:

```
Nº de registos na tabela:  1
+----+-------+------------+-------+-----+-------+
| ID |  Nome | Descricao  | Preco | IVA | Ativo |
+----+-------+------------+-------+-----+-------+
| 1  | Doces | Chocolates | 10.52 |  23 |   1   |
+----+-------+------------+-------+-----+-------+
```

Passo 1 – No terminal do Visual Studio, vamos instalar a biblioteca do Pretty Table com a seguinte instrução: **pip install -U prettytable**

Passo 2 – No ficheiro "operacoesDB.py", deve importar a biblioteca do pacote que instalou no passo anterior:

```
1   from conexaoBD import conexao_MySQL
2   import mysql.connector
3   from prettytable import PrettyTable
```

Passo 3 – Ainda no mesmo ficheiro vamos criar uma nova função com o nome listar(). Esta função deverá receber um parâmetro de entrada (neste caso, vamos assumir o nome id e que vai ser igualado a zero, caso não receba qualquer valor):

```
def listar(id = 0):
```

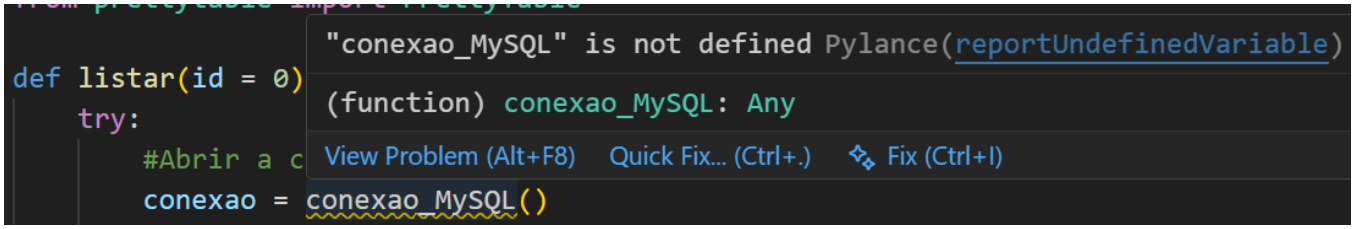
Dentro da função, vamos começar por criar o bloco de instruções Try e dentro deste fazer as seguinte operações:

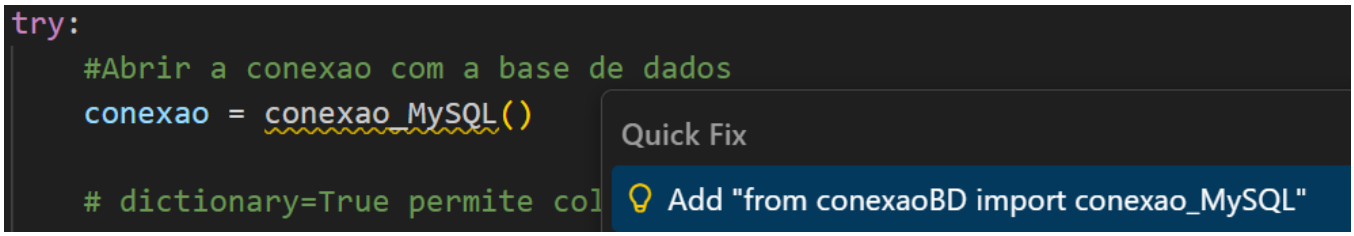
Passo 3.1 – Criar uma variável com o nome **conexao** (nome sugestivo) que vai invocar a conexão com a base de dados:

```
try:
    #Abrir a conexao com a base de dados
    conexao = conexao_MySQL()
```

Nota Importante: se a função conexao_MySQL ficar com o tracejado amarelo, significa que ainda falta importar essa informação do ficheiro "conexaoDB.py".

Como tal, pode usar as ferramentas de autoajuda, bastando colocar o rato em cima do nome da função e será exibida uma nova janela. Clique nas opções "**Quick Fix…**" e selecione a opção para importar o ficheiro com a função que pretende associar nas escolhas do menu:





Passo 3.2 – Criar uma variável com o nome **cursor** (nome sugestivo). Logo a seguir, vamos passar o valor da conexão e invocar a função cursor. Em Python, um cursor MySQL é o objeto utilizado para executar comandos SQL e ler resultados de uma base de dados. No entanto, se utilizar apenas o cursor de forma simples, vamos apenas ter os valores da tabela:

```
cursor = conexao.cursor()
```

Resultado:

```
[(1, 'Doces', 'Chocolates', Decimal('10.52'), 23, 1)]
```

Mas, o objetivo principal é implementar o nome dos campos por cada valor apresentado ficando num formato par chave-valor, ou seja, nome do campo e o valor.

