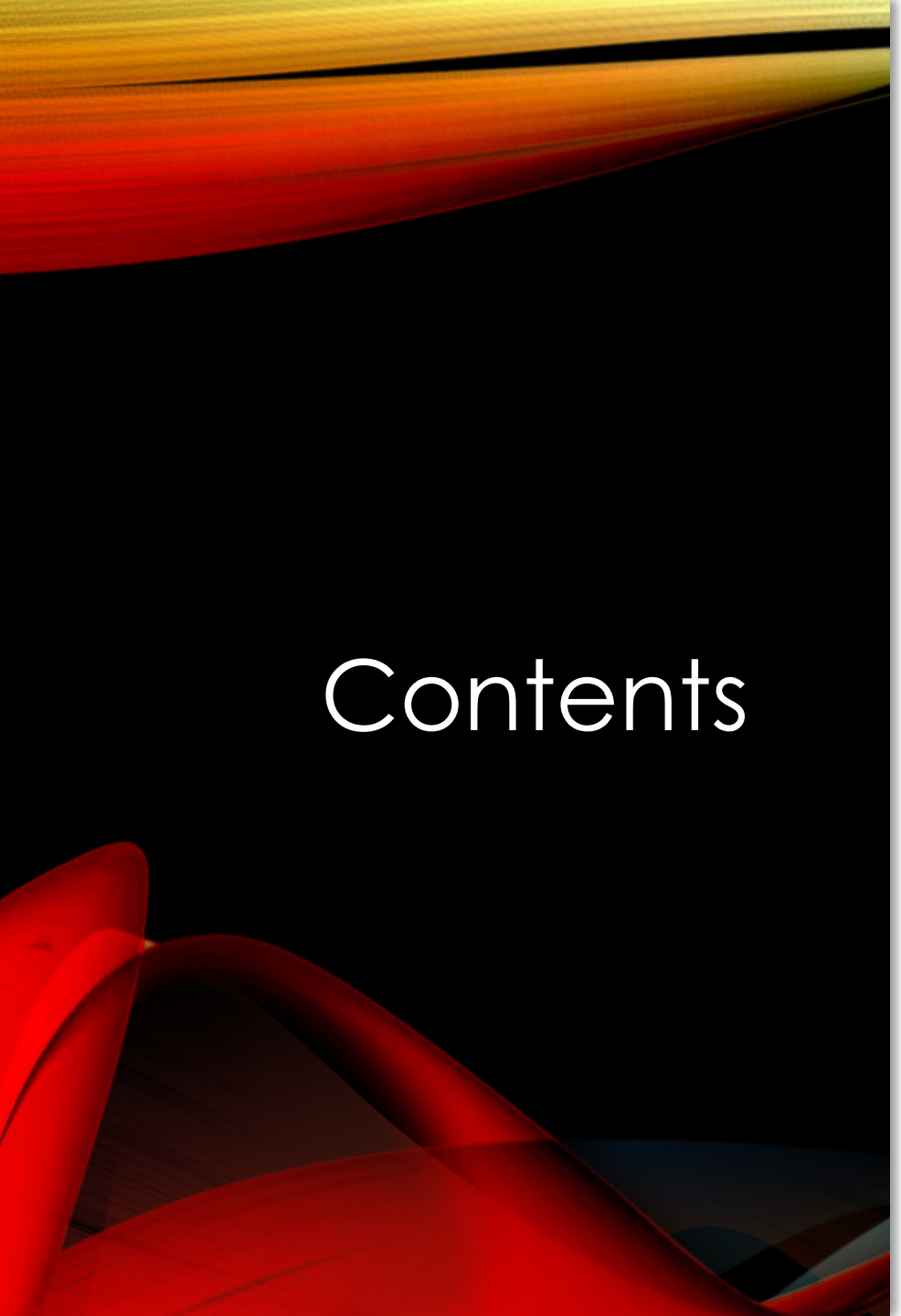
Python for Delphi Developers (Part II)

Webinar by Kiriakos Vlahos (aka PyScripter)
and Jim McKeeth (Embarcadero)



Contents

- Using python libraries and objects in Delphi code
  - VarPyth
- Python based data analytics in Delphi applications
  - Core libraries
  - Visualization
  - Machine Learning
- Running python scripts without blocking your GUI
  - TPythonThread
- Creating Python extension modules using Delphi
- Python GUI development using the VCL

