**University Lecture: Python Programming with Tkinter**

**Lecturer:** Ahmed Sami  
**Section 11 - Lesson 51 Part 1**  
**Topic: Introduction to GUI Programming with Tkinter**

**Introduction**

Good morning, students. Today, we embark on an exciting journey into the world of Graphical User Interface (GUI) programming using Python's Tkinter library. In this first part of Lesson 51, we will explore the fundamental concepts of creating desktop applications with graphical interfaces. This marks a significant transition from console-based programs to interactive, user-friendly applications that feature windows, buttons, and other visual elements.

The ability to create GUI applications is an essential skill for modern programmers. It allows us to develop software that non-technical users can easily interact with, making our programs more accessible and professional. Tkinter, which stands for "Tk interface," is Python's standard GUI library and comes pre-installed with most Python distributions, making it an ideal starting point for learning GUI programming.

**Body of the Lecture**

Let us now examine our code systematically, understanding each line and its purpose in creating a basic GUI application.

**Line-by-Line Code Analysis**

**Line 1: from tkinter import \***

This line imports all classes, functions, and constants from the tkinter module into our current namespace. The asterisk (\*) is a wildcard operator that means "import everything." While this approach is convenient for small programs and learning purposes, it is important to note that in larger production code, explicit imports are preferred to avoid namespace pollution and improve code clarity. The tkinter module provides us with all the necessary tools to create windows, widgets, and handle user interactions.

**Line 2: Empty line**

This blank line serves as a visual separator between the import statement and the main program logic. Following proper code formatting conventions enhances readability and is considered a best practice in Python programming.

**Line 3: age\_app = Tk()**

This crucial line creates the main application window object. The Tk() class is the foundation of any Tkinter application - it represents the root window that will contain all other GUI elements. We assign this window object to the variable age\_app, which we can then use to configure and control our application. Think of this as creating the canvas upon which we will paint our entire user interface. Every Tkinter application must have exactly one root window created using Tk().

**Line 4: age\_app.title("Calculate Your Age")**

Here, we set the title of our application window using the title() method. This text appears in the window's title bar at the top of the application frame. The string "Calculate Your Age" suggests that this application will eventually be developed into an age calculator. Setting a meaningful title is important for user experience as it immediately communicates the purpose of the application to the user.

**Line 5: age\_app.mainloop()**

This final line is perhaps the most important in any Tkinter application. The mainloop() method starts the event loop of the application. This event loop is responsible for listening to user interactions such as mouse clicks, keyboard input, and window events. It keeps the window open and responsive until the user closes it. Without this line, the window would appear and immediately disappear, as the Python script would complete execution. The mainloop continuously checks for events and dispatches them to the appropriate handlers, making the application interactive and responsive.

**Understanding the Program Flow**

When this program runs, it follows a sequential execution pattern. First, it loads the tkinter library into memory. Then, it creates a window object that serves as the container for our application. After setting the window's title to provide context to the user, it enters the main event loop, which keeps the application running and responsive to user actions.

This basic structure forms the foundation upon which we can build more complex GUI applications. In subsequent lessons, we will add widgets such as buttons, text fields, and labels to create a functional age calculator application.

**Conclusion**

In this introductory lesson, we have established the fundamental framework for GUI programming in Python using Tkinter. We learned how to import the necessary library, create a main application window, set its title, and maintain it in an active state using the event loop. These four essential components - import, window creation, configuration, and event loop - form the backbone of every Tkinter application you will create.

As we progress through the subsequent parts of this lesson, we will build upon this foundation by adding interactive elements and implementing the age calculation functionality. Remember that GUI programming requires a different mindset from console programming - we must think in terms of events, user interactions, and visual layouts rather  
<Streaming stoppped because the conversation grew too long for this model>

section 11 lesson 51p1 python code tested

from tkinter import \* # Import all classes and functions from the tkinter library.

age\_app = Tk() # Create the main application window.

age\_app.title("Calculate Your Age") # Set the title of the window.

age\_app.mainloop() # Run the application's event loop to keep it open.