Como tal, **dentro** dos parênteses da função **cursor()**, vamos colocar a instrução **dictionary=True**, do qual, vai permitir incluir os nomes dos campos por cada valor apresentado:

```
# dictionary=True permite colocar os nomes dos campos
# (em vez de ficar apenas os valores)
cursor = conexao.cursor(dictionary=True)
```

Resultado:

```
[{'ID_Produto': 1, 'Nome': 'Doces', 'Descricao': 'Chocolates', 'Pr
```

Deste modo, será mais fácil retirar a informação pelo nome do campo, do que pela posição.

Passo 3.3 – Criar uma estrutura de decisão dupla para verificar qual a consulta a ser executada na base de dados, em que:

- **Se o id não tiver nada e continuar com o valor 0:**
  - Pedir ao cursor para executar a query para devolver todos os registos da tabela;
- **Senão, significa que a função listar recebeu algum valor no id e nesse caso:**
  - Pedir ao cursor para executar a query para devolver apenas o registo com o id que recebeu (e se encontrar o registo pelo id, irá devolver os dados);

```
if(id == 0):
    cursor.execute("SELECT * FROM produtos")
else:
    cursor.execute(f"SELECT * FROM produtos WHERE ID_Produto = {id}")
```

Passo 3.4 – Criar uma variável com o nome **data** (nome sugestivo), passar o valor do cursor que foi executado no passo anterior e pedir para obter todos registos devolvidos na consulta na base de dados com a função **fetchall()**:

```
data = cursor.fetchall()
```

Passo 3.5 – A seguir, vamos criar as seguintes instruções:

- Uma estrutura de decisão simples para verificar se a variável data tem algum registo com a função len();
  - Mostrar quantos registos é que encontrou com a instrução **cursor.rowcount**;
  - Utilizar a biblioteca PrettyTable para criar uma tabela com os dados que vem da base de dados:
    - Criar a variável table (nome sugestivo) para invocar a biblioteca e a estrutura de uma tabela;
    - Criar o cabeçalho da tabela com a propriedade field_names e logo a seguir, deve criar uma lista com o nome das colunas da tabela;
    - Percorrer cada registo da tabela (neste caso a tabela Produtos) e colocar cada linha de informação na tabela do PrettyTable com a função add_row(), do qual deve indicar o nome do campo a resgatar a informação, por forma a encaixar a informação nas colunas que foram criados anteriormente;
    - No final, mostrar a tabela:

```
if len(data) > 0:
    print("Nº de registos na tabela: ", cursor.rowcount)

    #criar a estrutura da tabela
    table = PrettyTable()

    #Criar o Cabeçalho da Tabela
    table.field_names = ["ID","Nome","Descricao","Preco","IVA","Ativo"]

    #Percorrer cada registo da tabela Produtos e colocar na tabela
    for item in data:
        table.add_row([item["ID_Produto"], item["Nome"], item["Descricao"],
                       item["Preco"], item["IVA"], item["Ativo"]])

    print(table)

return cursor.rowcount
```

Resultado:

```
Nº de registos na tabela:  1
+----+-------+------------+-------+-----+-------+
| ID |  Nome | Descricao  | Preco | IVA | Ativo |
+----+-------+------------+-------+-----+-------+
| 1  | Doces | Chocolates | 10.52 |  23 |   1   |
+----+-------+------------+-------+-----+-------+
```

Passo 3.6 – Bloco catch (tratamento dos potenciais erros) e finnally (operações de finalização):

- **Exceção**: mysql.connector.Error
  - Será gerado caso não seja possível ler os dados da tabela;
- **Finalização**:
  - Será a conexão estiver ainda aberta, deve fechar a mesma;

```
except mysql.connector.Error as erro:
    print("Erro ao ler dados da tabela Produtos", erro)
finally:
    if conexao.is_connected():
        cursor.close()
        conexao.close()
```

Parte 6 – Ficheiro eliminar dados (eliminar.py) + (operacoesBD.py)

Este ficheiro vai servir para colocar as instruções para eliminar dados na tabela da base de dados.

