import os

import sys

import tkinter as tk

from tkinter import filedialog, messagebox

from openpyxl import load\_workbook

from openpyxl\_image\_loader import SheetImageLoader

def extract\_images\_from\_excel(excel\_path):

    try:

        wb = load\_workbook(excel\_path)

        ws = wb.active

        image\_loader = SheetImageLoader(ws)

        # 获取Excel所在文件夹路径

        output\_dir = os.path.dirname(excel\_path)

        row = 4

        extracted = 0

        while True:

            name\_cell = f"C{row}"

            photo\_cell = f"D{row}"

            name = ws[name\_cell].value

            if not name:

                break  # 数据读取结束

            if image\_loader.image\_in(photo\_cell):

                image = image\_loader.get(photo\_cell)

                filename = f"{name}\_大头照.jpg"

                save\_path = os.path.join(output\_dir, filename)

                image.save(save\_path)

                print(f"已保存: {save\_path}")

                extracted += 1

            else:

                print(f"未找到{name}的照片")

            row += 1

        messagebox.showinfo("完成", f"已提取 {extracted} 张照片，保存在\n{output\_dir}")

    except Exception as e:

        messagebox.showerror("出错", f"提取失败：{str(e)}")

def main():

    root = tk.Tk()

    root.withdraw()

    file\_path = filedialog.askopenfilename(

        title="请选择包含照片的Excel文件",

        filetypes=[("Excel 文件", "\*.xlsx \*.xlsm")]

    )

    if not file\_path:

        sys.exit()  # 用户取消

    extract\_images\_from\_excel(file\_path)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