**Lecture: Section 11 Lesson 51 Part 2**

**Introduction to GUI Development with tkinter**

Good morning, class. Today, we begin our exploration into graphical user interface (GUI) development using Python's standard tkinter library. A GUI provides a visual way for a user to interact with a program, using elements like windows, buttons, and text fields, which is a significant step beyond the command-line interface we have used so far. The code we will analyze today creates a simple window for an age calculation application.

**Detailed Code Explanation**

Let's break down the provided Python code line by line to understand how it constructs our first GUI application.

1. from tkinter import \*
   * This line is our first step. It imports all of the classes, functions, and variables from the tkinter library. The \* symbol is a wildcard that signifies "all." By doing this, we can directly use Tk(), Label, and other tkinter components without needing to prefix them with tkinter..
2. age\_app = Tk()
   * Here, we create the main application window, which is also known as the root window. The Tk() class is the primary class in tkinter and is responsible for creating the main application window. We assign this main window object to a variable named age\_app for easy reference throughout our code.
3. age\_app.title("Calculate Your Age")
   * This line customizes our main window. We are calling the .title() method on our age\_app object. The string argument "Calculate Your Age" sets the text that will appear in the title bar of the application window.
4. age\_app.geometry("400x200")
   * The .geometry() method is used to define the initial dimensions of the window. The string "400x200" specifies the window's width (400 pixels) and height (200 pixels). This allows us to control the size of our application window when it first opens.
5. the\_text = Label(age\_app, text="Write Your Age", height=2, font=("Arial", 20))
   * This line introduces our first GUI widget, the Label. A Label widget is used to display static text or images in a window. We are creating a new Label object and assigning it to the the\_text variable.
   * Let's look at its arguments:
     + age\_app: This is the first argument and it specifies the parent widget. In this case, our label will be placed inside the age\_app window.
     + text="Write Your Age": This argument sets the actual text that the label will display.
     + height=2: This sets the height of the label in text units.
     + font=("Arial", 20): This is a tuple that specifies the font family ("Arial") and the font size (20).
6. the\_text.pack()
   * The .pack() method is one of tkinter's geometry managers. Its job is to arrange the widget in a block within its parent widget. By calling the\_text.pack(), we are telling tkinter to place our Label object (the\_text) into the main age\_app window.
7. age = StringVar()
   * This line creates a special variable type from tkinter called StringVar. These variable types are unique to tkinter widgets and are used to hold the values of widgets that can change, such as text entry fields. Using a StringVar allows a widget to be automatically updated when its linked variable changes, and vice-versa.
8. age.set("00")
   * The .set() method is used to assign an initial value to a StringVar. In this case, we are setting the value of our age variable to the string "00".
9. age\_app.mainloop()
   * This is a critical and required line for every tkinter application. The .mainloop() method starts the event loop of the application. The event loop listens for events like mouse clicks, keyboard presses, and window resizing, and keeps the window displayed on the screen. The program will not run and display the window without this line.

**Conclusion**

In today's lecture, we have successfully built our very first GUI application. We learned how to create a main application window, set its properties like the title and size, and add a simple Label widget to display text. Most importantly, we understood the purpose of the mainloop() method, which is essential for any tkinter program to function.

This is just the beginning of our journey. We will be building on these concepts in future lectures, adding more widgets like buttons and text entry fields to make our application interactive. Please review the code below, and feel free to ask any questions.

**Complete Python Code for Section 11 Lesson 51 Part 2**

section 11 lesson 51p2 python code tested

from tkinter import \* # Imports all modules from the tkinter library.

age\_app = Tk() # Creates the main application window.

age\_app.title("Calculate Your Age") # Sets the title of the application window.

age\_app.geometry("400x200") # Sets the dimensions of the window (width x height).

the\_text = Label(age\_app, text="Write Your Age", height=2, font=("Arial", 20)) # Creates a text label widget.

the\_text.pack() # Places the label widget into the main window.

age = StringVar() # Creates a special tkinter variable to hold a string.

age.set("00") # Sets the default value for the 'age' variable.

age\_app.mainloop() # Runs the application's event loop to keep it open and responsive.

