Hello everyone!

This is Muhammad Azizul Hakim, Embarcadero Python Technical Blog Writer.

You can call me Muhammad, for short.

Welcome to the “Using Python Libraries from Delphi” session.

This session is very useful for the

Delphi Developer who want to add powerful Python libraries into their program,

Python Developer who want to easily create Desktop or custom apps, outside the common Tkinter GUI, Jupyter, or Streamlit ecosystem,

also for Programmers or Tech Enthusiast who wants to witness the magic of combining 2 programming language giants.

The goal of this session is to attract Python developers

to use Python4Delphi to create GUIs and learn more about Delphi.

To add Python Data Science library powers and simplicity for Delphi developers.

And to inspire the open-source community to develop more advanced use cases using Python4Delphi.

So, this is not a competition.

Instead, this is a demonstration of

how you can always find a specific task that a tool is better for, as we know that

no one tool is best for all tasks.

So our agenda today, we will talk briefly about the philosophy of Delphi and Python,

Introduction to Python4Delphi

then we’ll take a look at the installation of all the prerequisites for this demo

and finally, we will dive into the code and demo, for both Delphi and Python parts,

including tweaking parts for each demo

as it is a core part of this session.

The following Demo GUI will increase in complexity.

Demo 01 is about a GUI that will do web scraping, and print out the scraping results on the TMemo.

Demo 02, would show 2 outputs on the GUI, TMemo for text output, and TImage for image or plotting results.

And we will add TStringGrid for showing the data sets or any files in .csv format in Demo 03,

And finish up with some useful links for you to read and try more.

Before we dive in more into Python4Delphi, the following is Delphi’s DNA and Zen of Python,

the philosophy behind the two programming languages.

In a lot of ways, Delphi and Python are have similar philosophies,

but there are definitely some fundamental differences,

for example, Delphi is perfect for developer’s productivity,

as we really can easily and rapidly build end-to-end working product using Delphi,

and it is so visual, with drag and drop interfaces provided by Embarcadero’s RAD Studio.

On the other hand, even though Python is also great for developer’s productivity,

which you can solve complex problem using Python only in a few lines of code,

it is still oriented as a scripting language.

Not visually oriented, nor have drag and drop features.

So, in conclusion, it pretty much makes sense to combine the two.

What is Python4Delphi?

Python4Delphi or P4D, for short,

is a set of components that wrap up the Python DLL into Delphi and Lazarus (FPC).

They let Delphi developers easily execute Python scripts,

and create new Python modules, and new Python types.

On the other hand,

P4D empowers Python users with Delphi’s award-winning VCL functionalities for Windows

which enables us to build native Windows much faster.

This integration enables us to create a modern GUI with Windows 10 looks and responsive controls for our Python applications.

Python4Delphi comes with an extensive range of demos, use cases, and tutorials.

That I will give you the links to them at the end of the slide.

Next, the prerequisites for this session,

Beginner to intermediate knowledge in programming (especially in Delphi & Python) would help you a lot.

RAD Studio is installed, here is the link to download it if you are new to Delphi.

Python installed, of course.

Python4Delphi is installed, here is the link to the installation instructions.

And the installation of the following Python libraries:

Scrapy, Matplotlib, Fastai, scikit-learn, NetworkX, and Pandas.

A recommended practice in installing the required Python libraries is using conda install

or Anaconda Distribution, instead of the regular Python distribution.

Due to their active and mature community that supports Python for Data Science and Data Analytics,

it would save you from complicated conflicts between libraries when you install them.

The idea behind this project is to enhance the existing Python4Delphi Demos.

You can find it on this GitHub link.

You can run most of the known Python libraries inside Python4Delphi GUI,

as long as you successfully installed them without any conflict with the Python4Delphi.

We have lots of demos and blog posts on how to use the Python library inside Delphi

on blogs.embarcadero.com and pythongui.org.

But most of them are still implemented around the existing Demo 01,

which is not really sophisticated, as you can see here.

The Python script is still showing in the lower Memo,

and the output would be printed out on the upper Memo.

So, if you are planning to create a GUI that doesn’t show the Python script,

the basic Demo 01 might not make you satisfied.

This GUI doesn’t support image output, so it would be shown in the default Python image viewer, instead of shown on the GUI.

In this example, I run the textblob library inside the Python4Delphi GUI.

