Disciplina:
Programação de Sistemas

Python e Flask

Marco Meneses e Paulo Ferreira

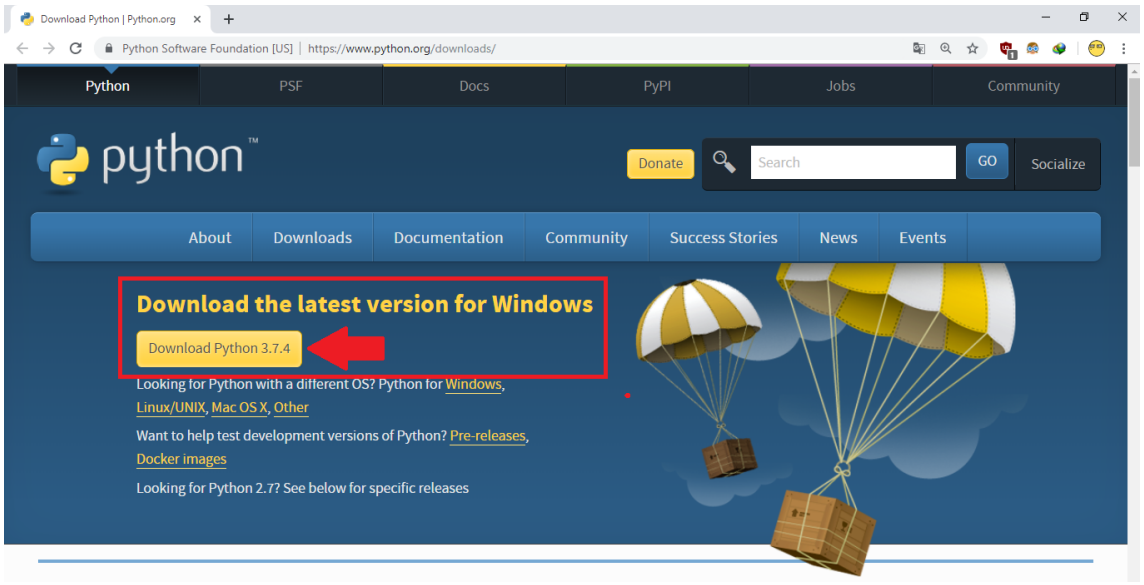
Última revisão – 01/09/2022

Índice

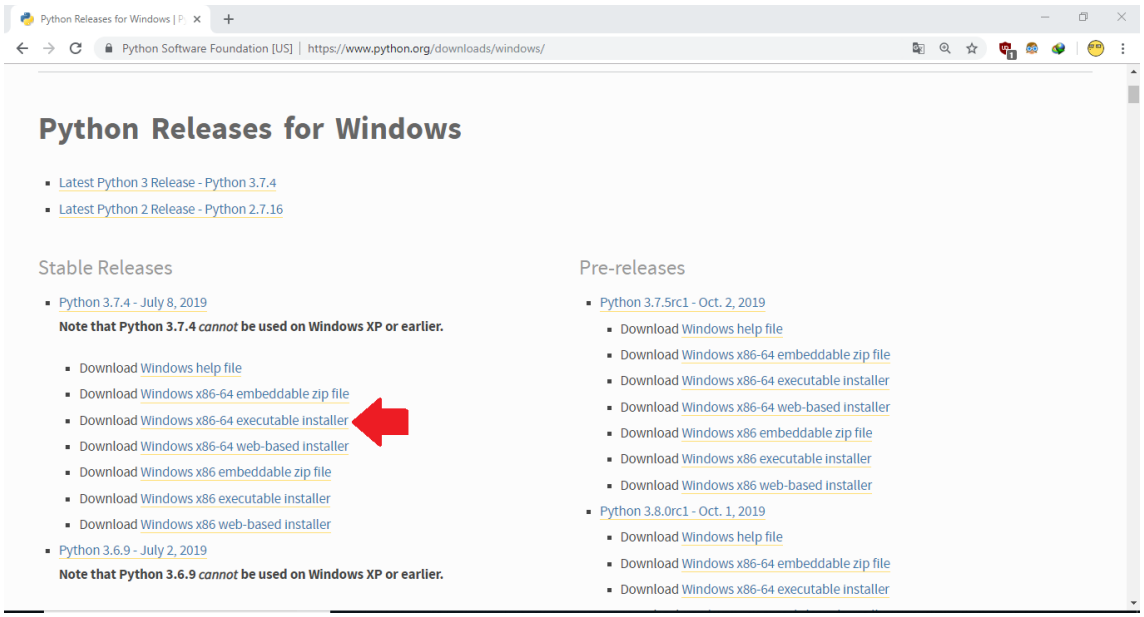
Instalação do Python

Instalando o Python 3 no Windows:

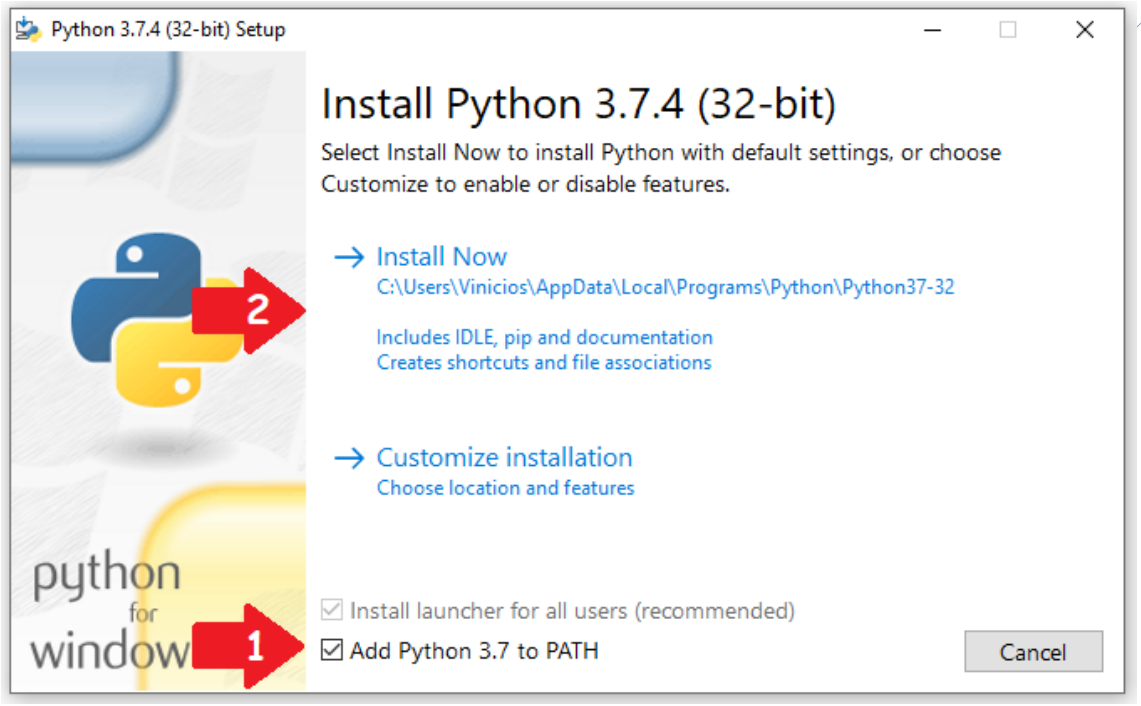
Para instalar o Python no seu computador, você precisa descarregar o instalador. Aceda ao site oficial a este link e clique em download, como mostrado abaixo.