Passo 1 – Criar o ficheiro "eliminar.py" e no início do ficheiro, vamos importar:

- o ficheiro **conexaoDB** para conseguir interligar com a base de dados e:
- a biblioteca do driver de conexão **mysql.connector**;

```
1   from conexaoBD import conexao_MySQL
2   import mysql.connector
```

- o ficheiro das operacoesBD.py, mais especificamente, a função listar (para usar mais tarde):

```
from operacoesBD import listar
```

Passo 2 – Vamos criar a função com o nome **remover_produto()**. Este por sua vez vai ser invocado mais tarde no programa principal por forma a fazer interligações entre menus:

```
def remover_produto():
```

Passo 3 – No início da função, vamos declarar uma variável com o nome "id" com valor 0, e de seguida, deve utilizar uma **estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha introduzido um valor inteiro válido (que será para pesquisar o valor na base de dados), mais especificamente:

- **Construção do bloco Try-Except**
  - **Bloco do Try:**
    - **Deverá receber o valor para a variável id** com a **função input()**
    - **Converter o valor recebido** para inteiro com a **função int();**
    - **Quebrar a estrutura se tudo correr bem com a instrução break**
  - **Bloco Except:**
    - **Caso ocorra exceção nesta zona (pois o utilizador pode ter metido valores diferentes de números),** devemos dar a mensagem de erro "Deve introduzir apenas números";

```
print("*****Remover Produto ***** \n")
id = 0

while True:
    try:
        id = int(input("Insira o ID a procurar: "))
        break
    except ValueError:
        print("Deve introduzir apenas números inteiros")
```

Passo 4 – Depois de obter o id, vamos realizar uma consulta na base de dados e verificar se existe o valor a pesquisar.

Passo 4.1 – Deve colocar a instrução dentro do bloco try e dentro deste, criar a variável data para invocar a função listar e dentro dos parenteses informar qual o valor do id que pesquisou:

```
#Consultar informação na BD
try:
    #Procurar dados pelo id
    data = listar(id)
```

Anteriormente, no ficheiro das "operacoesBD.py", foi criada uma estrutura de decisão dupla para verificar qual a consulta a ser executada na base de dados, em que:

- **Se o id não tiver nada e continuar com o valor 0:**
  - Pedir ao cursor para executar a query para devolver todos os registos da tabela;
- **Senão, significa que a função listar recebeu algum valor no id e nesse caso:**
  - Pedir ao cursor para executar a query para devolver apenas o registo com o id que recebeu (e se encontrar o registo pelo id, irá devolver os dados);

```
if(id == 0):
    cursor.execute("SELECT * FROM produtos")
else:
    cursor.execute(f"SELECT * FROM produtos WHERE ID_Produto = {id}")
```

Neste caso, vai ser executada a instrução contida no else, pois queremos fazer uma pesquisa por um valor específico e que seja devolvida apenas uma informação.

Passo 4.2 – A seguir, vamos realizar a construção de uma estrutura de decisão dupla para verificar duas situações:

- Se o que vier da função listar tiver informações, vamos realizar as seguintes operações:
  - Mostrar a mensagem que encontrou o registo e questionar o utilizador se deseja remover a informação encontrada:
    - Se o valor for 1, significa que desejamos remover.

      Antes de prosseguir com este código temos de ir até ao ficheiro "conexoesBD.py" e criar a função que permite eliminar dados da base de dados. Nesse ficheiro, deve criar a seguinte função:

```
def executar_remover(query):
    #Inserir dados na BD
    try:
        conexao = conexao_MySQL()

        # dictionary=True permite colocar os nomes dos campos
        # (em vez de ficar apenas os valores)
        cursor = conexao.cursor(dictionary=True)

        #Executar a query com os valores
        cursor.execute(query)
        conexao.commit()

    except mysql.connector.Error as erro:
        print("Erro ao eliminar dados", erro)
    finally:
        if conexao.is_connected():
            cursor.close()
            conexao.close()
```

      Volte ao ficheiro do "remover.py", e vamos:

      - Importar a nova função para dentro do ficheiro:

```
from operacoesBD import executar_remover, listar
```