**Lecture: Section 11 Lesson 51 Part 3**

**Adding Interactive Widgets to a tkinter App**

Good morning, class. In our last lecture, we successfully created the foundation of our age calculator application. Today, our goal is to make it interactive by allowing the user to provide input and then trigger an action. We will be adding two new widgets to our window: the Entry widget for text input and the Button widget for user actions.

**Detailed Code Explanation**

Let's examine the new lines of code and understand their purpose.

1. age\_input = Entry(age\_app, width=2 , font=("Arial", 30), textvariable=age)
   * This is a new line of code that introduces the **Entry widget**. An Entry is a single-line text input field where a user can type information.
   * Let's look at its arguments:
     + age\_app: This is the parent widget, specifying that our input field will be placed inside the main application window.
     + width=2: This sets the width of the input field in number of characters. A value of 2 is chosen here to accommodate a two-digit age.
     + font=("Arial", 30): This tuple sets the font and size of the text that the user types.
     + textvariable=age: This is a very important argument. It links the value of this Entry widget directly to the StringVar variable we created earlier named age. Any change the user makes in the input field will automatically update the age variable, and vice-versa.
2. age\_input.pack()
   * Just like with our Label widget, we need a geometry manager to place the Entry widget onto the window. The .pack() method places the input field below the previous widget in the window.
3. button = Button(age\_app, text="Calculate ", width=20, height=2, bg="#0e1063", fg="white", borderwidth=0)
   * This line introduces the **Button widget**. A Button is used to trigger an action when clicked.
   * The arguments are similar to those we have seen before, but with some new styling options:
     + age\_app: The parent widget.
     + text="Calculate ": The text that will be displayed on the button.
     + width=20 and height=2: These set the dimensions of the button.
     + bg="#0e1063": This sets the background color of the button using a hexadecimal color code.
     + fg="white": This sets the foreground (text) color.
     + borderwidth=0: This removes the default border around the button, giving it a flatter appearance.
4. button.pack()
   * We use .pack() again to add the button to the main window, placing it below the input field.

**Conclusion**

In today's lecture, we have successfully enhanced our GUI application by adding an interactive Entry widget for user input and a Button to perform an action. You have now learned how to use two of the most fundamental tkinter widgets, which are essential for building any functional application. We also saw how the textvariable argument can seamlessly link a widget to a StringVar.

Our application now has all the necessary components to get input from the user. In our next lecture, we will connect the button to a function that will perform the actual age calculation. Please review the complete code below, and feel free to ask any questions you may have.

**Complete Python Code for Section 11 Lesson 51 Part 3**

section 11 lesson 51p3 python code tested

# import  tkinter

from tkinter import \* # Imports everything from the Tkinter library.

# Create the Main App Window

age\_app = Tk() # Creates the main application window.

# Change App title

age\_app.title("Calculate Your Age") # Sets the title of the window.

# Set Dimensions

age\_app.geometry("400x200") # Sets the width and height of the window.

# Write Age Label

the\_text =Label(age\_app, text="Write Your Age", height=2, font=("Arial", 20)) # Creates a label with text.

# Place The Text Into The Main Window

the\_text.pack() # Adds the label to the window.

# Age Variables

age = StringVar() # Creates a variable to store the age as a string.

# Set Default Value For Age

age.set("00") # Sets the default value of the age variable to "00".

# Create the input for age

age\_input = Entry(age\_app, width=2 , font=("Arial", 30), textvariable=age) # Creates an input field for the user to type their age.

# Place the input into the main window

age\_input.pack() # Adds the input field to the window.