Isso fará o download do Python 3 para todos os tipos de sistemas (64 bits). Para outro tipo de sistema, aceda a este link e selecione o instalador de 64 bits apropriado, como mostrado em baixo.



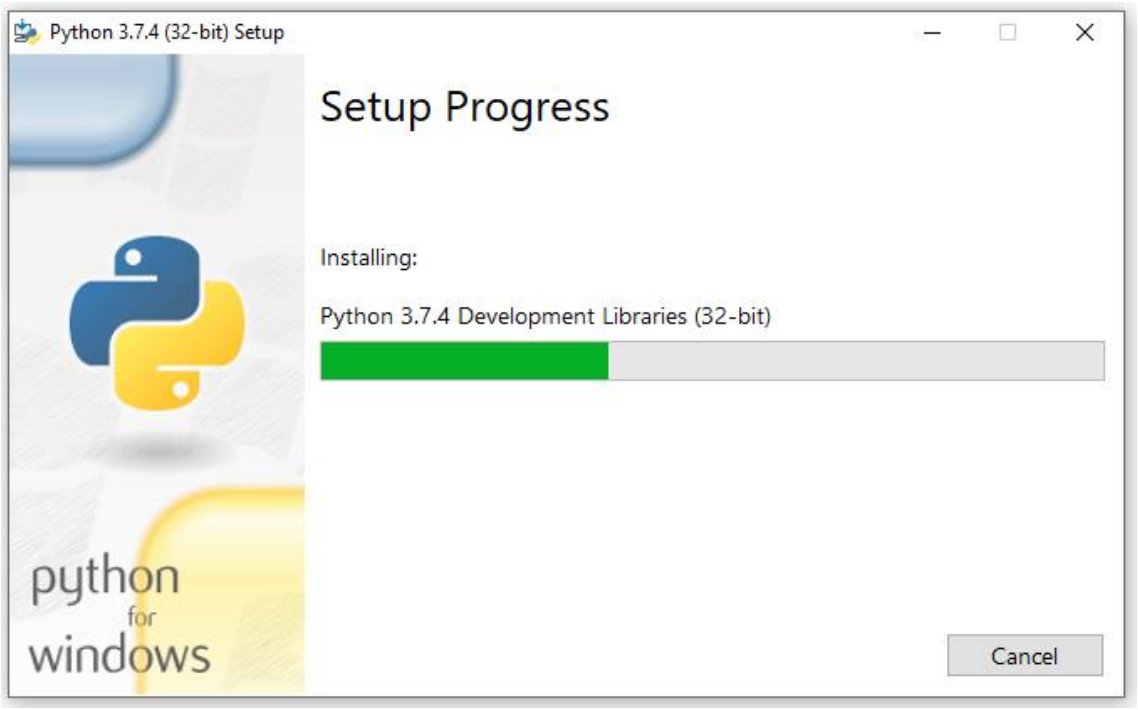
Faça a descarga do instalador executável e clique duas vezes sobre o ficheiro para iniciar o assistente de instalação do Python, como mostrado abaixo.



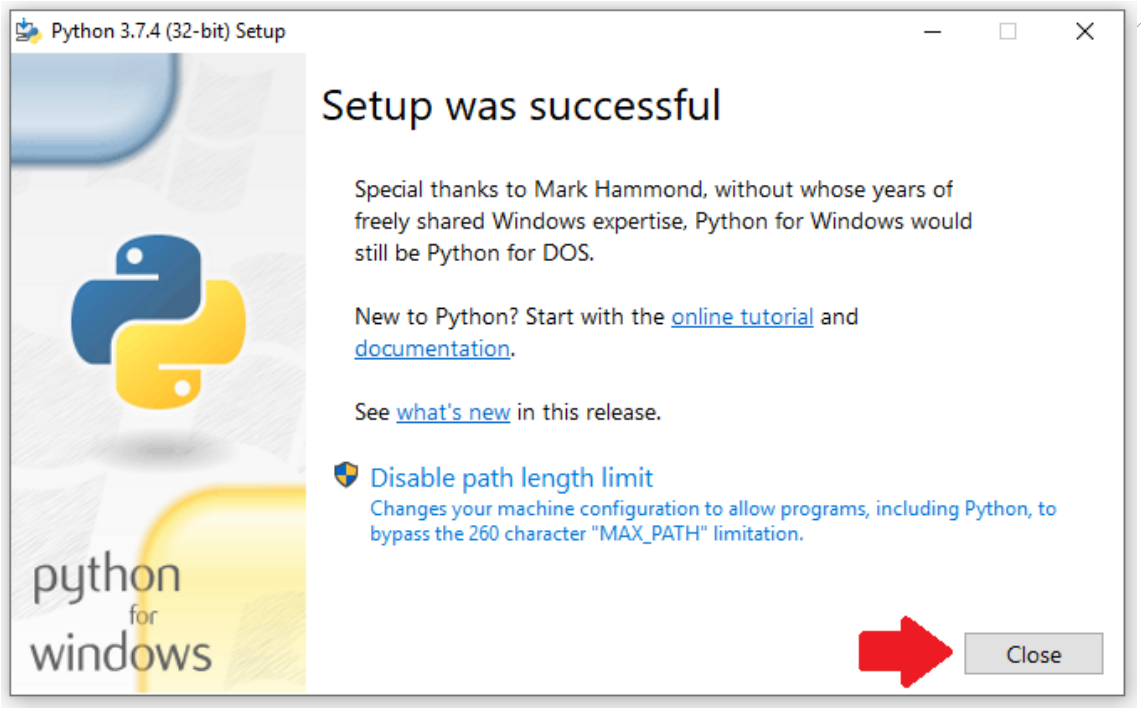
O processo de instalação é bem simples:

1. Marque a opção "Add Python to PATH"
    a. Faz com que as pastas com os comandos do Python possam ser acedidas de qualquer sítio
2. Clique em "Install Now"