And for the new GUI, that I’ve created specifically for each case and Python libraries.

I’ve achieved the following, in this first iteration:

All Python code is hidden or set up and run at the back-end.

Image and table output is shown inside the GUI.

It supports interchangeable between Python version & distributions (regular Python vs Anaconda distributions)

to avoid complicated conflicts for some Python libs.

Okay, let’s begin the demo.

In Demo 01, we will demonstrate how to use Python’s Scrapy library inside Delphi, using Python4Delphi

or I will call it Scrapy4D for short, through this session.

In this first Demo session, we will need to walk through some details,

to make us easier in understand how the GUIs work,

and make us easier in developing a more advanced GUI for the next Demo 02 and Demo 03.

What is Scrapy?

Scrapy is a fast high-level web crawling and web scraping framework

used to crawl websites and extract structured data from their pages.

It can be used for a wide range of purposes:

From data mining to monitoring and automated testing.

The following is the list of components used in the Scrapy4D demo app,

which you can see more clearly in the RAD Studio.

You don’t need the “scrapy startproject myProject”,

as you usually do when you use Scrapy with Python,

instead, the default example code is already embedded in this GUI app.

You just need to click the “Execute” button to get started.

Let’s see what the structure looks like on the RAD Studio IDE.

Here is the structure of the GUI.

This GUI was created by modifying Python4Delphi Demo34,

which makes us possibly change the Python version in the runtime,

this will save you from the seemingly complicated dll issues.

To see where we call the embedded Python code,

Navigate to the [UnitScrapy4D.pas](https://github.com/MuhammadAzizulHakim/embarcaderoBlog-repo/blob/main/Article17%20-%20Scrapy4D%20Demo/UnitScrapy4D.pas),

inside the FormCreate procedure,

add the following line to load our basic [scrapyApp.py](https://github.com/MuhammadAzizulHakim/embarcaderoBlog-repo/blob/main/Article17%20-%20Scrapy4D%20Demo/scrapyApp.py).

Using Python Libraries from Delphi this way is actually much cleaner and elegant,

if compared with the previous Python4Delphi demos.

As we’ve talked about in the previous slide,

with the existing Python4Delphi demos, we copy-paste-

okay, let’s back to the previous slide.

With the existing Python4Delphi demos, we copy-paste and run the Python script on the front-end of the GUI, or inside the lower TMemo.

Or on some of the Demos, the Python scripts are embedded on the back-end,

but it is written hardcode inside the .pas files.

Theoretically, you can change this .py file with any Python script you want.

As long as you successfully install the library correctly

The Python scripts are already tweaked correctly to produce the output inside the Python4Delphi GUI, instead of other media.

And, the Python library or Python scripts suitable for your GUI design and purpose.

And let’s open the PyScripter IDE, to see the Python files that we will run using Python4Delphi GUI.

Here it is.

The scrapyApp.py,

demo02\_quoteScraper.py

demo03\_wikiScraper.py

and, spider.py to scrape all the “machine learning” query search results on Google Scholar.

Let's go back to our slide again.

And to make sure all the library works well,

set up the following paths to your Environment Variable, for Anaconda Python.

What does this Scrapy4D GUI demo do?

It will import the Scrapy library and run the basic example, by executing the “scrapyApp.py” on the backend.

Basically, you can change or update the scrapyApp.py into anything you want, as long as you import the Scrapy library correctly.

We will walk through:

Scraping quotes from quotes.toscrape.com

Scraping titles and URLs from multiple Wikipedia pages at once

Scraping all Google Scholar search results, for “machine learning” query search

And finally,

You can read more details and a step-by-step tutorial on this article.

And, let’s run it.

Before executing any program, we can choose the Python version and distribution here.

As we’ve talked about before in the recommended practice,

let’s choose Anaconda Distribution.

Click “Execute” to import the library, and it will automatically run the very basic Scrapy example inside the “scrapyApp.py”.

This basic example would do web crawling into example.com

and it would print out the whole process.

And let’s try the second example:

Load script, and choose

demo02\_quotesScraper.py

Open

By clicking the “Open” button, it would replace the embedded “scrapyApp.py” at the backend

with this demo02\_quotesScraper.py.

This script will scrape all the quotes from quotes.toscrape.com.

Let’s run it. Execute this.

And

We successfully scrape the quotes!