# Create the calculate button

button = Button(age\_app, text="Calculate ", width=20, height=2, bg="#0e1063", fg="white", borderwidth=0) # Creates a button with custom text and colors.

# Place button in the main window

button.pack() # Adds the button to the window.

# Run App Infinitely

age\_app.mainloop() # Starts the Tkinter event loop, which runs the application.

from tkinter import \* # Imports everything from the Tkinter library.

# Create the Main App Window

age\_app = Tk() # Creates the main application window.

# Change App title

age\_app.title("Calculate Your Age") # Sets the title of the window.

# Set Dimensions

age\_app.geometry("400x200") # Sets the width and height of the window.

# Write Age Label

the\_text = Label(age\_app, text="Write Your Age", height=2, font=("Arial", 20)) # Creates a label with text.

# Place The Text Into The Main Window

the\_text.pack() # Adds the label to the window.

# Age Variables

age = StringVar() # Creates a variable to store the age as a string.

# Set Default Value For Age

age.set("00") # Sets the default value of the age variable to "00".

# Create the input for age

age\_input = Entry(age\_app, width=2, font=("Arial", 30), textvariable=age) # Creates an input field for the user to type their age.

# Place the input into the main window

age\_input.pack() # Adds the input field to the window.

# Create the calculate button

button = Button(age\_app, text="Calculate", width=20, height=2, bg="#0e1063", fg="white", borderwidth=0) # Creates a button with custom text and colors.

# Place button in the main window

button.pack() # Adds the button to the window.

# Run App Infinitely

age\_app.mainloop() # Starts the Tkinter event loop, which runs the application.

**Lecture: Section 11 Lesson 51 Part 4**

**Adding Functionality to the Button Widget**

Good morning, class. Our goal today is to breathe life into the button we created in the last session. Currently, clicking the "Calculate" button does nothing. We need to tell our program what to do when that specific action (a button click) occurs. We will achieve this by creating a Python function and linking it to the button. We will also learn how to use a new module to display the results in a pop-up message box.

**Detailed Code Explanation**

Let's break down the new lines of code and understand their significance.

1. from tkinter import messagebox
   * This is the first new line. While tkinter provides a vast array of widgets, some functionalities, like pop-up message boxes, are in a separate module. This line specifically imports the messagebox module, allowing us to use its functions to display informational pop-ups to the user.
2. def calc():
   * This line defines a new function named calc. In Python, we use the def keyword to create a function. All the code indented under this line will be executed every time the calc function is called. We are placing all our calculation logic inside this function.
3. the\_age\_value = int(age.get())
   * Inside our calc function, we first need to retrieve the value the user typed into the Entry widget. The age.get() method retrieves the string value currently stored in our StringVar variable, age. Since we need to perform mathematical operations, we must convert this string to an integer using the int() function. The result is stored in the the\_age\_value variable.
4. months = the\_age\_value \* 12
   * This is a simple calculation that multiplies the user's age by 12 to find the age in months. The result is stored in the months variable. Similarly, we calculate weeks and days.
5. line\_one = f"Your Age In Months Is: {months}"
   * Here, we use an **f-string** (formatted string literal) to create a clean, readable output string. The f before the opening quote indicates that this is a formatted string. We can then embed variables directly inside the string by enclosing them in curly braces {}. This creates a clear message like "Your Age In Months Is: 240" if the age is 20. We do the same for line\_two and line\_three.
6. all\_lines = [line\_one, line\_two, line\_three]
   * This line creates a simple Python list to hold all three of our formatted output strings. This is a convenient way to group the data we want to display.
7. messagebox.showinfo("Your Age In All Time Units", "\n".join(all\_lines))
   * This is the final and most exciting part. We are calling the showinfo function from the messagebox module to display a pop-up window.
   * The first argument, "Your Age In All Time Units", sets the title of the pop-up window.
   * The second argument, "\n".join(all\_lines), is the message body. The .join() method is a powerful string method that concatenates all items in a list into a single string, using the string it is called on ("\n") as the separator. In this case, it inserts a newline character between each line, ensuring that each age calculation appears on a new line in the pop-up.
8. button = Button(..., command=calc)
   * This is the last and most critical addition. We have modified the Button widget creation to include the command argument. The value of this argument is the name of the function we want to execute when the button is clicked. When the user clicks the "Calculate" button, tkinter automatically calls the calc function for us, running all the code inside it.