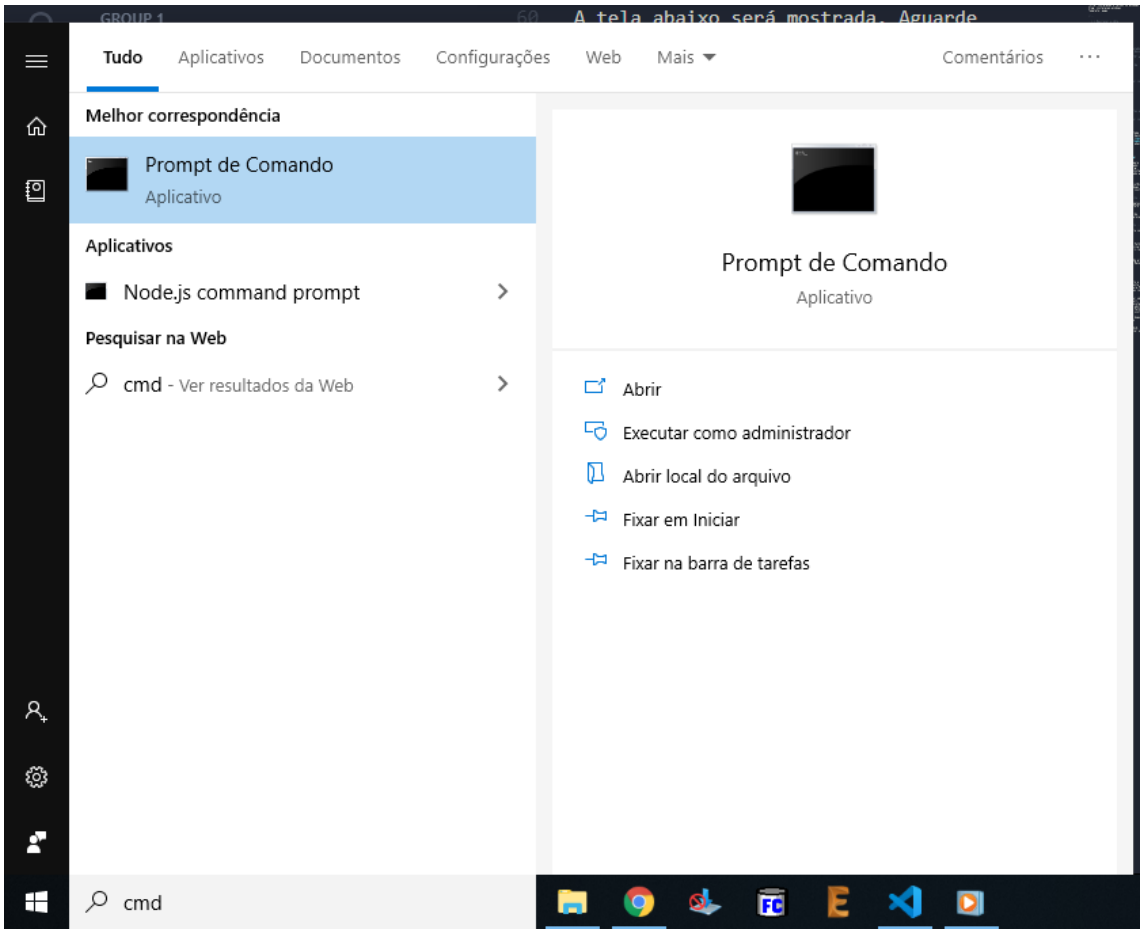
A janela abaixo será mostrada, devendo aguardar até que a instalação finalize o processo de instalação.



Se tudo ocorrer bem, a próxima janela será mostrada e clique em "Close".



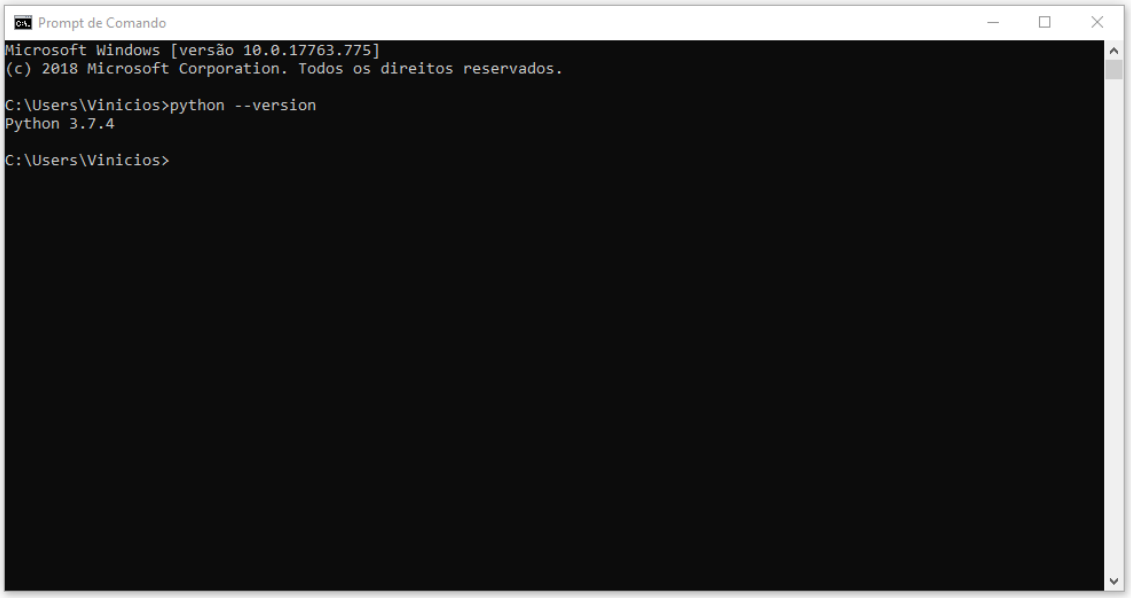
Para verificar se a instalação do Python foi bem-sucedida, abra a linha de comandos.



Digite o seguinte comando:

```
python --version
```

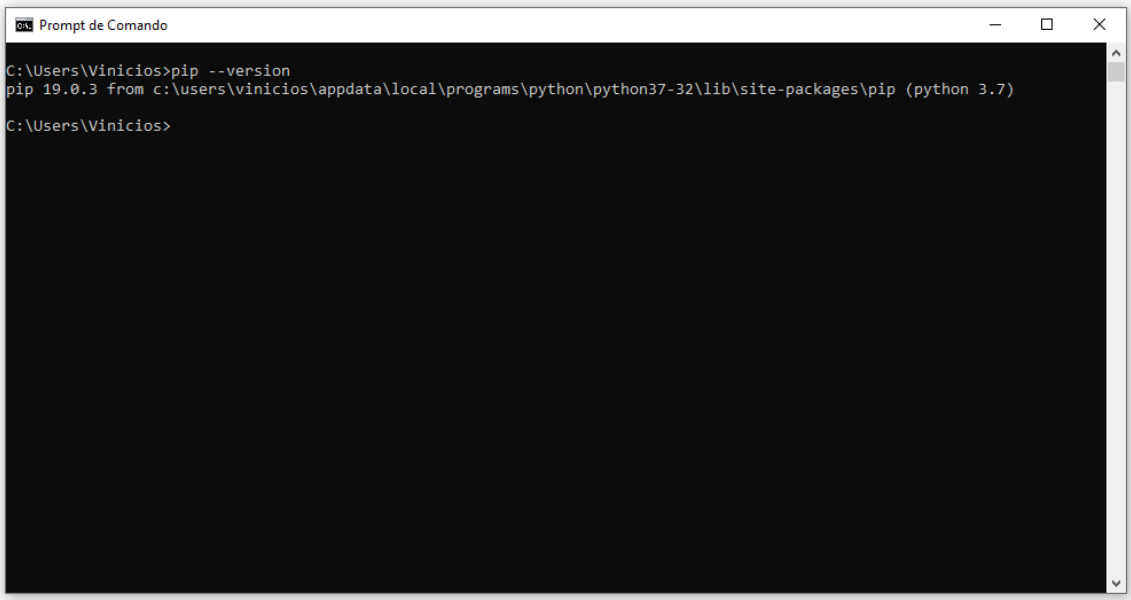
Mostrará a versão do Python que está instalada no seu computador.