VarPyth

- High-level access to python objects from Delphi code
- It also allows you to create common python objects (lists, tuples etc.)
- Uses custom variants
- No need to use the low-level python API or deal with python object reference counting
- Small performance penalty
- Example
  - `ShowMessage(SysModule.version)`

- Explore Demo25 and the VarPyth unit tests

VarPyth Demo

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   var np := Import('numpy');
   var np_array: Variant :=
     np.array(VarPythonCreate([1,2,3,4,5,6,7,8,9,10]));
   PythonModule.SetVar('np_array',
     ExtractPythonObjectFrom(np_array));
end;
```

```
Python code:

from delphi_module import np_array
print("type(np_array) = ", type(np_array))
print("len(np_array) = ", len(np_array))
print("np_array = ", np_array)

res_array = np_array.copy()
for i in range(len(np_array)):
    res_array[i] *= np_array[i]
print("res_array = ", res_array)
```

```
procedure TForm1.btnRunClick(Sender: TObject);
begin
  GetPythonEngine.ExecString(UTF8Encode(sePythonCode.Text));
  for var V in VarPyIterate(MainModule.res_array) do
    ListBox.Items.Add(V);
end;
```

```
Output:

type(np_array) =  <class 'numpy.ndarray'>
len(np_array) =  10
np_array =  [ 1  2  3  4  5  6  7  8  9 10]
res_array =  [  1   4   9  16  25  36  49  64  81 100]
```

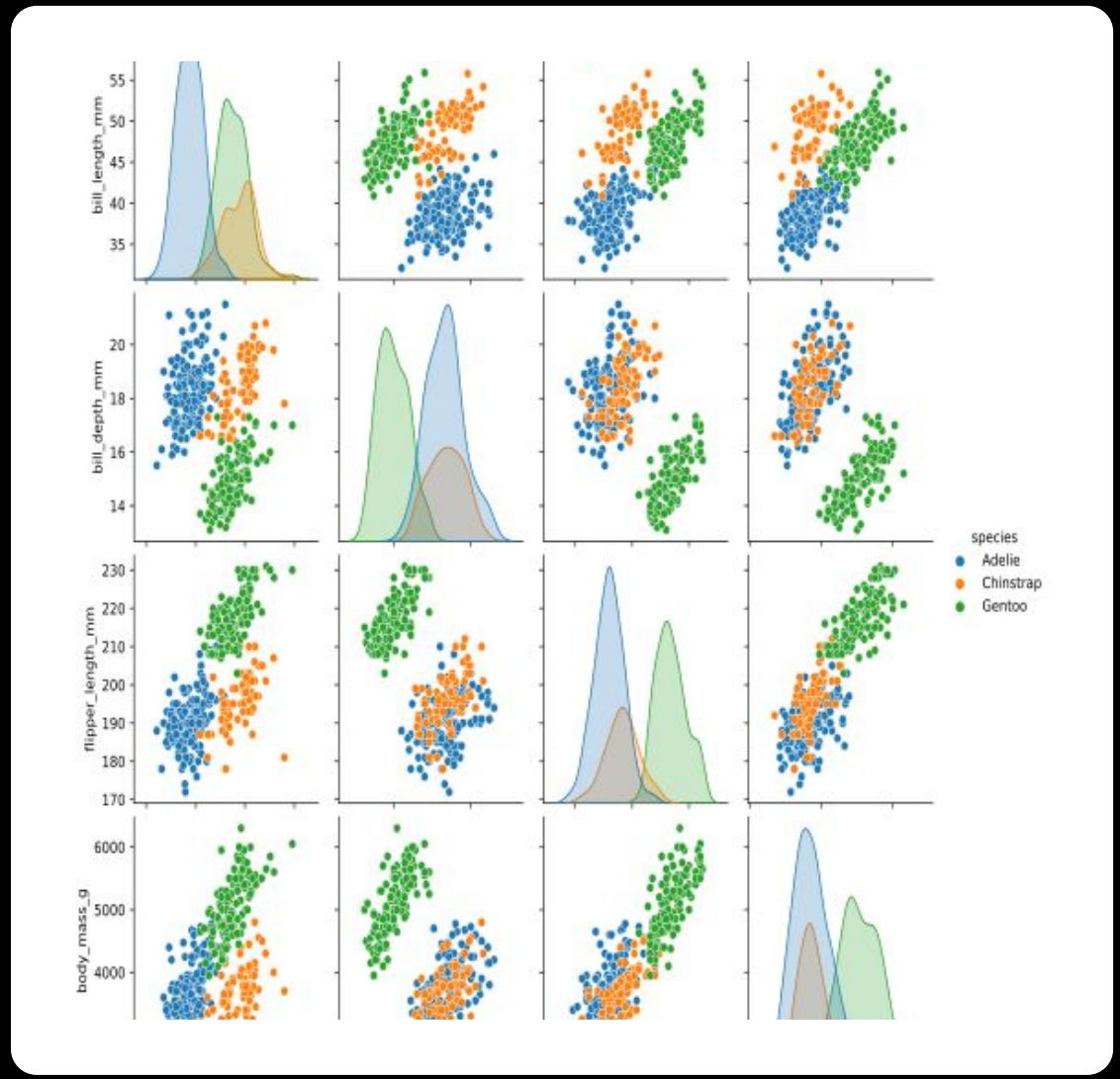
To learn more, explore Demo25 and the VarPyth unit tests

Core Python Libraries

- Core libraries
  - **numpy** – arrays and matrix algebra
  - **scipy** – math, science and engineering
  - **pandas** – data structure and analysis, R-like dataframes
- Data visualization
  - **matplotlib** – matlab-like plotting
  - **seaborn** – statistical data visualization
  - **mpld3** – turn matplotlib plots into interactive web pages