####简介：是一个 图形化操作的“Excel照片批量提取器”，

####帮助用户从 Excel 表格中自动提取嵌入的照片，并按姓名保存成 JPG 文件。

####具体流程：1、启动后弹出文件选择对话框 → 选择 Excel 文件。

####具体流程：2、Excel 中从第4行开始读取：C列读取姓名、D列检查是否嵌入图片

####具体流程：3、如果有嵌入图片：提取出来，以“姓名\_大头照.jpg”格式保存到 Excel 所在文件夹

####具体流程：4、最后弹出提示：共提取多少张，保存在哪里。

####使用环境：在Windows下运行，使用.xlsx格式。excel格式要求：第4行起，每一行代表一个人；C列是姓名，D列每格插入图片，图片嵌入单元格中。

####输出说明：照片保存在与 Excel 相同目录下，命名格式为：“姓名\_大头照.jpg”。

###这个程序从头到尾可以分为4大主要模块

##一、模块导入区（准备工具）

##作用：导入功能库，比如文件路径处理（os）、图形界面（tkinter）、读取Excel（openpyxl）、

##提取图片（SheetImageLoader）等。就像你准备好了工具箱，等待后续使用。

import os

# import 是“导入”的意思；

# os ：operating system（操作系统），os 是一个“操作系统”模块，管文件路径、文件名等；

# 【功能】os 提供操作系统相关功能，如获取文件路径、拼接文件路径、获取文件夹名等。

# os 你想知道 Excel 文件在哪个文件夹里，就可以用 os 模块的功能。

# import os【意思】导入 os 模块

# 它的作用：和操作系统打交道，总之，只要是和“文件和路径”打交道，就离不开它。

# 比如：获取文件/文件夹的路径、拼接路径、列出文件夹里的文件、

# 删除文件判断文件或文件夹是否存在、创建/删除文件夹。

#【什么时候】涉及文件路径操作/操作文件夹（比如批量处理文件、自动化保存文件、遍历文件夹等）时，

# 当需要读取或拼接文件路径（把一个文件夹路径和一个文件名合并成一个完整的路径，直接用来读写文件）时，

# 当需要跨平台（Windows、Mac、Linux）处理文件路径时。

# 怎么一眼知道要不要用os？

# 一旦提到文件夹或路径大概率：import os。

# 关键词：文件夹（比如：读取文件夹、遍历文件夹）、路径（比如：拼接路径、获取文件夹）、

# 批量处理文件（比如：保存、移动、删除文件）、跨平台（比如：Windows、Mac、Linux）。

# 例子 1：“把 Excel 图片保存到同一个文件夹” ➔ 要知道文件夹路径 ➔ 用 os.path.dirname() ➔ 要导入 os

# 例子 2：“列出文件夹里的所有文件” ➔ 需要列出 ➔ 用 os.listdir() ➔ 要导入 os

# 例子 3：“拼接路径” ➔ 把文件夹+文件名 ➔ 用 os.path.join() ➔ 要导入 os

# 例子 4：“批量删除某个文件夹里的所有图片” ➔ 用 os.listdir()+os.remove() ➔ 要导入 os

import sys

# syssys 是 Python 的一个内置模块（标准库），全称是 "system"。

# 它提供了与 Python 解释器和系统相关的功能。就像一个工具箱，里面装着各种与系统交互的工具。

# 为什么要导入 sys？在这里有什么作用？

# 在我的代码中，sys 只用了一次：

# if not file\_path:

#    sys.exit()  # 用户取消选择文件时，退出程序

# 作用：

# sys.exit() 用来立即退出程序

# 当用户在文件选择对话框中点击"取消"时，file\_path 就是空的

# 这时程序就调用 sys.exit() 直接结束，不再继续执行后面的代码

# sys是 Python 的“系统模块”（system），可以和解释器交互，比如：

# 退出程序（sys.exit()）、读取命令行参数（sys.argv）、获取当前解释器的路径（sys.path）/获取系统消息（# 就像查看手机的"关于本机"信息）

# sys.argv 的英文全称：argv = argument vector "系统参数列表"，其中argument = 参数、论据、vector = 向量、数组、列表

# argument（参数）：就是你传给程序的额外信息；vector（向量）指 "有序的列表"，类似数组。

# \*\*命令行参数就是：在运行程序时，额外告诉程序的信息。

# \*\*命令行参数，比喻：我要点餐 川菜= 参数1（菜系） 麻婆豆腐= 参数2（菜名） 中辣= 参数3（辣度）

# \*\*命令行参数，@小明 今天吃什么？@小明 = 参数1（告诉微信发给谁） 今天吃什么？ = 参数2（消息内容）

# \*\*命令行参数，python 我的程序.py 牛肉汉堡 大份 不要洋葱

# \*\*命令行参数，这里：python = 告诉电脑用Python运行 我的程序.py = 程序名

# \*\*命令行参数，这里：牛肉汉堡 大份 不要洋葱 = 参数（传给程序的信息）

# 假设在命令行输入：python hello.py 张三 25 北京。这时候 sys.argv 就变成了：sys.argv = ['hello.py', '张三', '25', '北京']

# sys.argv[0] = 'hello.py'    # 第0个盒子：程序名

# sys.argv[1] = '张三'         # 第1个盒子：第一个参数

# sys.argv[2] = '25'           # 第2个盒子：第二个参数

# sys.argv[3] = '北京'        # 第3个盒子：第三个参数

# sys 模块可以控制程序运行，比如“退出程序”。

# 【功能】sys 提供与 Python 解释器交互的功能，比如 sys.exit() 可以用来退出程序。

# sys 用户没选文件，程序就应该退出，不继续运行或者说：

# sys 用户没选 Excel 文件，程序就可以用 sys.exit() 来退出，不再继续运行。

# 【为什么】当用户没有选择 Excel 文件时，程序就需要退出，防止继续运行时出现错误。

# 【什么时候】需要控制程序流程（比如异常退出、检查依赖项等）时，都可以用到 sys。

# 怎么第一眼知道需要导入 sys？

# 方法1：看到特定的功能需求

# 需要退出程序 → 想到 sys.exit()

# 需要命令行参数 → 想到 sys.argv

# 需要系统信息 → 想到 sys.platform、sys.version

# 需要访问Pyhon解释器的额特性

# 方法2：看错误提示

# 如果你写了 exit() 但没导入 sys，Python 会提示：

# NameError: name 'sys' is not defined

# 方法3：参考经验 退出程序 → import sys

import tkinter as tk

# tkinter：Tk interface（Tk 是 GUI 系统），举例：弹窗、按钮等图形界面。

# tkinter 是 Python 自带的图形界面工具，可以做窗口、按钮、弹窗等；

#【功能】tkinter 是 Python 自带的 GUI 模块，用于创建窗口、按钮、弹窗等界面。