Agora digite:

```
pip --version
```

Este comando mostrará a versão do pip que está instalada no computador. O pip é o gestor de pacotes do Python. Com este poderá adicionar novas funcionalidades ao Python.

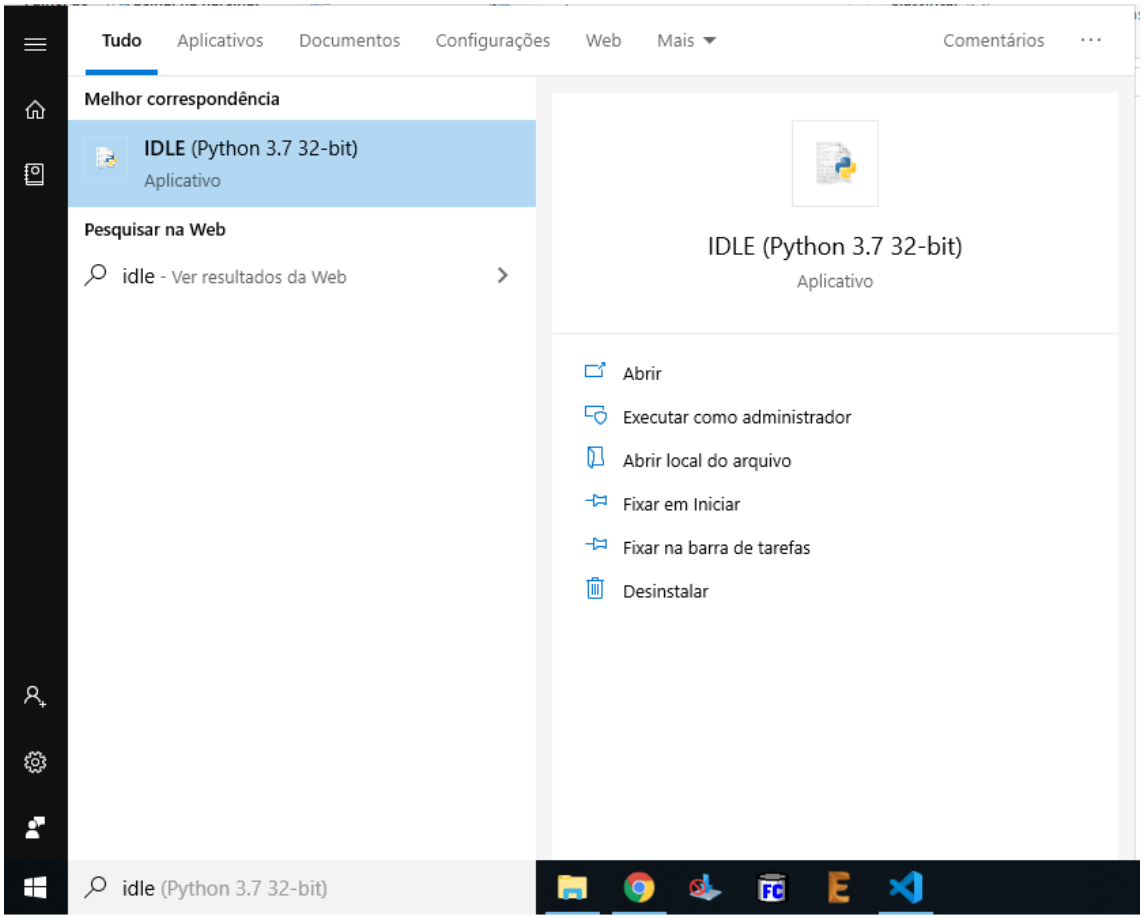


Idle

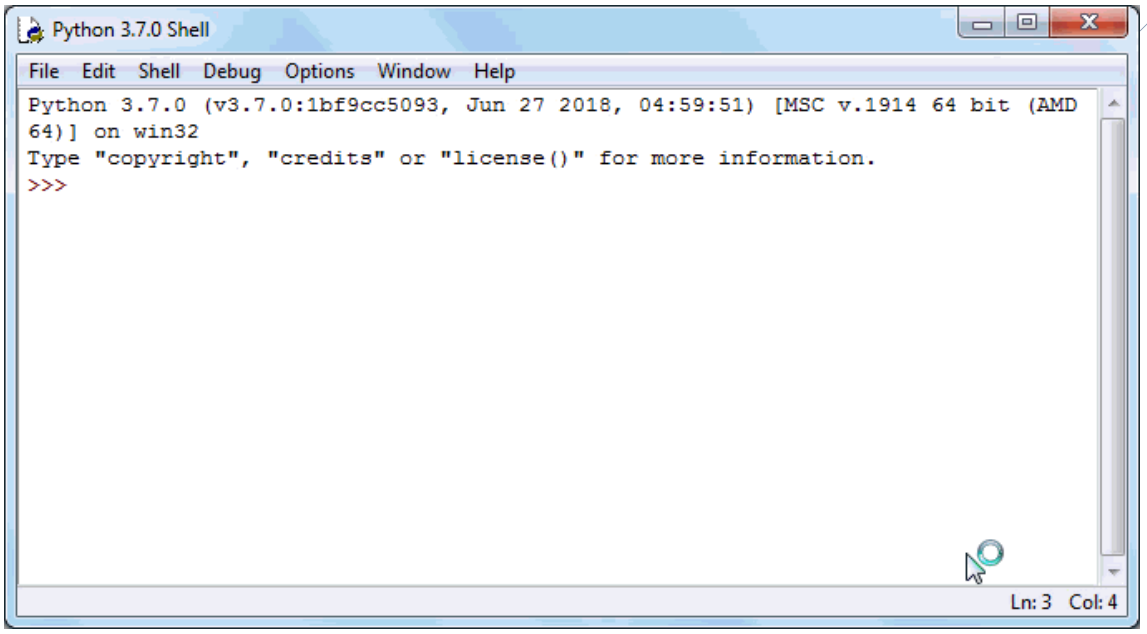
O IDLE (Ambiente de Desenvolvimento e Aprendizagem Integrado) é um ambiente de desenvolvimento integrado (IDE) para o Python. O instalador do Python para Windows contém o módulo IDLE por padrão.