**Conclusion**

In this lecture, we have completed the core functionality of our age calculator application. You have now learned how to define and use functions in tkinter, how to display information to the user using the messagebox module, and most importantly, how to link a button's click event to a specific function using the command argument. Our application can now receive user input, perform a calculation, and present the results in a user-friendly way.

This is a complete, working version of our simple application. Please review the code below to see all the pieces working together.

**Complete Python Code for Section 11 Lesson 51 Part 4**

# import tkinter

from tkinter import \* # Imports all widgets from the Tkinter library.

from tkinter import messagebox # Imports the messagebox module for pop-up windows.

# Create the Main App Window

age\_app = Tk() # Creates the main application window.

# Change App title

age\_app.title("Calculate Your Age") # Sets the title of the window.

# Set Dimensions

age\_app.geometry("400x200") # Sets the width and height of the window in pixels.

# Write Age Label

the\_text =Label(age\_app, text="Write Your Age", height=2, font=("Arial", 20)) # Creates a text label to instruct the user.

# Place The Text Into The Main Window

the\_text.pack() # Places the label widget into the window.

# Age Variables

age = StringVar() # Creates a special Tkinter variable to store the age input.

# Set Default Value For Age

age.set("00") # Sets the initial value of the age variable to "00".

# Create the input for age

age\_input = Entry(age\_app, width=2 , font=("Arial", 30), textvariable=age) # Creates an input field for the user to enter their age.

# Place the input into the main window

age\_input.pack() # Places the input field widget into the window.

# Create the calculate function

def calc(): # Defines a function to perform the age calculation when called.

    # Get Age In Years and convert to an integer

    the\_age\_value = int(age.get()) # Retrieves the value from the `age` variable and converts it to an integer.

    # Get Time Units

    months = the\_age\_value \* 12 # Calculates the number of months.

    weeks = months \* 4 # Calculates the number of weeks.

    days = the\_age\_value \* 365 # Calculates the number of days.

    # Create Text Lines

    line\_one = f"Your Age In Months Is: {months}" # Formats a string to display age in months.

    line\_two = f"Your Age In Weeks Is: {weeks}" # Formats a string to display age in weeks.

    line\_three = f"Your Age In Days Is: {days}" # Formats a string to display age in days.

    # Group all lines

    all\_lines = [line\_one, line\_two, line\_three] # Combines all lines into a list.

    # Show The Message Box

    messagebox.showinfo("Your Age In All Time Units", "\n".join(all\_lines)) # Displays a pop-up box with the calculated age details.

# Create the calculate button

button = Button(age\_app, text="Calculate ", width=20, height=2, bg="#0e1063", fg="white", borderwidth=0, command=calc) # Creates a button that executes the `calc` function when clicked.

# Place button in the main window

button.pack() # Places the button widget into the window.

# Run App Infinitely

age\_app.mainloop() # Starts the Tkinter event loop to run the application and wait for user interaction.

Hello, students. I'm Ahmed Sami, and welcome to the final lecture in our series on building the "Calculate Your Age" application. We've successfully written all the code needed for our app, and now we will focus on the last and most exciting step: packaging our Python script into a standalone executable file that can be shared with anyone, even if they don't have Python installed.

This lecture will cover the command-line tools necessary for this process. As this is not a traditional coding lecture, the Python file provided below is a copy of our final, working script, which is the file we will be packaging.