# as ：作为

# as tk 的意思是：我们给 tkinter 起个短名字叫 tk，后面用起来更方便。

# 【意思】导入 tkinter 模块，并起别名 tk

# 【为什么】需要图形界面（比如文件选择对话框和提示框），并且 tk 别名简洁易用。

# 【什么时候】需要用户界面交互（文件选择、确认对话框、用户输入）时可以用到。

# tkinter = 灵活性 ：用户可以选择任意文件 固定路径 = 局限性：只能处理特定位置的文件

# 使用tkinter相比于固定路径的优点：用户可以选择任意位置的文件；每次运行都可以选择不同的文件；

# 适合多人使用，每个人的文件路径都不同；用户友好，点击选择比输入路径简单；容错性好，用户选错了可以重新选择。

# 比如：一个py或者封装后的exe，如果使用固定路径，换一台电脑就无法使用，路径改变就很麻烦。

# 最适合：单机桌面工具，特别是数据处理和文件操作类应用。

# 做的实例中，第一个例子：提取Excel中的图片：tkinter 具体做了什么？

# import tkinter as tk

# from tkinter import filedialog, messagebox

# (1) 创建了一个主窗口（root = tk.Tk()），然后使用 root.withdraw() 把主窗口隐藏了。

# 这是一个小技巧：我们只想用 tkinter 弹出文件选择对话框和信息提示框，而不想显示完整的窗口。

# (2) 使用了 filedialog.askopenfilename() 弹出文件选择对话框，让用户选择包含照片的 Excel 文件。

# 这解决了“文件路径固定”带来的问题，让用户可以灵活选择文件。

# (3) 使用了 messagebox.showinfo() 和 messagebox.showerror() 来弹窗显示程序执行结果。

# 如果成功提取了图片，就显示“已提取 x 张照片”；如果出错，就用错误提示。

# tkinter 在这里的角色：负责文件选择（弹窗对话框）、信息提示（完成或错误的弹窗）、让程序更友好、更直观（不需要命令行输入文件路径）。

# 做的实例中，第二个例子：处理Excel花名册

# from tkinter import filedialog, Tk

# Tk().withdraw()

# source\_file = filedialog.askopenfilename(title="请选择原始花名册", filetypes=[("Excel files", "\*.xlsx")])

#  tkinter 具体做了什么？

#  (1) 创建了一个 Tk() 主窗口（并隐藏它）。

# 这里用 .withdraw() 也是为了只用对话框，不显示完整窗口。

# (2) 使用了 filedialog.askopenfilename() 弹出文件选择对话框。

# 让用户选择要处理的 Excel 文件。

# (3) 如果用户没有选择文件，就打印“❌ 未选择文件，程序已退出。”然后退出程序。

#  这个逻辑很实用：通过弹窗选择文件，避免了写死的路径问题。

#  所以 tkinter 在这里的角色：负责文件选择（弹窗对话框）；让用户自主选择文件（而不是写死路径）；提升了程序的可移植性（换台电脑也能用）。

# 这两个例子（提取 Excel 中的图片、处理 Excel 花名册）用到的 tkinter 主要是文件选择对话框（filedialog.askopenfilename()）

# 和提示框（messagebox.showinfo() / showerror()），这是 tkinter 最常用的轻量化 GUI 功能。

from tkinter import filedialog, messagebox

# 这是从 tkinter 里“挑选”出我们要用的两个功能：

# filedialog 是“文件对话框”，让用户选择 Excel 文件；

# messagebox 是“弹窗消息框”，可以弹出“成功”或“出错”的对话框。

# tkinter.filedialog：弹出“文件选择”窗口；为什么需要：否则用户怎么选 Excel？

# tkinter.messagebox：弹出“成功/失败”提示框；为什么需要：用来提示用户处理结果。

# filedialog：文件对话框；messagebox：消息框；

# 【意思】从 tkinter 模块中只导入 filedialog 和 messagebox

# 【功能】filedialog：弹出文件选择对话框；messagebox：弹出提示对话框（完成/出错）。

# 【为什么】程序只用到这两个功能，直接导入减少冗余。

# 【什么时候】需要文件选择或者消息提示（如上传文件、保存文件、错误提示）时都可以用。

from openpyxl import load\_workbook

# openpyxl：一个第三方模块，用于处理 Excel 文件。

# load\_workbook：openpyxl 里的一个函数，用于加载 Excel 文件。

# 功能：加载 Excel 文件到 Python，后面可以读取或写入数据。

# 为什么要用：必须先把 Excel 文件加载到内存里才能进行数据操作。

# 什么时候用到：处理 Excel 文件时（如批量导出数据、自动化生成表格）。

from openpyxl\_image\_loader import SheetImageLoader

# openpyxl\_image\_loader：第三方模块，用于提取 Excel 中的图片。

# image 图像 loader：加载器 image\_loader：图像加载器

# SheetImageLoader：模块里的一个工具类，用于提取 Excel 中的图片。

# SheetImageLoader：表格图像加载器

# 功能：提取 Excel 表格中的图片。

# 为什么要用：openpyxl 本身不能直接提取图片，必须借助这个模块。

# SheetImageLoader：是 openpyxl\_image\_loader 模块中的一个类，用于加载并提取 Excel 中的图片。

# 什么时候用到：需要从 Excel 表格中提取图片时（如身份证照、员工照片）。

# openpyxl 本身是最常用的 Python Excel 读写库，但它不直接支持图片的提取（只能读取单元格、格式等数据）。

# openpyxl\_image\_loader 是 openpyxl 的一个“插件”，

# 专门用来把 Excel 文件中的嵌入图片提取出来，比如工单照片、身份证照片等。

# 也就是说，如果你用 openpyxl 想提取图片，最简单的方法就是用 SheetImageLoader。

# 什么时候用 SheetImageLoader？

# 如果你用 openpyxl 处理 Excel，并且需要提取里面的图片（特别是插入单元格的嵌入式图片），

# 建议使用 SheetImageLoader，因为它集成度高、使用简单、跨平台支持好。

##二、功能函数区（提取图片的核心功能）

##作用：定义照片提取的完整流程：打开Excel → 遍历数据 → 提取嵌入图片 → 命名并保存。

##如果失败，会弹出错误提示。就像你写好了一套“提取照片的机器人操作流程”。

##Python 程序执行顺序：

#Python 程序执行时，不是按写的顺序从上往下“全部执行”，

#而是先 读入函数定义（但不运行），再从 main() 或入口处开始“真正执行”。

#Python 会先读“定义”，再按调用顺序执行。

#写在前面的函数，只是定义好了位置，不会自动跑起来，只有被“叫到名字”时才执行。

#当 Python 运行这段程序时，它会先做一件事：

#把 def extract\_images\_from\_excel(...) 和 def main(): 都读进来，先记住定义了哪些函数。

#但此时函数内部的代码都没有被执行，只是“准备好”，等以后需要再调用它们。

#到了最后这句：if \_\_name\_\_ == "\_\_main\_\_":  main() ，从 main() 函数里的第一行开始运行：def main()

#再def extract\_images\_from\_excel(...) 直到messagebox.showinfo等。

#def extract\_images\_from\_excel(excel\_path)这一整段函数都是：定义函数（只是准备好，但不执行）

def extract\_images\_from\_excel(excel\_path):