O IDLE pode ser usado para executar uma única instrução, como o Python Shell, mas também para criar, modificar e executar scripts Python. O IDLE fornece um editor de texto completo para criar scripts Python que incluem recursos como destaque de sintaxe, preenchimento automático e recuo inteligente. Possui também, um depurador com recursos de etapas e pontos de interrupção.

Para iniciar o shell interativo IDLE, procure o ícone IDLE no menu Iniciar e clique sobre este.



Isso abrirá o IDLE, onde você pode escrever o código Python e executá-lo como mostrado abaixo.



Neste momento o Python, o pip e o Idle já estão instalados no seu sistema.

Instalação e configuração do Visual Studio Code

O Visual Studio Code é um editor de código criado pela Microsoft e que tem uma grande adoção pelas comunidades de diversas linguagens e tecnologias.

Requisitos

O Visual Studio Code tornou-se famoso por sua leveza na execução, sendo um dos editores de código mais leve e rápidos que temos no mercado.
Desta forma, os requisitos de hardware são baixos, porém recomendamos uma máquina com pelo menos Intel Core i3, Dual Core, com 8GB de Ram e no mínimo 128GB de disco. Estas recomendações são baseadas na execução dos nossos cursos, então no caso do seu cenário ser diferente, elas podem variar.
Recomendamos também manter o Sistema Operativo SEMPRE ATUALIZADO, e em caso de uso do Windows, utilizar sempre a versão 10 ou superior.

Instalação no Windows

A instalação no Windows é relativamente simples, basta aceder ao site oficial do Visual Studio Code e fazer descarga, neste link, da sua última versão, e para os diversos sistemas.
Em seguida, execute o instalador descarregado e siga as instruções da janela, não se esqueça de marcar a opção "Add to Path" para adicionar o Visual Studio Code nas variáveis de ambiente.
Finalizada a instalação, feche todos os terminais abertos e abra um novo (Power Shell ou Windows Terminal) e digite o comando abaixo.

```
code --version
```

Se tudo ocorrer como esperado, esta mostrará a versão do Visual Studio Code instalada no seu computador e estará tudo pronto para começar.

Instalação no Mac

A instalação no Mac é relativamente simples, basta acerder o site oficial do Visual Studio Code e fazer a descarga da sua última versão, neste link.
Abra o instalador e copie o Visual Studio Code para a pasta Applications do seu Mac e pronto, a instalação está devidamente feita.

Adicionando ao PATH

Podemos ainda adicionar o Visual Studio Code ao PATH para ter acesso ao comando code no terminal, podendo abrir diretamente pastas de lá no Visual Studio Code.
Abra seu menu de aplicativos e inicie o Visual Studio Code e na janela inicial pressione F1 para abrir a tela de execução de comandos.
Procure pelo comando Shell Command: Install 'code' command in PATH para adicionar o Visual Studio Code ao PATH do seu sistema operacional.

Instalação no Linux (Ubuntu)

A instalação no Linux é relativamente simples, basta aceder o site oficial do Visual Studio Code e fazer a descarga da sua última versão, neste link.
Como resultado da descarga terá um pacote .deb na pasta Downloads/Transferências.
Abra um terminal, navegue até a mesma e execute o seguinte comando.

```
sudo apt install ./FICHEIRO_DESCARREGADO.deb
```

Pronto, a instalação do Visual Studio Code está efetuada e devendo aparecer no menu de aplicativos.

Adicionando ao PATH

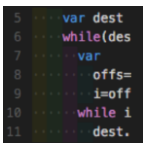
Podemos ainda adicionar o Visual Studio Code ao PATH para ter acesso ao comando code no terminal, podendo abrir diretamente pastas de lá no Visual Studio Code.
Abra seu menu de aplicativos e inicie o Visual Studio Code e na janela inicial pressione F1 para abrir a tela de execução de comandos.
Procure pelo comando Shell Command: Install 'code' command in PATH para adicionar o Visual Studio Code ao PATH do seu sistema operacional.

Extensões

As extensões são uma espécie de aplicativos complementares instalados dentro do Visual Studio Code que nos permite acesso a funcionalidades extras.
Existem muitas extensões já criadas e novas chegando a cada dia, porém vamos debater as utilizadas nas disciplinas do curso.
Com o Visual Studio Code aberto, pressione CTRL (CMD)+SHIFT+X ou localize o ícone de extensões no menu lateral direito da janela.

Uma nova janela se abrirá, para pesquisarmos as extensões, e vamos instalar as seguintes:

- Uso sem Python instalado:

	Auto Close Tag – Automaticamente adiciona o fecho de uma tag HTML/XML.
	Auto Rename Tag – Emparelha os nomes das tags HTML/XML, na sua abertura ou fecho.
	Better Comments – Ajuda a melhorar os comentários, criando anotações com alertas, questões, etc.
	Code Runner – Corre instruções ou ficheiros de código para múltiplas linguagens.
	Colorize – Extensão vscode para ajudar na visualização das cores em css.
	CSS Peek – Permite ir buscar ID e classes do css nos ficheiros html onde são chamados.
	Icons Fonts - Snippets (são blocos de códigos utilizados nos programas para agilizar o desenvolvimento de código) para fontes de ícones populares, como Font Awesome, Ionicons, Glyphicons, Octicons,etc.
	indent-rainbow – Torna a indentação mais fácil de ler, através de cores
	IntelliSense for CSS class names in HTML – Conclusão do nome da classe CSS para o atributo de classe HTML com base nas definições encontradas em seu espaço de trabalho
	O JavaScript (ES6) code snippets – Snippets para javascript e typescript
	Live Server – Inicia um servidor local de desenvolvimento com recurso de recarga ao vivo para páginas estáticas e dinâmicas.

	npm Intellisense – Visual Studio Code plugin que autocompleta módulos npm através da importação de instruções
	Path Intellisense – Visual Studio Code plugin que autocompleta caminhos e nomes de ficheiros
	Prettier - Code formatter – Formatador de código
	Tabnine AI Autocomplete – JavaScript, Python, Java, Typescript e outras linguagens – Codifique mais rápido com conclusões de código IA (Inteligência Artificial)
	vscode-icons – Icons para o Visual Studio Code