Lecture: Section 11 Lesson 51 Part 5

This final lecture is dedicated to a crucial step in application development: deployment. Until now, our program has required a Python interpreter to run. With the help of a tool called PyInstaller, we can convert our .py file into a single, executable .exe file for Windows.

Introduction

Good morning, class. We have completed the coding phase of our project. While our application works perfectly on our machine, it's not very useful if we can't share it. Today, we'll learn how to transform our script into a shareable application. This process is called "packaging" or "freezing."

Body

The process of packaging our script is done through the command line. We'll be using pip, the standard package installer for Python, to get our main tool, PyInstaller. Then, we'll run PyInstaller on our script.

Installing PyInstaller

pip install pyinstaller

This is the first command you need to run in your terminal or command prompt.

pip: This is Python's package manager. Think of it as a tool that helps you download and install libraries that aren't included with Python by default.

install: This is the command to install a package.

pyinstaller: This is the name of the package we want to install. PyInstaller is a third-party library that analyzes your Python script and packages it along with all the necessary dependencies and libraries into a self-contained executable.

Converting the Script to an Executable

pyinstaller --onefile lesson51p5.py

Once PyInstaller is installed, you can use this command to create your executable.

pyinstaller: This is the program we are running.

--onefile: This is a very important option. It instructs PyInstaller to bundle everything—your code, the tkinter library, and all other dependencies—into a single, standalone executable file. Without this flag, it would create a folder with many different files.

lesson51p5.py: This is the name of the Python script we want to package. You must be in the same directory as this file in your terminal for the command to work correctly.

Finding the Executable

explorer .

After the pyinstaller command finishes, it will create a folder named dist (short for "distribution") in your current directory. Inside this folder, you will find your new executable file, lesson51p5.exe.

This final command is a quick shortcut on Windows to open the file explorer for the current directory (.), so you can easily navigate to the dist folder.

Conclusion

Congratulations! You have now learned how to turn your Python code into a professional, shareable application. This final step is what separates a script from a deployable program. By using PyInstaller, you can share your creations with a wider audience who don't need to know anything about Python.

The code below is our final script from the previous lecture, which is the file we will be using with the pyinstaller command.

This concludes our lecture series on building and deploying a simple GUI application with tkinter. We have covered everything from basic window creation to packaging the final product.

What would you like to explore next? We could dive into more advanced tkinter features, begin a new project, or move on to a completely different topic.

section 11 lesson 51p5 python code

pip install pyinstaller # This command uses the \*\*pip\*\* package manager to \*\*install\*\* the pyinstaller

pyinstaller --onefile 51p4.py # This command uses the installed pyinstaller library to turn the Python script \*\*`51p4.py\*\* into a single, standalone \*\*.exe executable file.

explorer . # This command, used on Windows, opens a File Explorer window for the current directory

**Lecture: Section 11 Lesson 52**

**Introduction**

Good morning, class. In today's session, we will tackle a fundamental task in data processing. Imagine you have a large amount of raw data stored in a simple text file. This data is often disorganized, lacking proper headers or structure. Our goal today is to convert this raw data into a clean, well-structured CSV file, a format that is easily readable by other programs and humans. To do this, we will use the **pandas** library, which is the go-to tool for data analysis in Python.

**Body**

Before we dive into the code, let's look at the source data file, Points.txt.

As you can see, it's a very simple file. It contains a list of names and their corresponding scores, but it lacks any column headers to identify what the data represents. This is a common scenario. Now, let's examine the Python script that will transform this data.