# def：define，定义一个函数。当你有一段功能（比如提取图片），想在多处调用它时。

# extract\_images\_from\_excel 从excel提取图片，函数的名字（自定义的）

# (excel\_path) 圆括号：表示函数的参数列表。

# excel\_path：是函数的参数（输入），指的是 Excel 文件的完整路径。

# path：文件路径，指 Excel 文件在计算机上的存储位置。

# 功能：告诉函数你要处理哪个 Excel 文件。

# 为什么：不同用户选的文件路径可能不同，必须通过参数传入。

# 什么时候可以用到：当你需要对不同的 Excel 文件进行操作时。

# 定义了一个名为 extract\_images\_from\_excel 的函数，它接收一个参数 excel\_path，

# 表示要提取的 Excel 文件路径。函数的功能是：从指定的 Excel 文件中提取图片。

# 把整行换成“白话”说法就是：“我要写一个功能，名字叫 extract\_images\_from\_excel，

# 它会从用户传进来的 Excel 文件（路径）里提取出图片，然后保存到指定位置。”

# 当某段功能需要多次使用、逻辑清晰、便于维护和扩展时，就可以把它写成一个函数（def 函数名(参数):）。

# 什么时候用到？

# 整行：当你需要封装一段逻辑（比如从 Excel 提取图片），并且希望这段逻辑可以多次调用、方便修改和调试时。

# 分解：

# def：当需要写函数时。

# extract\_images\_from\_excel：当需要给函数起名字时。

# (excel\_path)：当需要给函数传入外部数据时（比如文件路径、配置等）。

# 函数的基本结构：

# def 函数名(参数1, 参数2, ...):

#       """（可选）函数的文档字符串，用于说明这个函数是干什么的"""

#       函数体

#       return 返回值（可选）

# def：英文 define，表示定义函数的意思。

# 函数名：自定义的名字，用来描述函数的功能（见名知意）。

# ()：括号里放的是参数，表示外部要传进来的数据。

# :：冒号，函数定义时必须写冒号，表示函数体开始，下面的缩进部分是函数的主体代码。

# return（可选）：表示函数的输出结果（如果需要）

# 练习 2：写一个函数，输入文件夹路径，打印该文件夹里所有文件的名字。

# import os

#

# def list\_files(folder\_path):

#     """打印文件夹中的所有文件"""

#     files = os.listdir(folder\_path)

#     for file in files:

#         print(file)

# 练习 3：写一个函数，输入 Excel 文件路径，读取第一行第一列的值并返回。

# from openpyxl import load\_workbook

# def get\_first\_cell(excel\_path):

#     """读取 Excel 第一行第一列的值"""

#     wb = load\_workbook(excel\_path)

#     ws = wb.active

#    value = ws['A1'].value

#    return value

#  问题1、练习 2：写一个函数，输入文件夹路径，打印该文件夹里所有文件的名字。

#  和练习 3：写一个函数，输入 Excel 文件路径，读取第一行第一列的值并返回。与我的例子有什么相同之处？

#  它们都用到了函数结构：def 函数名(参数):

# 这里的 def 都表示定义一个函数，后面都有自定义的函数名，然后括号里都有参数

#（练习 2 是文件夹路径，练习 3 是 Excel 文件路径，我的例子也是 Excel 文件路径）。

# 它们都是处理外部文件：练习 2：文件夹、练习 3：Excel 文件、我的例子：Excel 文件

# 它们都采用了“写一个功能块”的方式，把功能独立到一个函数里，

# 方便调用、维护、测试，也方便后面在主程序里调用。

# 问题2：为什么练习 2 用到了 import os，而练习 3 没有用到 os？

# 练习 2：需要列出文件夹里的所有文件，这就必须和操作系统打交道，

# 比如读取文件夹路径、遍历文件名，这时候就需要用到 os 模块。

# 练习 3：只是打开一个 Excel 文件并读取数据，不需要遍历文件夹，

# 也不需要拼接路径（只要 Excel 文件路径正确就可以），所以不需要 os。

# 为什么练习 3 用的是 from openpyxl import load\_workbook，而不是导入其他模块？

# 因为练习 3 只需要用到 Excel 文件的读取功能，

# 而 openpyxl 是专门处理 Excel 文件的库，里面的 load\_workbook 就是用来打开 Excel 的。

# 这段代码只需要 Excel 文件，不需要像练习 2 那样对文件夹进行处理，因此只需要导入 openpyxl。

# 问题 3：练习 2 和练习 3 的相同之处（除了函数体）：

# 都使用了 def 关键字定义了一个函数。

# 都在括号里接收了一个参数（练习 2 是文件夹路径，练习 3 是 Excel 文件路径）。

# 都把功能“封装”到一个函数里，方便重复使用，也方便主程序调用。

# 都用冒号 : 开始了函数体（虽然这里不深入函数体）。

# 这两个函数都可以被别人调用，比如：

# list\_files("D:/data")

# get\_first\_cell("D:/data/example.xlsx")

# 练习 2和练习3的共同点：都用了 def、参数、函数名、封装功能；

# 练习2和练习3的不同点（不管函数体）：

# 练习2需要遍历文件夹，必须导入os；练习3只需要打开excel，不需要操作文件夹，不需要os。

    try:

    #尝试运行这段代码

        wb = load\_workbook(excel\_path)

        ws = wb.active

        image\_loader = SheetImageLoader(ws)

#wb：Workbook（工作簿），代表Excel 文件（无论是新建的还是加载的）；

#load\_workbook：加载工作簿

#(excel\_path)：传入的参数，表示 Excel 文件的路径（文件位置）

#wb 就是内存中的 Excel 文件对象。

#从指定路径（excel\_path）加载（即：读取）一个已经存在的 Excel 文件，

#并把它存到变量 wb 里，方便后续进行操作（例如读取数据、提取图片、修改内容等）。

#ws：Worksheet（工作表），代表 Excel 文件中的一张表格（Sheet）；

#active：活动的，当前激活的、默认的（一般是第一个 Sheet）；

#wb.active：Excel 文件中“当前激活的工作表”。

#这里的 wb（Workbook）表示一个 Excel 文件，而 active 表示“当前激活的工作表”。

#对比了本人三个py文件：从表格导出图片、根据照片生成新列表、更新在线花名册三组：

#新建一个空白的 Excel用：wb = Workbook() 和 wb = openpyxl.Workbook() 都是新建 Excel，一模一样

#加载一个现有的 Excel用：wb = load\_workbook(excel\_path) 是读取已有文件（需要使用 openpyxl 库打开指定路径的 Excel 文件）

#所以这三个都是在不同情况下用的工具，目的不同，但都叫 wb（只是方便起名而已）。

#在 Excel 中，可以有很多个工作表（Sheet1、Sheet2、Sheet3），

#而 active 指的是当前被选中的那个工作表（也就是 Excel 里点击正在显示的那个）。

#ws = wb.active  从 Excel 文件（wb）里拿到当前的激活工作表（active）；

#并且把它赋值（把右边的结果放到左边的变量里）给一个变量 ws（代表“Worksheet”）；

#以后所有的写数据、读数据都可以通过 ws 这个变量来操作。

#ws = wb.active = 取 Excel 文件中当前激活的工作表 = 方便我们直接操作（写表头、写数据、加格式等）。

#SheetImageLoader是Excel工作表的图片加载器，是 openpyxl\_image\_loader 库里的一个类，专门用来从 Excel 表格（Sheet）中提取图片；