  - **bokeh** - interactive visualization library for modern browsers
  - **plotly** – interactive browser-based graphing library
  - **altair** – based on the visualization grammar vega-light

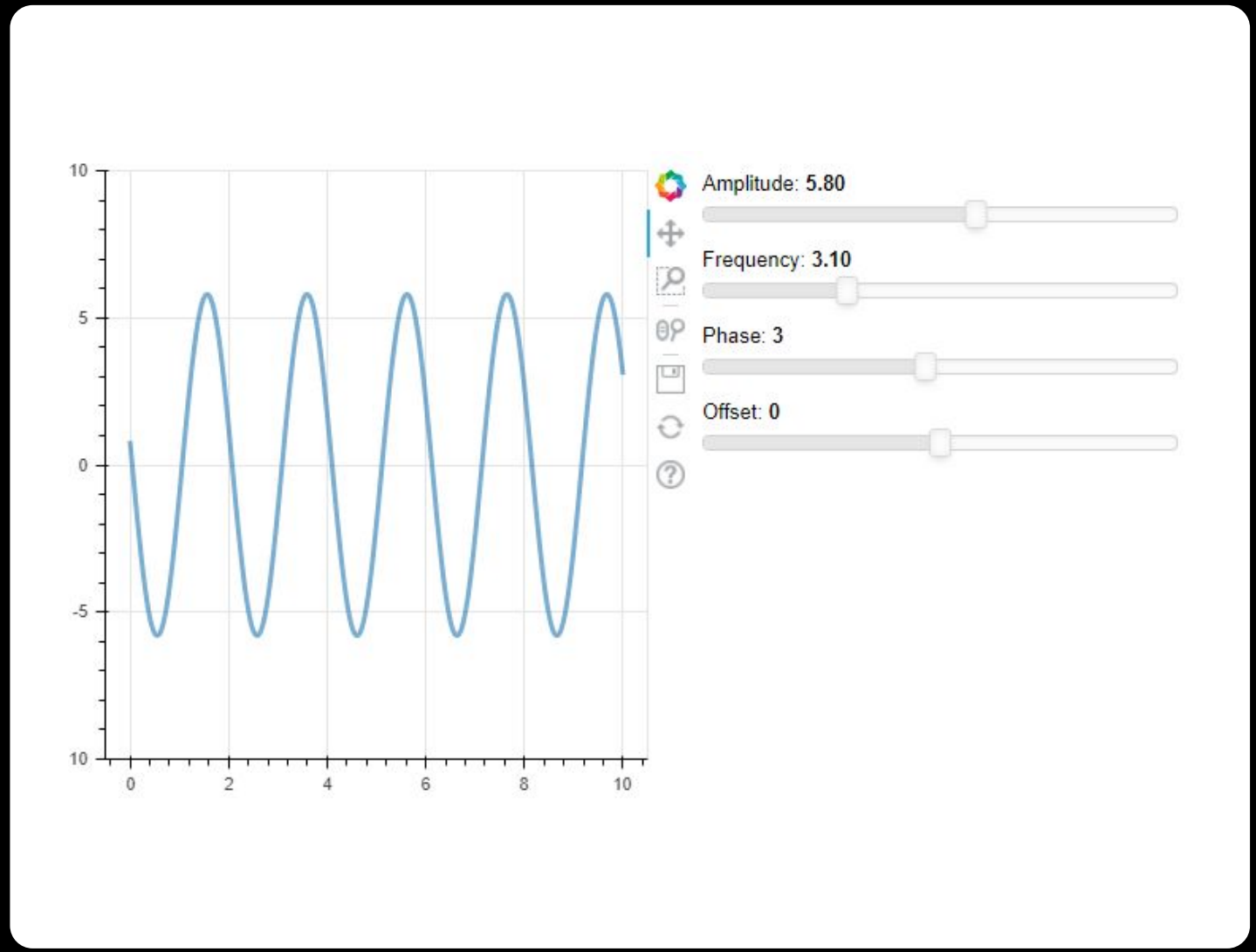
Data visualization Demos - pyVIZsvg

- Create charts with python libraries, save them in svg format and plot them in Delphi
- Uses **matplotlib** and **seaborn** python libraries
- Uses TSVGIconImage from EtheaDev SVGIconImageList components



Interactive Data Visualization Demo - PychartHtml

- Create interactive charts in python save them to html and show them inside Delphi applications
- Showcases the new **TEdgeBrowser**
- Uses the **matplotlib** with **mpld3**, **altair** and **bohek** python libraries

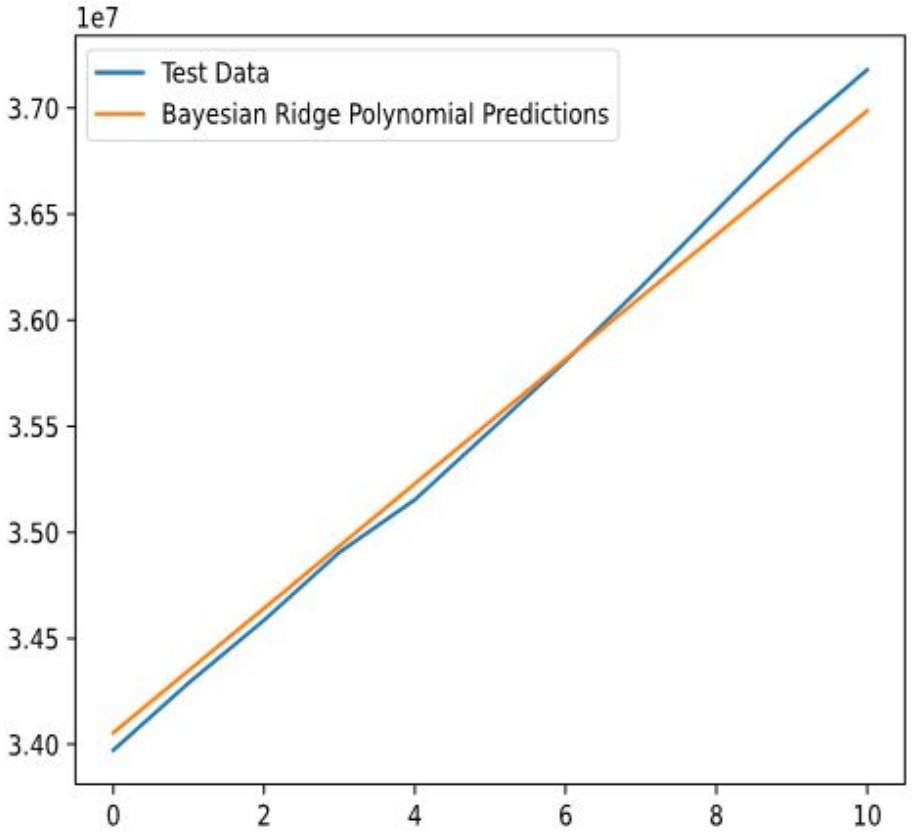


Machine Learning Python Libraries

- tensorflow (Google)
  - keras – high level pythonic interface