Let’s see it clearly, like this:

And again, it’s also logging the whole details of the scraping processes.

In the same way,

Let’s try the demo03\_wikiScraper.py to collect multiple titles and URLs from multiple Wikipedia pages at once.

Load the demo03\_wikiScraper.py at runtime by clicking the “Load script…” button, and then “Open” and “Execute”.

Here is the output:

We can see the title,

The URLs

Title, URLs.

For multiple Wikipedia pages.

And the logs of the whole details of the scraping processes.

If the previous examples still do not impress you, nor your boss or colleague, let’s try this advanced example, the last one:

Scraping Google Scholar search results, for all the “machine learning” query search results!

Okay

Here is the “machine learning” query search results on Google Scholar

that we want to scrape-we want to scrape all of this.

The original code for this example is credited to @geekan (https://github.com/geekan/), and all the contributors of the [scrapy-examples](https://github.com/geekan/scrapy-examples) project.

I modified the code to update some obsolete Python lines,

like some scripts that are still using Python 2, so I need to update it into Python 3.

Another modification is replacing the exception handling with the new convention, as the existing code still uses the old convention in writing the exception handling.

Installing all required libraries & dependencies for the scrapy-example project,

and setting the code to send the output to the Delphi GUI instead of command prompt

and to make it suitable to run inside of Python4Delphi GUI instead of regular command-line operations.

And the good news is, that I added all the working code to the Coding Boot Camp repository.

And let’s take a look, at how it works.

And oh, before we begin.

It seems that I’ve got banned by Google Scholar, because of I run the scraper too often,

so, the trick to bypass this is to turn on a VPN, before running our web scraper.

Okay, the VPN is on.

Run

Load script.

And then, navigate to  ../googlescholar

/googlescholar

/spiders/

open the spider.py file.

And execute

This program literally scrapes all the Google Scholar search results, for the “machine learning” query search.

The scraping outputs and processes look pretty interesting.

And it needs a while to complete.

Yup. It’s done!

It’s a lot of data here.

The structure of the outputs are:

authors,

citation-text,

citation-url,

description,

journal-year-src,

related-text,

related-type,

related-url,

title,

and url.

Let’s try to open one link here.

And it’s a valid link to the Machine Learning paper.

Pretty impressive right?

For productivity, you can even open multiple Scrapy4D Windows, and run different scrapers at once, in parallel, like this.

Okay.

So, What Next?

The following are the lists that I think would be nice to have for our next Web Scraping GUI app:

Save the output to JSON, instead of only printing it on TMemo

For some scenarios, we want to save the output into CSV instead of JSON

And show the output in TStringGrid, instead of TMemo

And add some NLP capabilities and analytics to the GUI

And feel free to give your suggestions!

You can give comments straight to the article, or try out the demo from our repository, for example.

Next is Demo 02.

In Demo 02, we will talk about the integration between Python4Delphi with

Matplotlib, fastai, scikit-learn, and NetworkX.

The idea of this second demo session is to start from a natural idea to make a more useful and advanced GUI,

that instead of only capable to print out the text output,

it can show the image output inside the GUI.

This GUI actually is much more advanced than the usual Python script that shows the image in the default matplotlib or standard Python plot viewer.

The first one is Matplotlib4D.

What is Matplotlib?

Matplotlib is a comprehensive Python library for creating static,

animated, and interactive visualizations.

Matplotlib produces publication-quality figures in a variety of formats and interactive environments across platforms.

The following is the list of components used in the Matplotlib4D demo app,

which we can see more clearly in the RAD Studio.

What differs this GUI from the Scrapy4D is,

we add TImage here, to present the plotting result.

Here is the structure of the GUI, seen inside the RAD Studio IDE.

We will show you two plotting examples in this demo.

First, Annotated heatmap

The context for this example is,

that we often want to show data that depends on two independent variables

or visualize the strength of correlations between two variables as a color-coded image plot.

This data visualization technique is often referred to as a heatmap.

If the data is categorical, this would be called a categorical heatmap.

This example shows how to create a heatmap with annotations, which helps us to read the quantity on top of the color-coded plot.

In this example, we will also print out the labels of the data inside the TMemo.

The second example is: The anatomy of a figure

Context: To get the most out of Matplotlib, you need to know all the anatomy of a figure or a plot. In this example, we will learn them all: All the available anatomy of a figure that is provided by Matplotlib.