#ImageLoader是图片加载器，从 Excel 里提取嵌入的图片的工具。

#SheetImageLoader(ws)，创建一个“图片加载器”对象，并把 ws（活动工作表）传给它，

#表示“我要从这个工作表里提取图片”。

#“第三行整段意思：从工作表 ws 中创建一个图片加载器（SheetImageLoader），并把它存到 image\_loader 变量里，以便后面提取图片。”

#所以try：后面这三行代码是：打开Excel文件，准备读取其中的图片数据。

#ws：工作表对象，通过 wb.active 获取的 Excel 的活动工作表（Worksheet）。

#放在try下面：整个图片提取功能的初始化步骤，为后续的图片提取做准备。它们建立了访问 Excel 文件内容和图片的基础连接。

#如果这三个步骤中任何一个失败（比如文件不存在、文件格式错误、权限问题等），程序会跳转到 except 块，显示错误信息而不是崩溃。

        # 获取Excel所在文件夹路径

        output\_dir = os.path.dirname(excel\_path)

#output\_dir：这是一个变量名，代表“输出目录（文件夹）/输出目录（变量名）”。

#output是输出的意思，dir是 directory（目录）的缩写。

#=：右侧的值赋给左侧的变量。

#os：操作系统模块，它提供了很多与操作系统交互的功能，比如创建文件夹、列出文件等等。

#.：访问模块或对象中的属性或方法的点符号。

#path路径，：这是 os 模块中的子模块，专门用于处理路径。

#dirname()：这是 os.path 子模块中的一个函数，用于获取文件路径中的“目录名部分”。

#excel\_path：这是一个变量，代表 Excel 文件的完整路径（包含文件名）。

#os.path：访问 os 模块中的 path 子模块，用于处理路径字符串。

#os.path.dirname(...)：获取路径中目录部分的函数/使用 os.path 中的 dirname() 函数，获取指定路径的“目录部分”。

#Python 标准库中的一个函数/调用的具体函数。它接收一个文件路径作为输入，返回该路径的目录部分。

#例如，如果输入路径是 /home/user/documents/file.xlsx，它返回 /home/user/documents。

#excel\_path：表示 Excel 文件的完整路径/传入的 Excel 文件完整路径。这个目录路径将在脚本中用于保存提取的图片。

#os.path.dirname(excel\_path) ：获取excel\_path这个文件路径的目录部分。

#这个表达式调用 dirname 函数（function），传入 excel\_path 作为参数。它提取 excel\_path 的目录部分。

#意思就是：你给它一个完整的文件路径，它会把文件名去掉，只留下前面的文件夹路径。

#例如，如果 excel\_path 是 C:\Users\Documents\example.xlsx，它返回 C:\Users\Documents。

#整句：从选择的 Excel 文件的完整路径中，提取出所在的文件夹的路径，

#然后把这个文件夹路径赋值给 output\_dir 这个变量。 之后，程序就会用 output\_dir 来确定图片应该保存到哪里。

#简单说：获取 Excel 文件所在的文件夹路径，以便后续在同一文件夹里保存提取出来的照片。

#在脚本中，这一行至关重要，因为它决定了从 Excel 文件中提取的图片将保存的位置。

#脚本从 Excel 表格中提取图片并保存为 .jpg 文件，保存在与输入 Excel 文件相同的目录中。

#通过 os.path.dirname(excel\_path)，脚本确保输出图片保存在 Excel 文件所在的文件夹，方便用户查找。

#举例1：excel\_path = r"C:\Users\Username\Documents\my\_excel.xlsx"

#os.path.dirname(excel\_path)结果：C:\Users\Username\Documents

#output\_dir = "C:\Users\Username\Documents"后续保存的图片就会放在这个文件夹里。

#举例2：os.path.dirname("C:/Folder/subfolder/file.txt")

# 返回 "C:/Folder/subfolder"

#举例3：excel\_path = "D:\\工作\\员工照片.xlsx"

#output\_dir = os.path.dirname(excel\_path)

#output\_dir = "D:\\工作"

        row = 4    # 从第 4 行开始

        #row：行数；代码设置 row = 4，表示从 第 4 行 开始读取数据，前面三行默认是表头。

        extracted = 0   # 初始化照片计数器

        #extracted 是一个计数器，初始值为 0，表示一张还没提取。

        #初始化计数器，用来记录成功提取的照片数量。最后会弹窗显示“已提取 x 张照片”。

        #每成功保存一张照片，extracted += 1（即 extracted = extracted + 1），表示提取的照片数量 +1。

        #最后会显示 "已提取 X 张照片"，这个 X 就是 extracted 的值。

        #如果 张三、李四 的照片成功保存，extracted 会从 0 变成 2。

        #如果 王五 没有照片，extracted 不会增加。最终会显示 "已提取 2 张照片"。

        while True:    # 开始循环读取每一行

        #while True：开始一个无限循环，直到遇到break语句退出停止；

        #这是一个无限循环，表示从第 4 行开始，循环会一直检查 Excel 的每一行，直到遇到姓名单元格为空（if not name: break）为止。

            name\_cell = f"C{row}"    # 获取当前行的姓名位置（比如 C4）

            photo\_cell = f"D{row}"   #获取当前行的照片位置（比如 D4）

            #构造单元格位置字符串：

            #name\_cell 是姓名所在的单元格，例如 "C4"，"C5" ...， name\_cell 用于读取 姓名（假设 C 列存储的是姓名）。

            #f"C{row}" 是一个 f-string（格式化字符串），它会动态生成 Excel 单元格的坐标。

            #例如：

            #第一次循环 row = 4 → name\_cell = "C4"（第 4 行 C 列）  #第二次循环 row = 5 → name\_cell = "C5"（第 5 行 C 列）