      - utilizar a função executar_remover do ficheiro "operacoesBD.py" e passar a instrução SQL para remover a informação da tabela pelo ID fornecido;
    - Senão, deve exibir a mensagem de operação cancelada;
- Senão, deve exibir a mensagem de que não foi possível encontrar o id que procurou;

```
if data > 0:
    print(f"Foi encontrado o registo com o ID {id}.")
    print(f"Deseja eliminar o produto {id}? ")

    confirma = int(input("(1 - Sim | 0 - Não) : "))

    if confirma == 1:
        executar_remover(f"DELETE FROM produtos WHERE ID_Produto = {id}")
        print("Registo apagado!")
    else:
        print("Operação cancelada!")
        input("Prima qualquer tecla para continuar... ")
else:
    print(f"Não foi possivel encontrar o Produto com o {id}")
    input("Prima qualquer tecla para continuar... ")
```

Passo 4.3 – No final do bloco try, vamos utilizar o bloco except para tratar de potenciais exceções:

```
except mysql.connector.Error as erro:
    print(f"Erro ao eliminar ID Produto {id}", erro)
```

Parte 7 – Ficheiro alterar dados (alterar.py) + (operacoesBD.py)

Este ficheiro vai servir para colocar as instruções para alterar dados na tabela na base de dados.

Passo 1 – Criar o ficheiro "alterar.py" e no início do ficheiro, vamos importar:

- o ficheiro **conexaoDB** para conseguir interligar com a base de dados e:
- a biblioteca do driver de conexão **mysql.connector**;

```
1   from conexaoBD import conexao_MySQL
2   import mysql.connector
```

- o ficheiro das operacoesBD.py, mais especificamente, a função listar:

```
from operacoesBD import listar
```

Passo 2 – Vamos criar a função com o nome **remover_produto()**. Este por sua vez vai ser invocado mais tarde no programa principal por forma a fazer interligações entre menus:

```
def alterar_produto():
```

Passo 3 – No início da função, vamos declarar uma variável com o nome "id" com valor 0, e de seguida, deve utilizar uma **estrutura de repetição While** para repetir o bloco de instruções até que o utilizador tenha introduzido um valor inteiro válido (que será para pesquisar o valor na base de dados), mais especificamente:

- **Construção do bloco Try-Except**
  - **Bloco do Try:**
    - **Deverá receber o valor para a variável id** com a **função input()**
    - **Converter o valor recebido** para inteiro com a **função int();**
    - **Quebrar a estrutura se tudo correr bem com a instrução break**
  - **Bloco Except:**

- **Caso ocorra exceção nesta zona (pois o utilizador pode ter metido valores diferentes de números),** devemos dar a mensagem de erro "Deve introduzir apenas números";

```
print("*****Remover Produto ***** \n")
id = 0

while True:
    try:
        id = int(input("Insira o ID a procurar: "))
        break
    except ValueError:
        print("Deve introduzir apenas números inteiros")
```

Passo 4 – Depois de obter o id, vamos realizar uma consulta na base de dados e verificar se existe o valor a pesquisar.

Passo 4.1 – Criar a variável data para invocar a função listar e dentro dos parenteses informar qual o valor do id que pesquisou:

```
#Consultar informação na BD
data = listar(id)
```

Anteriormente, no ficheiro das "operacoesBD.py", foi criada uma estrutura de decisão dupla para verificar qual a consulta a ser executada na base de dados, em que:

- **Se o id não tiver nada e continuar com o valor 0:**
  - Pedir ao cursor para executar a query para devolver todos os registos da tabela;
- **Senão, significa que a função listar recebeu algum valor no id e nesse caso:**
  - Pedir ao cursor para executar a query para devolver apenas o registo com o id que recebeu (e se encontrar o registo pelo id, irá devolver os dados);

```
if(id == 0):
    cursor.execute("SELECT * FROM produtos")
else:
    cursor.execute(f"SELECT * FROM produtos WHERE ID_Produto = {id}")
```

Neste caso, vai ser executada a instrução contida no else, pois queremos fazer uma pesquisa por um valor específico e que seja devolvida apenas uma informação.

Passo 4.2 – A seguir, vamos realizar a construção de uma estrutura de decisão dupla para verificar duas situações:

- Se o que vier da função listar tiver informações, vamos realizar as seguintes operações:
  - Mostrar a mensagem que encontrou o registo;

```
if data > 0:
    print(f"Foi encontrado o registo com o ID {id}.")
```

  - Criar uma estrutura de repetição while e mostrar um menu para o utilizador puder escolher qual o campo que deseja alterar e pedir a opção ao utilizador (validando a opcao inserida e aceitar somente números):

```
while True:
    print("Qual a alteração que deseja fazer? ")
    print("1 - Nome")
    print("2 - Descrição")
    print("3 - Preço")
    print("4 - IVA")
    print("5 - Ativo")
    print("6 - Voltar atrás")

    #Pedir a opção ao utilizador
    while True:
        try:
            opcao = int(input("Insira a opção: "))
            break
        except ValueError:
            print("Deve introduzir apenas números inteiros")
```