1. **Importing the pandas Library**
   * import pandas
   * The first line of our script is a simple import statement. We are importing the entire pandas library, which contains all the functions and data structures we need for data manipulation. We'll use the alias pandas to refer to it throughout our code.
2. **Reading the Text File**
   * my\_data = pandas.read\_csv("Points.txt")
   * This is where the magic begins. The pandas.read\_csv() function is incredibly versatile. Despite the function's name, it can read data from a variety of delimited text files, including our simple .txt file. It reads the data and automatically organizes it into a **DataFrame**, which you can think of as a powerful, spreadsheet-like table.
3. **Assigning Column Headers**
   * my\_data.columns = ["Name", "Points", ]
   * After reading the data, our DataFrame is missing proper column names. This line of code allows us to assign a list of new names to the columns of our DataFrame. This makes the data much more meaningful and easier to work with. We are explicitly telling pandas that the first column should be called "Name" and the second should be called "Points".
4. **Exporting to a New CSV File**
   * my\_data.to\_csv("Points.csv", index=None)
   * This is the final, crucial step. The .to\_csv() method is used to write the contents of our DataFrame to a new file. We provide the desired filename, "Points.csv". The parameter index=None is very important. By default, pandas adds a numerical index column to our DataFrame. This parameter prevents that index from being written into our output file, ensuring the CSV is clean and only contains the data we want.

**Conclusion**

We have successfully taken a simple, unstructured text file and, in just a few lines of code, converted it into a clean, well-structured CSV file. This demonstrates the power and efficiency of the **pandas** library for handling common data tasks. The entire process, from reading to cleaning to exporting, is streamlined and highly readable.

This concludes our lecture on data conversion. The full, commented code is provided below for your reference.

# This script uses the pandas library to convert data from a simple text file to a structured CSV file.

# The contents of the source text file 'Points.txt' are as follows:

# Name, Points

# Osama, 100

# Ahmed, 120

# Sayed, 320

# Mahmoud, 40

# Ali, 90

# Sameh, 500

# Eman, 300

# Saleh, 300

# Import the pandas library, which is essential for data manipulation and analysis.

import pandas

# Read the data from the text file "Points.txt" into a pandas DataFrame.

# The read\_csv() function is flexible and can handle text files delimited by commas.

my\_data = pandas.read\_csv("Points.txt")

# Assign clear and descriptive column names to the DataFrame.

# This makes the data easier to understand and work with.

my\_data.columns = ["Name", "Points", ]

# Write the DataFrame to a new CSV file named "Points.csv".

# The 'index=None' argument prevents pandas from writing the DataFrame's index as an extra column.

my\_data.to\_csv("Points.csv", index=None)

**Lecture: Section 11 Lesson 53**

**Introduction**

Good morning, class. In our last session, you learned how to use PyInstaller to create a standalone executable file from your Python script. However, if you've tried running that executable, you might have noticed a command-line window popping up in the background. While this is fine for debugging, it can look unprofessional for a final product. Today, we'll learn a simple modification to our PyInstaller command to hide that window and create a cleaner user experience.

**Body**

We'll be modifying our PyInstaller command to include a new option. Let's look at the command.

1. **Installing PyInstaller**
   * pip install pyinstaller
   * This command is the same as before. You only need to run it once to install the PyInstaller library on your system. It's the essential first step to getting the tool you need for packaging.
2. **Converting the Script to an Executable with a Hidden Console**
   * pyinstaller -w --onefile calculate\_age.py
   * This is our new, improved command. You can see we've added a new option: -w.
   * -w: This is the key. The -w stands for "windowed." It instructs PyInstaller to create an executable that does **not** launch with a command-line or console window in the background. This makes the application look more professional and is a best practice for GUI applications like the one we've built.
   * --onefile: This option, as we learned, bundles your entire application into a single .exe file, making it easy to share.
   * calculate\_age.py: This is the name of our Python script. Note that we've used a new, more descriptive name for our file to reflect its purpose.

**Conclusion**

By adding just one simple flag, -w, we have significantly improved the professionalism of our application. This small change demonstrates an important principle in software development: paying attention to the user experience. A seamless and clean interface, even for a simple application, makes a big difference.

The code below is the full Python script we will be packaging using our new command.

section 11 lesson 53 python code

pip install pyinstaller

pyinstaller -w --onefile calculate\_age.py