            #name\_cell = f"C{row}" ： 构造姓名单元格的坐标（C列+当前行号）（姓名）

            #photo\_cell 是照片所在的单元格，例如 "D4"，"D5" ...，用于检查 D 列是否有照片，如果有就提取并保存。

            #f"D{row}" 会动态生成 D 列的单元格坐标：

            #row = 4 → photo\_cell = "D4"（第 4 行 D 列） row = 5 → photo\_cell = "D5"（第 5 行 D 列）

            #row += 1   每次循环后，行号 +1，处理下一行

            #photo\_cell = f"D{row}" ：构造照片单元格的坐标（D列+当前行号）（照片）

            #使用 f-string（格式化字符串）动态生成每行对应的单元格地址。

         #这样设计的好处是：自动适应不同行数的 Excel（不需要手动指定结束行）。遇到空行自动停止（if not name: break）。

         #整段：从第 4 行开始，一行一行地读取姓名（C列）和对应的照片（D列），并保存图片。

            name = ws[name\_cell].value

             #第一行：name = ws[name\_cell].value

            #name：变量名，表示“名字”，用来接收 Excel 表格中提取出来的姓名。

            #[ 和 ]：方括号用于根据单元格坐标（如 C4、C5）访问具体的单元格。

            #name\_cell：一个变量，存储当前行姓名所在的单元格位置，如 "C4"、"C5" 等。

            #.value：表示获取该单元格的内容值（也就是里面写的“姓名”）。

            #ws[name\_cell]：根据 name\_cell 的值（如 "C4"）获取表格中的具体单元格对象。

            #ws[name\_cell].value：表示“获取这个单元格中写的值”，通常就是某一位员工的名字。

            #把当前行的 C 列（即名字列）中的内容提取出来，赋值给变量 name。

            if not name:

            #not：布尔运算符，意思是“非”，用于判断变量是否为空、为假、为 None。

                #name：变量名，代表上一步提取到的姓名。

                #如果这一行没有名字（即 name 是空的、None 或空字符串），就执行接下来的语句。

                break  # 数据读取结束

                #break：Python 的控制语句，用于跳出并终止当前循环。

                #数据读取结束：注释，说明“跳出循环”的原因是“已经没有更多的数据可以读取了”。

                #如果没有名字（说明这一行的数据是空的），程序就认为数据已经处理完了，退出整个循环，不再继续往下读。

            #总结整体作用：这三行代码的功能是：从 Excel 表格中一行一行提取“姓名”列的数据，如果遇到空行就停止处理。

            #流程如下：1. 读取 C 列单元格（如 C4）的值，并赋给变量 name。

            #2. 如果这一行的 name 是空的（说明后面没有人了），就停止读取。

            #3. break 用于退出 while 循环，结束提取过程。)

            if image\_loader.image\_in(photo\_cell):

            #"\_"下划线 → 有意义的名字；

             #"."点号 →谁做的，属于谁的功能了，不可变，Python 的语法，表示“我要用这个对象的一个功能”；

            #"( )"括号 → 要传入数据，给了什么参数；

            #圆括号 ( ) 就像嘴巴，是用来说话、传递信息的。 当你想让某个功能知道一些事情的时候，就把信息放进括号里。

            #image\_loader：变量名，表示“图像加载器”对象；，一个能帮你找照片的对象；一个工具（就像“手机”）；

            #image\_in：方法名/函数名，一个可执行的动作，表示“检查某张照片是否存在”的功能；它的功能之一（就像“打电话”功能）；

            #(photo\_cell)：圆括号，表示你要给这个功能传一个参数（输入值），圆括号来表示“给这个动作传入什么东西”；

            #调用这个功能时，传入的参数（就像“输入电话号码”）；

            #括号里是你告诉它的信息 —— 我要哪张照片？在哪个单元格？；

            #"："冒号 → 接下来要做什么,要缩进写操作，表示下面要缩进写执行的内容；

            #使用 image\_loader 这个工具的 image\_in 功能，并检查 photo\_cell 这个单元格有没有照片。

            #if image\_loader.image\_in(photo\_cell):判断当前单元格有没有照片

            #image\_loader.image\_in(photo\_cell) 如果对应单元格有多个图片，它只会提取第一个，其他的忽略。可提醒用户 一个单元格只能插入一张图。

                image = image\_loader.get(photo\_cell)

                #.get 是 Python 中常见的一个方法名（method） ，表示：获取某个东西” ，比如获取一张照片、一段数据、一个文件…… ； 不可变，别人写好的功能；

                #它通常属于某个对象，比如：字典.get(key)获取键对应的值；图像加载器.get(photo\_cell)获取指定的照片；

                #使用 image\_loader 这个工具，根据 photo\_cell 提供的信息，找到并获取这张照片，然后保存到变量 image 中。

                #image = 把获取到的照片存到变量image里，方便后面使用。

                #使用 image\_loader 这个工具，根据 photo\_cell 提供的信息，找到并获取这张照片，然后保存到变量 image 中。

                #变量 = 对象.get(参数)

                #image = image\_loader.get(photo\_cell)：如果有，就把照片取出来

                filename = f"{name}\_大头照.jpg"