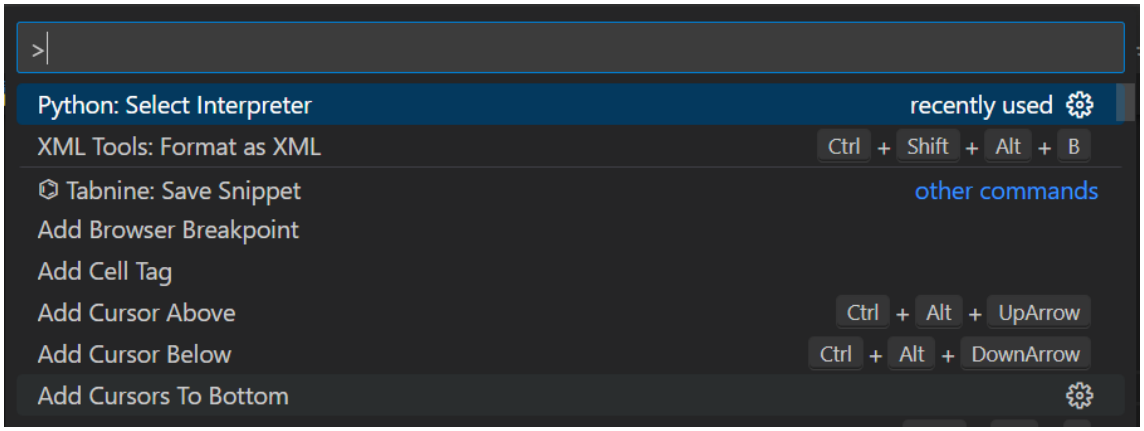
- Uso com Python instalado:
  - Python - Uma extensão do Visual Studio Code com suporte avançado para a linguagem Python (para todas as versões ativamente suportadas da linguagem: >=3.7), incluindo recursos como IntelliSense (Pylance), linting (processo que analisa o código fonte, permitindo identificar potenciais problemas/erros no código), depuração, navegação de código, formatação de código, refatoração, explorador de variáveis, explorador de testes e muito mais! Com a instalação desta extensão, vão ser instaladas outras automaticamente.

    
  - Jupyter - Suporte de blocos Jupyter, para programação e computação interativa com Intellisense, depuração e muito mais. Com a instalação desta extensão, vão ser instaladas outras automaticamente.

Como pode haver extensões com nomes iguais ou similares, portanto, é muito importante que se observe o símbolo da extensão.

Configurações

O Visual Studio Code permite muitas configurações, e você está livre para brincar aqui, porém vamos deixar as configurações que utilizamos nos cursos listada abaixo.
Para aceder às configurações, basta pressionar CTRL(CMD)+VÍRGULA, e será apresentado a uma interface onde poderá modificar ou pesquisar uma configuração.
No canto superior direito desta janela, haverá um botão chamado Open Settings (JSON) que irá exibir o JSON das configurações atuais.
Caso queira manter todas as configurações, não faça alterações e feche a janela.
Quando usar o Python pela primeira vez, tem de atualizar o interpretador para o Python. Para isso, use a combinação das teclas CTRL+SHIFT+P e selecione o interpretador mostrado na imagem.



Instalação do Flask

Pode consultar tudo no site https://flask.palletsprojects.com/en/2.2.x/installation/
Comando para a instalação do Flask:

```
pip install flask
```

Caso ocorra um erro de execução em que o cominho para o ficheiro python ou pip dê erro ou esteja mal direcionado, entre no ambiente do seu projeto (venv), ative-o e corra o seguinte comando:

```
python -m pip install --upgrade --force-reinstall pip
```

API's que podem interessar:

- Trabalhar com ficheiros do Microsoft Excel
  `pip install openpyxl==2.4.8`
  `from openpyxl import load_workbook, Workbook`
- Trabalhar com numeração e arredondamentos

```
pip install numpy
import numpy as np
```

- Trabalhar com pastas e ficheiros (biblioteca incluída no Flask)

`import os` ou `from pathlib import Path`

- Trabalhar com datas e horas (biblioteca incluída no Flask)

```
import datetime
```

- Encriptação de dados (biblioteca incluída no Flask)

```
from werkzeug.security import generate_password_hash
```

- Gerar ficheiros PDF

```
pip install fpdf2
import fpdf2 as p
```

Atualizar Flask

Comando para a atualização do Flask:
`pip install --upgrade Flask`

Atualizar PIP

Comando para a atualização do PIP:
`python -m pip install --upgrade pip`

Saber qual a versão Flask e seus componentes

Saber a versão do flask instalada: `flask –version`
Python 3.11.1
Flask 2.2.2
Werkzeug 2.2.2

Ficheiro requeriments.txt

Na janela do terminal, correr o comando **pip freeze > requirements.txt** para criar o ficheiro requirements.txt na pasta do projeto.
Qualquer um (ou qualquer servidor) que receba a cópia do projeto necessita apenas que corra o comando **pip install -r requirements.txt** para reinstalar os pacotes contidos no ambiente original. (O recetor continua a necessitar de criar, contudo, o seu ambiente virtual).

Trabalhar com o SQLite

O SQLite é uma biblioteca de linguagem C que implementa um mecanismo de base de dados SQL pequeno, rápido, independente, de alta confiabilidade e completo. O SQLite é o mecanismo de base de dados mais usado no mundo e está embutido em todos os smartphones e na maioria dos computadores, vindo embutido em inúmeros outros programas que as pessoas usam todos os dias. Pode consultar tudo no site https://www.sqlite.org/index.html
Para podermos criar e manipular bases de dados SQLite, usaremos o programa "SQLiteStudio", que pode fazer o seu descarregamento gratuito no site https://sqlitestudio.pl/
A incorporação de bases de dados no pyhton, basta usar a seguinte linha de código:

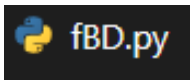
```
import sqlite3
```

Todos os comandos SQLite podem ser consultados em (entre outros):

- https://www.sqlitetutorial.net/
- https://www.tutorialspoint.com/sqlite/index.htm

Criação de um ficheiro python com comandos pré-definidos

O melhor conselho que é dado, para que um ficheiro não seja uma amalgama de código que ninguém percebe, deve dividi-lo em mais ficheiros, cada um com a sua funcionalidade. Podemos então criar um ficheiro unicamente para colocar as funções necessárias para lidarmos com a base de dados, como: abrir e fechar a base, para inserir ou atualizar ou apagar registos de uma tabela, entre outros que acharmos necessário.
Em primeiro lugar, criamos o ficheiro colocar tudo o que tenha a ver com as bases de dados:



Segundo, importamos todas as bibliotecas necessárias:

```
from tkinter import messagebox
from flask import g, jsonify, json
import sqlite3 as s
import os, fgerais, datetime
# from sqlalchemy import null
from werkzeug.security import generate_password_hash, check_password_hash
```

Terceiro, definimos um nome para a base de dados:

```
# Nome da base de dados
nome_bd_gmp_web = "gmp_web_v1.db"
```

Agora incorporamos todas as funções necessárias para trabalhar com uma base de dados SQLite, mas não é necessário seguirem à risca todas as funções aqui apresentadas. Pode acrescentar ou eliminar funções.