If you need more examples, read them in my article here.

I provide 5 plotting examples in that article, to give you some insights into possible use cases for this GUI.

Like creating a survey app, creating subplots, visualizing trends and the tightness of the relations on the data, etc.

Let’s begin the Demo.

The default example is matplotlibApp.py

It will import the matplotlib library,

plot annotated heatmap, and print out all the labels of the data.

And produce the plot with the file name matplotlibPlot.jpg.

Let’s “Execute” it.

Click the “Show plot” button to show it.

And voila, it shows up!

Again, like the Scrapy example in the previous section.

This plot can be shown on GUI, by tweaking the matplotlib code, so it would save the plotting result as a file, and then we will load it into the Python4Delphi GUI,

instead of showing the plot inside the default matplotlib image viewer.

Let’s try the second example,

“Load script…”

Choose the demo05\_anatomyOfAFigure.py file

Open

Execute

And “Show plot”

It’s a pretty cool figure.

It gives labels to each part of the figure, making us learn about it very clearly.

The “Title”

X-axis label

Y-axis label

Grid

Upper line plot

Lower line plot

Markers

Major and minor tick, etc

Knowing the anatomy of the figure enables us to develop the best data visualizations.

The next library that we can implement with this second GUI structure is fastai

And we will call it Fastai4D.

What is fastai?

fastai is a deep learning library built on top of PyTorch,

one of the leading modern and flexible deep learning frameworks.

It has a goal to make the training of deep neural networks as easy as possible, and, at the same time, make it fast and accurate using modern best practices.

It provides practitioners with high-level components that can quickly and easily provide state-of-the-art results

and provides researchers with low-level components that can be mixed and matched to build new approaches.

This Fastai4D GUI has the same GUI structure as the previous Matplotlib4D.

It has TMemo to present the text output, from dataset labels to the details from the deep learning process.

And it has TImage to present the plotting outputs.

We will show you two operations in this demo.

First, loading the image datasets with their labels.

In this fastai embedded Python code, we are going to use the famous Oxford-IIIT Pet Dataset by O. M. Parkhi et al., 2012

which features 12 cat breeds and 25 dog breeds.

Our model will learn how to differentiate between these 37 distinct categories.

According to their paper, the best accuracy they could get in 2012 was 59.21%,

using a complex model that was specific to pet detection, with separate “Image”, “Head”, and “Body” models for the pet photos.

The second example is: Train deep learning model.

This example will train ResNet-34.

We will use a convolutional neural network backbone and a fully connected head with a single hidden layer as a classifier.

ResNet-34 is a 34-layer convolutional neural network that can be utilized as a state-of-the-art image classification model.

If you need more examples, read them in my article here.

I provide the end results and a detailed explanation of it.

Let’s begin the Demo.

The default example is fastaiApp.py

It will import the fastai library,

and load the image datasets with their labels.

The image output would be saved in a file, as fastaiImage.jpg.

We can click the “Show plot” button to show it.

Let’s “Execute” it.

Let’s see the interesting outputs.

It downloads and untars the image datasets.

Print out the path to all the images.

And print all the image annotation vocabularies.

Click “Show plot” to see how the image datasets look.

It shows up! Very interesting dataset.

Let’s try the second example, to try the deep learning model.

“Load script…”

Choose the demo02\_trainModel.py file

Open

Execute

This script is building a model which will take images as input and will output the predicted probability for each of the categories

(in this case, it will have 37 outputs),

and we will train for 4 epochs (4 cycles through all our data).

The details about the model are printed on TMemo.

So this is some valuable information.

Training the Resnet34 for all epochs through all our data might take a long time for a regular laptop.

So we won't s see it through the end, so you can go to our blog post,

on the link in the previous slide, if you are curious about the end results.

Actually, we still have 2 implementations on this Demo 02 GUI:

ScikitLearn4D and NetworkX4D.

But we have a limited duration here.

So maybe we will just fast forward into it, as the principles in running the GUI are basically the same.

Here is the ScikitLearn4D.

scikit-learn is an open-source Python machine learning library.

Prepared for this Demo, it has 10 different Unsupervised Machine Learning algorithms that are ready to execute.

And visualize each plot, produced by different Machine Learning algorithms.

And here is the NetworkX4D.

NetworkX is a Python package for creating, manipulating, and studying complex networks’ structure, dynamics, and functions.