                #f"{变量}xxx" 就是“把变量的内容放进字符串里”。

                #filename = f"{name}\_大头照.jpg" 给这张照片起一个名字，比如“张三\_大头照.jpg”

                save\_path = os.path.join(output\_dir, filename)

                #save\_path：可以改。是定义的变量名，可以换成别的名字；

                #os.path.join：不可动，Python 标准库函数，用于安全地拼接路径；

                #（）：左右括号也不能动，语法符号，表示这是个函数调用；

                #dir directory 文件夹、目录

                #(output\_dir, filename)：可以改，是传入的两个参数，表示“文件夹路径”和“文件名

                #output\_dir = 输出目录/输出的目标文件夹，你要把程序处理完的文件、图片、数据等“输出”到哪里

                #我有一个文件名 filename（比如 “小明\_大头照.jpg”），还有一个文件夹路径 output\_dir（比如 “D:/照片备份”），

                #把它们拼在一起，得到完整的路径："D:/照片备份/小明\_大头照.jpg"

                #这行代码作用：把文件夹路径和文件名安全地拼在一起，生成一个完整的保存路径，方便后续保存文件。

                #把照片要保存的路径拼起来，比如“D:/学生照片/张三\_大头照.jpg”

                image.save(save\_path)

                #.save：方法名，表示“我要保存这张图”。

                #image.save(save\_path) 把照片保存到电脑上

                print(f"已保存: {save\_path}")

                #告诉用户“这张照片已经保存成功”，并显示路径；

                #print(f"{变量}") 就是“把变量的内容告诉我”。

                #print(f"已保存: {save\_path}")  打印一条信息告诉照片保存成功

                extracted += 1

                #extracted，计数器

                #变量 += 1 就是“我自己加一”。

                #记录“我已经成功保存了几张照片”。

                #成功提取的照片数量加一

            else:

            #else:“否则”，表示没有满足条件时要做的事，如果没找到照片。

                print(f"未找到{name}的照片")

                #print(f"{变量}")：把变量的内容告诉我。

                #提示说“某某同学的照片没找到”。

            #如果没有找到这张照片，请告诉我是谁的照片没找到。

            row += 1

            #当前的行号加一。

            #如果不写 row += 1，程序就会一直处理同一行，变成“死循环”。

            #row += 1 的作用是：让程序从当前行处理完后，自动跳到下一行，确保能读取表格中所有的数据。

            #处理完这一行后，准备下一行。

            #这一整段：判断→获取→命名→保存→计数” 是一套完整的流程。

            #这段代码的作用是：自动从 Excel 表格中提取照片，并按姓名命名保存到电脑上；如果找不到照片，就给出提示。

        messagebox.showinfo("完成", f"已提取 {extracted} 张照片，保存在\n{output\_dir}")

        #在这里调用弹出“成功”小窗口

        #messagebox.showinfo()，表面顺序：写在前面，实际运行顺序：后面运行（在提取完照片后）。

    except Exception as e:

    #except：在编程中表示“捕获异常”，表示“如果try 代码块中出错，就执行这里的代码”。

    #Exception：异常，错误 → Python 中所有错误的“父类”，可以捕捉所有常见的运行时错误。

    #except Exception：捕获所有类型的运行错误（除非特别严重）

    #as e ：把捕获到的错误信息命名为 e，可以通过 str(e) 来获取错误的描述

    #as：作为...，把发生的错误信息保存为变量名 e。

    #e：自定义的变量名，可改，用来存储实际出错信息的对象。

    #这句代码意思：“如果上面的 try: 代码块中出错（比如文件打不开、图片格式不对、找不到单元格等），

    #就进入这个 except 部分，把错误信息保存到变量 e 里。”

    #这就让程序不会直接崩溃，而是优雅地处理错误，告诉用户出错原因。

    #except Exception as e: 如果上面 try 代码出错了，就执行这里的语句，并把错误信息保存为变量 e。”

    #可以通过 str(e) 获取错误原因，用来提示用户或写入日志。

        messagebox.showerror("出错", f"提取失败：{str(e)}")

        #在这里调用弹出“出错”小窗口

        #代码中 messagebox.showinfo 和 messagebox.showerror 这两句“显示成功或失败的小窗口”。

# 这些函数虽然写在def main()代码“前面”，但它们是在 main() 里面被\*\*“后来”执行的\*\*，是在你点击了选择文件之后才触发的.

#函数定义（extract\_images\_from\_excel），表面顺序：写在上面，实际运行顺序：不会立刻执行。

##三、程序入口区（引导用户操作）

##作用：启动界面（隐藏主窗口）、弹出文件选择框。如果用户选了文件 → 调用上面的提取函数去处理。

##像你在启动一个“对话窗口”，引导用户选择文件，然后交给机器人去处理。

#def main()这一整段都是定义主函数

def main():

#main()函数调用，表面顺序：写在最后，最先执行（程序入口）。

#def 用法：定义一个函数

#函数内部自己搞定一切，不需要参数；函数需要外部告诉它一些信息，需要参数。

#main() 是一个入口函数，不是“必须加参数”的规定格式

#是否给 main() 加参数，取决于你需不需要从外部传入值给它。

#上下文中def两处区别：

#def extract\_images\_from\_excel(excel\_path) 是一个功能函数，接收一个 Excel 路径参数，专门负责“提取图片”。

#def main() 是主程序函数，不需要参数，负责控制整个流程，包括弹出文件选择框、调用提取函数。

    root = tk.Tk()