  - Criar a estrutura de decisão seletiva, do qual, deve avaliar a opção escolhida e realizar a operação mais adequada:

```
#estrutura de decisao seletiva
match opcao:
```

Antes de prosseguir com este código temos de ir até ao ficheiro "conexoesBD.py" e criar a função que permite alterar dados da base de dados. Nesse ficheiro, deve criar a seguinte função:

```
def executar_alterar(query, valores):
    #Inserir dados na BD
    try:
        conexao = conexao_MySQL()

        # dictionary=True permite colocar os nomes dos campos
        # (em vez de ficar apenas os valores)
        cursor = conexao.cursor(dictionary=True)

        #Executar a query com os valores
        cursor.execute(query, valores)
        conexao.commit()

        #Devolver qual foi o ID atribuido
        return print("Registo Alterado")

    except mysql.connector.Error as erro:
        print("Erro ao alterar dados", erro)
    finally:
        if conexao.is_connected():
            cursor.close()
            conexao.close()
```

Passo 4.2 (continuação) – Feito o passo anterior, podemos criar os casos/opões para a estrutura de decisão seletiva.

- Caso escolha a **opção 1**, vamos fazer a alteração do campo **Nome**:

```
case 1:
    while True:
        nome = input("Nome: ")

        if len(nome) <= 3:
            print("Nome deve ter mais do que 3 carateres")
        else:
            break

    #Alterar dados na BD
    query = "UPDATE produtos SET nome = %s WHERE ID_Produto = %s"

    #Associar os valores por cada %s da query
    valores = (nome, id)
    #executar a alteração na BD com a função executar_alterar()
    executar_alterar(query, valores)
```

- Caso escolha a **opção 2**, vamos fazer a alteração do campo **Descrição**:

```
case 2:
    while True:
        descricao = input("Descrição: ")

        if len(descricao) <= 3:
            print("Descrição deve ter mais do que 3 carateres")
        else:
            break

    #Alterar dados na BD
    query = "UPDATE produtos SET descricao = %s WHERE ID_Produto = %s"

    #Associar os valores por cada %s da query
    valores = (descricao, id)
    #executar a alteração na BD com a função executar_alterar()
    executar_alterar(query, valores)
```

- Caso escolha a **opção 3**, vamos fazer a alteração do campo **Preço**:

```
case 3:
    while True:
        try:
            preco = float(input("Preço: "))
            break
        except ValueError:
            print("Deve introduzir apenas números")

    #Alterar dados na BD
    query = "UPDATE produtos SET preco = %s WHERE ID_Produto = %s"

    #Associar os valores por cada %s da query
    valores = (preco, id)
    #executar a alteração na BD com a função executar_alterar()
    executar_alterar(query, valores)
```

- Caso escolha a **opção 4**, vamos fazer a alteração do campo **IVA**:

```
case 4:
    while True:
        try:
            iva = float(input("IVA: "))
            break
        except ValueError:
            print("Deve introduzir apenas números")

    #Alterar dados na BD
    query = "UPDATE produtos SET iva = %s WHERE ID_Produto = %s"

    #Associar os valores por cada %s da query
    valores = (iva, id)
    #executar a alteração na BD com a função executar_alterar()
    executar_alterar(query, valores)
```

- Caso escolha a **opção 5**, vamos fazer a alteração do campo **Ativo**:

```
case 5:
    while True:
        try:
            ativo = float(input("Ativo: "))
            break
        except ValueError:
            print("Deve introduzir apenas números")

    #Alterar dados na BD
    query = "UPDATE produtos SET ativo = %s WHERE ID_Produto = %s"

    #Associar os valores por cada %s da query
    valores = (ativo, id)
    #executar a alteração na BD com a função executar_alterar()
    executar_alterar(query, valores)
```

- Caso escolha a **opção 6**, vamos sair do menu atual e voltar para o menu anterior:

```
case 6:
    break
```

- Todos as outras opções, mostrar a mensagem “Opção inválida”:

```
case _: print("Opção inválida")
```

Passo 4.3 – Feche a sequência da instrução if com o else para mostrar a seguinte mensagem, caso não encontre o produto que procurou:

```
else:
    print(f"Não foi possivel encontrar o Produto com o {id}")
```