Prepared for this Demo, it would plot graphs and networks

And the second example is to perform and plot degree analysis

As usual, I’ve provided you with the link to read more details.

Okay.

So, What Next?

The following are the lists that I think would be nice to have for our next Machine Learning and Data Visualization GUI app:

Save or record all the text output, instead of only printing it on TMemo

Add TStringGrid to present the structured data. It is nice if we can keep on track with the datasets. This update would be shown on Demo 03.

Call the datasets directly from the database, instead from local files.

Show multiple image outputs, maybe it is nice if we have some kind of tabs to see and change between the image outputs.

And feel free to give your suggestions!

You can give comments straight to the live chat, to the article, or try out the demo from our repository.

Okay, the last one, Demo 03

Pandas4Delphi

What is pandas?

pandas is a Python package that provides fast, flexible, and expressive data structures designed to work with structured

(tabular, multidimensional, potentially heterogeneous) and time-series data easily and intuitively.

pandas aim to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis and manipulation tool available in any language.

It is already on its way toward this goal.

The following is the list of components used in the Pandas4D demo app,

which we can see more clearly in the RAD Studio.

What differs this GUI from the previous Demo 02 is,

we add TStringGrid here, to present the dataset that we want to analyze,

and we add TEdit to show the path to the dataset.

Here is the structure of the GUI, seen inside the RAD Studio IDE.

What Pandas4D Demo do?

Import the pandas library and run the basic example, by executing the “pandasApp.py” on the backend

Show the dataset on TStringGrid

Perform 17 data analysis steps

And we will create 2 plots.

Histogram and scatterplot matrix.

Let’s begin the Demo.

Here is what the Python code looks like, inside the PyScripter IDE.

It has 17 data analysis operations here.

In this Demo version, if we want to plot the histogram, we need to comment on the scatterplot matrix manually.

And save.

Let’s open the RAD Studio IDE.

The default example is pandasApp.py

It will import the pandas library, and perform the 17 data analysis operations.

Let’s run it,

and load the dataset by clicking the “Load .csv file…”

Open the Churn\_Modelling.csv data.

And it loads perfectly.

Let’s see it by scrolling to the whole data.

What is really good about this is,

if you are an experienced Python developer that has experience in using Jupyter Notebook,

you might find that keep on track of your dataset is a bit challenging,

as if you show the dataset on the notebook, you cannot scroll them to take a look at the data.

And the data would block a huge space in the notebook that might not be good visually.

But, you won’t encounter such problems with this GUI.

And then, let’s “Execute”.

That’s a lot of useful data analysis operations here.

We might not dive deep into them one by one as it is outside the scope of this Demo,

But if you are curious about it, stay tuned to blogs.embarcadero.com, as I will publish the article about it there.

Let’s show the histogram plot for a selected query.

If you still want to look at the relations among the data,

by visualizing it using a scatterplot matrix,

let’s go back to the PyScripter IDE.

Uncomment the scatterplot matrix code chunk.

And put a comment to the histogram code chunk.

Save.

Go back to the RAD Studio.

Rerun the Pandas4D GUI.

Load the data.

Execute.

And “Show plot”.

And it’s done!

What Next?

The following are the lists that I think would be nice to have for our next Pandas Data Analysis GUI app:

Save/record all the text output, instead of only printing it on TMemo

Create more dynamic tables using TStringGrid to present the output, instead of TMemo. Because sometimes we want to alter the table and see it in the form of a table, instead of just some text or data frame.

Call the datasets directly from the database, instead of only from local files.

Show multiple image outputs, so we don't need to comment-uncomment script each time we want to present different plots.

And feel free to give your suggestions!

You can give comments straight to the live chat, to the article, or try out the demo from our repository.

For some scenarios, we want to save the output into CSV instead of JSON

And show the output in TStringGrid, instead of TMemo

And add some NLP capabilities and analytics to the GUI

And feel free to give your suggestions!

You can give comments straight to the article, or try out the demo from our repository, for example.

Further readings

Here is the list of useful references and links that you might need

if you want to try and reproduce what I’ve shown you in this section today.

Feel free to add and develop more advanced use cases,

using the existing Python4Delpi Demos as its basis.

That’s it for now.

I hope you enjoy this session and learn some useful things.

Thank you for your time and attention!

Goodbye!