    #这行代码，表面顺序：写在后面，实际运行顺序：最先运行，初始化tkinter。

    #root：变量名，保存这个主窗口对象

    #tk：是 import tkinter as tk 中的 tk，指向 tkinter 库

    #Tk()：创建一个 Tkinter 应用窗口的“主窗口”（顶层界面）

    #创建了一个 Tkinter 的主窗口，并保存在变量 root 中。

    #如果只有上面一句root = tk.Tk()则是默认创建一个空白页面主窗口。

    #如果不写tk.Tk()，弹窗会报错；

    #如果想弹出一个“选择文件”的窗口（filedialog），而不想出现那个空白的主界面，就加上下面这句：

    root.withdraw()

    #withdraw()：是 Tk() 的方法，表示“隐藏这个窗口”，不让它弹出来。

    #把 tkinter 创建的主窗口隐藏掉，不让它出现在屏幕上。

    #意思：用 tkinter 的功能，不要显示主窗口，悄悄地处理。

    #如果不写 withdraw()，主窗口会跳出来，会出现一个空白主窗口，体验不好。加上，只显示想要的选择框、提示框，更简洁。

    #虽然 root = tk.Tk() 在代码的行数上是在后面，但在程序运行的顺序上，它是“先执行的”，

    #因为 main() 是最早被调用的函数，messagebox 的弹窗在后面函数中才执行。

    file\_path = filedialog.askopenfilename(

        title="请选择包含照片的Excel文件",

        filetypes=[("Excel 文件", "\*.xlsx \*.xlsm")]

    )

    #这个弹窗需要 tkinter 的主窗口上下文（虽然隐藏了，但依然存在）。

    #filedialog和askopenfilename还有符号都是固定的。filedialog是tkinter 模块中的固定名称，

    #askopenfilename是 filedialog 中的方法名，不能随便改。符号是py的语法结构，不能改。

    #filedialog是tkinter（图形界面库）中的一个子模块（文件对话框模块），

    #专门用来弹出文件或文件夹选择窗口，让用户点击选择文件。

    #可以通过代码导入它：from tkinter import filedialog（这也就是前面出现过的）

    #filedialog提供了一组函数在程序里实现：

    #让用户选择一个文件（打开）对应函数：askopenfilename()

    #让用户选择多个文件（打开）对应函数：askopenfilenames()

    #让用户选择一个文件（保存）对应函数：asksaveasfilename()

    #让用户选择一个文件夹对应函数：askdirectory()

    #这些函数调用后，都会弹出一个像这样的系统窗口：打开文件窗口（可以浏览电脑、点击选择文件）

    #filedialog.askopenfilename(...)弹出一个“打开文件”的选择框，让用户选择一个 Excel 文件。

    #title="..."设置弹出窗口的标题，用来提示用户：“请选择包含照片的Excel文件”。

    #filetypes=[("Excel 文件", "\*.xlsx \*.xlsm")]控制用户只能选择 Excel 文件，

    #这个参数的格式是一个列表，列表里的每一项是一个元组：("显示的名字", "匹配的扩展名")

    #[ ] 是 Python 中的列表（List）符号，在这里表示：“文件类型的选项是一个列表（可以列出一个或多个文件类型）”。

    #为什么这里用[ ]？因为filetypes支持多个文件类型，所以用列表来传递。

    #（）元组符号，表示一组固定配对的数据项，在这里表示“显示的名字”和“匹配的扩展名”是一对。

    #这一段：弹出一个“打开文件”的窗口，标题是“请选择包含照片的Excel文件”，

    #只允许选择后缀名是 .xlsx 或 .xlsm 的文件，然后把用户选中的文件路径（字符串）保存到变量 file\_path 里。

    #整个作用：这句代码是程序中\*\*“输入文件”的入口\*\*，没有它，用户就没法告诉程序：要处理哪个 Excel 文件。

    if not file\_path:

        sys.exit()  # 用户取消

# 在我的 Excel 图片提取例子里，为什么要用 sys？

# 当用户没有选择文件时（filedialog 取消了），程序就必须退出。

# 这时候就写：

# import sys

# if not file\_path:

#     sys.exit()

# 否则程序继续运行就会报错。

# 题目关键词：“没选文件就退出” ➔ sys.exit() ➔ 需要 import sys。

    extract\_images\_from\_excel(file\_path)

    #extract\_images\_from\_excel(file\_path) 是核心功能的“启动开关”。

    #你虽然定义了 extract\_images\_from\_excel() 函数，但如果不调用它，它就不会自动执行。

    #就像你准备好了工具，但不按“开始”按钮，工具不会动。

    #作用是调用自定义的函数 extract\_images\_from\_excel 来执行提取 Excel 中嵌入照片的主要逻辑。

    #定义函数 ≠ 执行函数。想让函数“动起来”，就必须手动调用它。

    #有这行代码：调用函数 → 提取图片、命名、保存 → 弹窗提示

    #没这行代码：函数没调用 → 什么都不做，程序无实际效果

    #如果没有，用户虽然选择了 Excel 文件，但什么都不会发生。

##四、启动控制区（程序入口开关）

##作用：判断当前文件是不是“主程序”在运行。如果是，就执行 main() 启动整个流程。

##相当于“启动按钮”，不按下这个按钮，一切都不会发生。

#真正入口，从这里开始执行。

if \_\_name\_\_ == "\_\_main\_\_":

    main()

#\_\_name\_\_：是 Python 自带的内置变量，表示当前模块的名字

#== "\_\_main\_\_"  固定判断：当前是不是“主程序”

#作用：只有当这个 .py 文件是作为“主程序”运行时，才会执行 main() 函数。

#如果这是我直接运行的主程序，那就去执行 main() 里面的内容

#main() 可以换成你定义的任何函数名，比如 start() 或 run()，只要上面你有定义它

#函数体内执行的内容：main() 里面你可以写任何你希望启动时执行的逻辑

#if \_\_name\_\_ == "\_\_main\_\_" 是判断“要不要启动”的条件