Função que devolve a conexão à base de dados e caso ela não exista cria-a. Permite, também, que possamos definir uma função para criar as tabelas para a base de dados e os registos a inserir por defeito.

```
###############################################################################
# Função que faz e devolve a conexão à base de dados
#_____________________________________________________________________________#
# tabelas_criar - criar as tabelas na base de dados
# inserir_registos - inserção dos registos por defeito
###############################################################################
########################

def get_db (tabelas_crear=False, inserir_registos=False):
    db = getattr(g, 'database', None)
    if db is None:
        db = g._database = s.connect(nome_bd_gmp_web)
        db.isolation_level = None

    cur = db.cursor()
    if tabelas_crear:
        crear_tabelas(cur)
        inserir_registos=True

    if inserir_registos:
        registos_inserir(cur)

    db.commit()

    return db
```

Função que permite fechar a conexão à base de dados, caso esteja aberta.

```
###############################################################################
# Função que fecha a conexão à base de dados
###############################################################################

def close_connection():
    db = getattr(g, 'database', None)
    if db is not None:
        db.close()
```

Função que devolve uma lista com os campos de uma tabela.

```
###############################################################################
# Função que devolve os campos de uma tabela
#_____________________________________________________________________________#
# tabela - tabela a recolher os campos
```

```
# sem_incremento - sem incrementos
# devolver_como_vetor - False (por defeito) como string e True como vetor
# mostrar_mensagem - mostrar a mensagem com os dados a devolver
###############################################################################

def devolve_campos_tabela(tabela, elimina_campos = [], sem_incremento =
False, devolver_como_vetor = False, mostra_mensagem=False):
    db = get_db()
    c = db.cursor()

    try:
        # lista_campos=[]
        comando_create = c.execute(f'''SELECT sql FROM sqlite_master
WHERE lower(tbl_name) = lower('{tabela}')''').fetchone()
        # messagebox.showinfo('INFO',comando_create)

        if not comando_create:
            messagebox.showinfo('ERROR', f'A tabela {tabela} não
existe.')
        else:
            # Limpar o comando CREATE TABLE ( ... )
            pos_i=comando_create[0].find("(")+1
            comando_create = comando_create[0][pos_i:-1]

            lista_campos=[x.strip() for x in
comando_create.strip('[]').split(',')]

            campos=[x.split(' ') for x in lista_campos]

            if not devolver_como_vetor:
                str_campos=""
            else:
                str_campos=[]

            for item in campos:
                isAI = False
                if sem_incremento:
                    for cell in item:
```

```
                        if cell.lower() == 'autoincrement':
                            isAI=True
                if not isAI:
                    if not devolver_como_vetor:
                        if elimina_campos==[]:
                            str_campos+=item[0] + ", "
                        else:
                            str_campos+= '' if item[0].lower() in
elimina_campos else item[0] + ", "
                    else:
                        if elimina_campos==[]:
                            str_campos.append([item[0], item[1]])
                        elif not (item[0].lower() in elimina_campos):
                            str_campos.append([item[0], item[1]])

            if not devolver_como_vetor:
                str_campos=str_campos[:-2]

            if mostra_mensagem:
                if devolver_como_vetor:
                    messagebox.showinfo("INFO", "Campos vetor: " +
str(str_campos))
                else:
                    messagebox.showinfo("INFO", "Campos string: " +
str_campos)

        #comando_create = c.execute(f'''SELECT str FROM sqlite_master
WHERE lower(tbl_name) = lower('{tabela}')''').fetchone()

        if comando_create:
            if not devolver_como_vetor:
                return str_campos
            else:
                return campos
        else:
            return None
    # em caso de erro no comando sqlite
    except s.Error as err:
```

```
        messagebox.showerror("ERRO - devolve_campos_tabela SQL", err)
    except Exception as e:
        messagebox.showerror("ERRO - devolve_campos_tabela Código", err)
    finally:
        close_connection()
```

Validação de utilizadores, após verificação se estes estão não autorizados, inativos, tem obrigação de alterar a palavra-passe no login, utilizador não existente e palavra-passe errada.

```
##############################################################################
# Função que valida o utilizador e devolve informações
#_____________________________________________________________________________#
# utilizador - nº de processo do utente
# palavra_passe - palavra-passe do utente
##############################################################################

def db_procura_utilizador(utilizador, palavra_passe):
    db = get_db(True)
    c = db.cursor()

    try:
        #                           0        1           2           3
     4          5                6
        data = c.execute("SELECT codigo, nProcesso, palavraPasse,
nomeCurto, ativo, autorizadoLogin, trocarPasse FROM temp_Utentes WHERE
nprocesso=?",(utilizador,)).fetchone()

        if data:
            if (check_password_hash(data[2],palavra_passe)):
                if data[4]==0:
                    return -3 # Utilizador inativo
                if data[5]==0:
                    return -4 # Utilizador não autorizado
                if data[6]==1:
                    return -5 # Utilizador tem de trocar a passe

                tipoUtil = db_lista_tabela("temp_Tipos_Utentes A,
temp_Utentes_Tipos B","max(nivel), sigla, tipo", \
                    f"A.codigo = B.codTtipo and B.codUtente = {data[0]}")

                if not tipoUtil:
                    return -6 # Utilizador não tem associado nenhum tipo
de utilizador

                #           0       1       2       3
                #         codigo  nProc    NomeC  Tipos de Utilizador
```

```
                devolver=[data[0],data[1],data[3]] + tipoUtil
                #messagebox.showinfo("INFO",str(devolver))

                return devolver
            else:
                return -2 # Palavra-Passe errada
        else:
            return -1 # Utilizador não existe
    finally:
        close_connection()
```

Função que devolve o valor de um campo específico da tabela.

```
##########################################################################
# Função que devolve o valor de um campo específico na tabela
#_______________________________________________________________________#
# tabela - tabela a recolher os campos
# campo_a_mostrar - nome do campo da tabela
# query_where - condições de seleção do registo
# mostrar_mensagem - mostrar a mensagem com os dados a devolver
##########################################################################

def db_valor_campo_tabela(tabela, campo_a_mostrar, query_where='1=1',
mostrar_mensagem=False):
    db = get_db()
    c = db.cursor()

    try:
        str_sql = f'SELECT {campo_a_mostrar} FROM {tabela} WHERE
{query_where}'

        if mostrar_mensagem: messagebox.showinfo("INFO",str_sql)

        data = c.execute(str_sql).fetchone()
        close_connection()

        if data:
            return data[0]
        else:
            return None
    finally:
        close_connection()
```

Função que devolve um vetor (array) com a lista de valores encontrados para uma ou mais tabelas.

```
##########################################################################
# Função que devolve uma lista de valores de uma ou mais tabelas
#_______________________________________________________________________#
```

```
# tabela - tabela a recolher os campos
# campo_a_mostrar - nome do campo da tabela
# query_where - condições de seleção do registo
# mostrar_mensagem - mostrar a mensagem com os dados a devolver
###############################################################################

def db_lista_tabela(tabelas, campos_a_mostrar, query_where='1=1',
mostrar_mensagem=False):
    db = get_db()
    c = db.cursor()

    try:
        sql = 'SELECT ' + campos_a_mostrar + ' FROM ' + tabelas + ' WHERE
' + query_where
        if mostrar_mensagem: messagebox.showinfo("INFO",sql)
        data = c.execute(sql).fetchall()
        close_connection()

        if data:
            return data
        else:
            return None
    except s.Error as err:
        messagebox.showerror("ERRO", f"Erro: {err}\nSQL: {sql}")
        return None
    finally:
        close_connection()
```

Função que devolve o número de registos encontrados para uma tabela ou mais tabelas.

```
###############################################################################
# Função que devolve o número de registos
#_____________________________________________________________________________#
# tabelas - tabela a recolher os campos
# query_where - condições de seleção do registo
# mostrar_sql - mostrar uma mensagem com o comando sql
###############################################################################

def db_contar_registos_tabela(tabelas, query_where='1=1', mostrar_sql =
False):
    db = get_db()
    c = db.cursor()

    try:
        str_sql = f"SELECT count(*) as contador FROM {tabelas} WHERE
{query_where}"

        if mostrar_sql:
            messagebox.showinfo("INFO",str_sql)
```

```
        data = c.execute(str_sql).fetchone()
        close_connection()

        if data:
            return data[0]
        else:
            return None
    finally:
        close_connection()
```

*Função que executa o comando **insert into** e devolve o valor **False** se o comando não foi executado corretamente, caso contrario devolve o valor **True**.*

```
##########################################################################
# Função que devolve verdadeiro/false caso o comando de insert into seja
executado com sucesso
#_____________________________________________________________________#
# tabela - tabela a fazer o insert
# campos - campos da tabela separados por , - pode utilizar a função
devolve_campos_tabela
# valores - valores para os campos separados por , - não deve colocar
plicas nos valores
# divisor_campos - carater ou carateres utilizados para a divisão dos
campos
# divisor_valores - carater ou carateres utilizados para a divisão dos
valores
# mostrar_mensagem - mostrar a mensagem com os dados a devolver
##########################################################################

def db_INSERT_tabela(tabela, campos, valores, divisor_campos=",",
divisor_valores=",", mostra_mensagem = False):
    # Teste se o número de campos coinside com o número de valores
    campos1=campos.split(divisor_campos)
    valores1=valores.split(divisor_valores)

    n_campos=len(campos1)
    n_valores=len(valores1)

    if (n_campos != n_valores):
        messagebox.showerror(f"ERRO - db_INSERT_tabela - {tabela}", f"O
nº de campos {n_campos} deve ser igual ao nº de valores {n_valores}" + \
            f"Campos = {campos1}\nValores={valores1}")
        return None

    # abertura da base de dados e do cursor
    db = get_db()
    c = db.cursor()

    try:
```

```
        # Passagem dos valores None para null
        valores=valores.replace("None","null")
        if divisor_valores!=",":
valores=valores.replace(divisor_valores,",")
        if divisor_campos!=",": campos=campos.replace(divisor_campos,",")

        # Comando sqlite para efetuar o insert into
        sql=f''' INSERT INTO {tabela} ({campos}) VALUES ({valores}) '''

        campos1.insert(0, tabela)

        dados = c.execute(sql)

        db.commit()
        c.close()

        # Devolve True
        return [True, sql]

    # Caso de erro do comando sqlite
    except s.Error as err:
        messagebox.showerror("ERRO", err.message)
        # Devolve False
        return False
    except Exception as e:
        messagebox.showerror("ERRO", e.message)
        # Devolve False
        return False
      finally:
        close_connection()
```

*Função que devolve o valor **True/False**, caso o comando de **update** seja executado com sucesso.*

```
##############################################################################
# Função que devolve verdadeiro/false caso o comando de update seja
executado com sucesso
#_____________________________________________________________________________#
# tabela - tabela a fazer o update
# campos - campos da tabela separados por , - pode utilizar a função
devolve_campos_tabela
# valores - valores para os campos separados por , - não deve colocar
plicas nos valores
# query_where - condições de seleção do registo
# divisor_campos - carater ou carateres utilizados para a divisão dos
campos
# divisor_valores - carater ou carateres utilizados para a divisão dos
valores
# mostrar_mensagem - mostrar a mensagem com os dados a devolver
```

```
#############################################################################

def db_UPDATE_tabela(tabela, campos, valores, query_where,
divisor_campos=",", divisor_valores=",", mostrar_mensagem = False):
    valores=valores.replace("None","null")

    if divisor_valores!=",": valores=valores.replace(divisor_valores,",")
    if divisor_campos!=",": campos=campos.replace(divisor_campos,",")

    campos=campos.split(",")
    valores=valores.split(",")

    n_campos=len(campos)
    n_valores=len(valores)

    if (query_where==""):
        messagebox.showerror(f"ERRO - db_UPDATE_tabela", f"Não foi
definida a query de procura")
        return False

    if (n_campos != n_valores):
        messagebox.showerror(f"ERRO - db_UPDATE_tabela - {tabela}", f"O
nº de campos {n_campos} deve ser igual ao nº de valores {n_valores}")
        return False

    str_valores=""
    for i in range(len(campos)):
        str_valores += (campos[i] + "=" + valores[i]) + " ,"
        if (i==len(campos)-1):
            str_valores=str_valores[:-2]

    db = get_db()
    c = db.cursor()

    try:
        query_where = "WHERE " + query_where
        sql=f''' UPDATE {tabela} SET {str_valores} {query_where} '''
        if mostrar_mensagem: messagebox.showinfo("INFO",sql)
        data = c.execute(sql)
        return True
    except s.Error as err:
        messagebox.showerror("ERRO", err.message)
        # Devolve False
        return False
    except Exception as e:
        messagebox.showerror("ERRO", e.message)
        # Devolve False
        return False
    finally:
```

```
        close_connection()
```

*Função que devolve o valor **True/False**, caso o comando de **delete** seja executado com sucesso.*

```
##############################################################################
# Função que devolve verdadeiro/false caso o comando de delete seja
executado com sucesso
#_____________________________________________________________________________#
# tabela - tabela a fazer o
update
# query_where - condições de seleção do registo
##############################################################################

def db_DELETE_tabela(tabela, query_where):
    if (query_where==""):
        messagebox.showerror(f"ERRO - db_UPDATE_tabela", f"Não foi
definida a query de procura")
        return False

    db = get_db()
    c = db.cursor()

    try:
        query_where = "WHERE " + query_where
        sql=f''' DELETE FROM {tabela} {query_where} '''
        # messagebox.showinfo("INFO",sql)
        data = c.execute(sql)
    except s.Error as err:
        messagebox.showerror("ERRO", err)
        # Devolve False
        return False
    finally:
        close_connection()
        # Devolve True
        return True
```

Execução de bibliotecas próprias

Para conseguir executar qualquer uma das funções presente no ficheiro fBD.py, é necessário efetuar a importação deste ficheiro numa outra página.

```
import fBD
```

Para executarem uma das funções do ficheiro fBD.py (<nome do ficheiro>.<nome da função()>):

```
fBD.close_connection()
```

Ou caso queiram, podem também especificar as funções a importar.

```
from fBD imports close_connection()
```

Nesta caso, basta chamarem no nome da função (<>):

```
close_connection()
```

Não é necessário especificar o nome do ficheiro, pois fica implícito na importação da função.