- PyTorch (Facebook)
- CNTK - The Microsoft Cognitive Toolkit (Microsoft)
- Spark MLlib and mxnet (Apache Foundation)
- scikit-learn
  - pure-python, general library, wide coverage, good choice for newcomers to ML/AI

Data Analytics Demo: COVID-19

- Demonstrates the combined use of **numpy**, **pandas**, **matplotlib** and **scikit-learn**
- Use pandas to download Covid-19 confirmed cases data from Web and aggregate it
- Use scikit-learn to
  - Split the data into training and test sets
  - Transform the data
  - Define model (Bayesian Ridge Regression)
  - Optimize model parameters
  - Generate predictions
- Use matplotlib to compare actual vrs fitted test data.



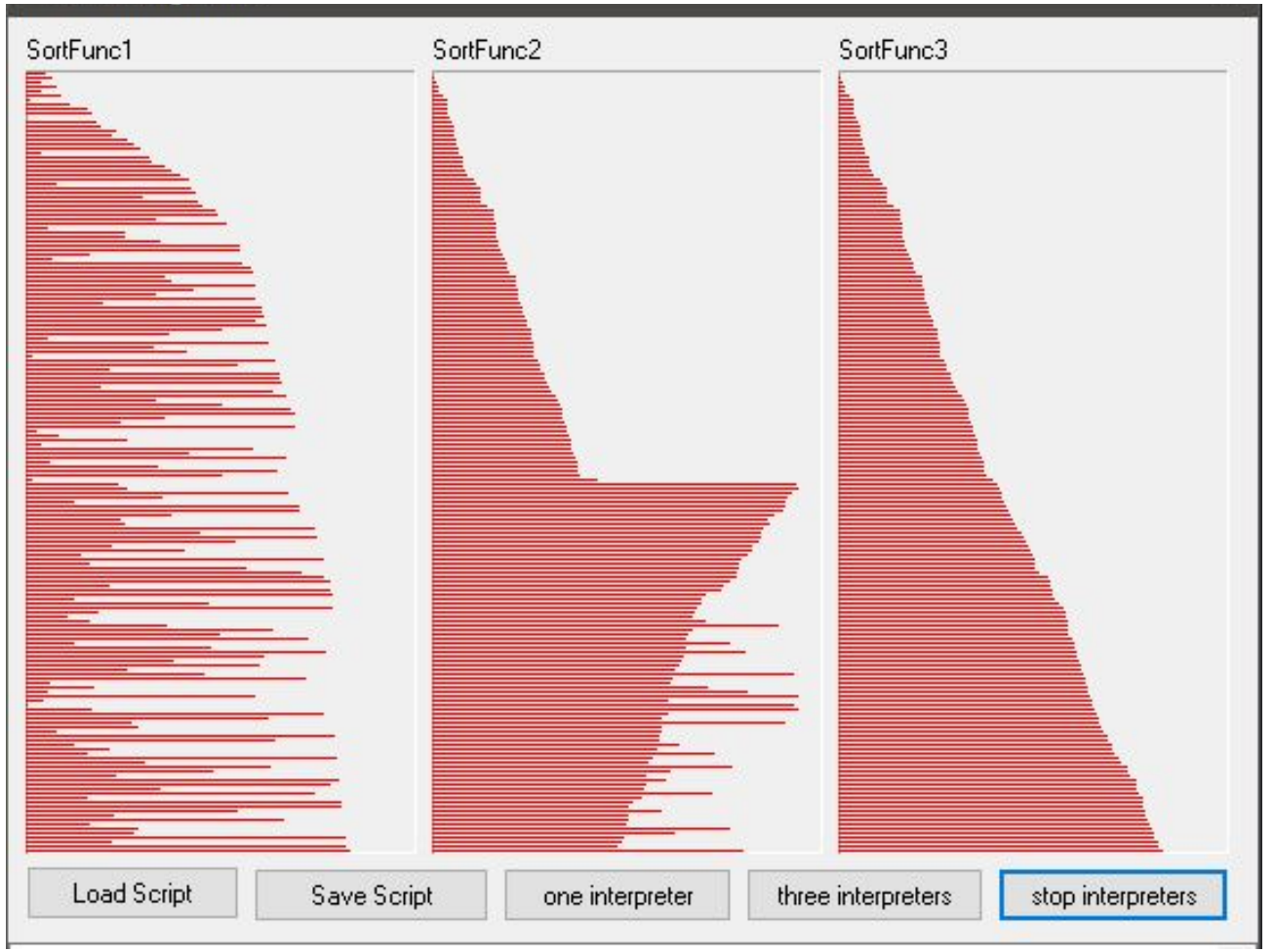
Running Python Scripts Without Blocking Your Main Thread

- Python uses the infamous Global Interpreter Lock (GIL)
- This means that only one thread can run python code at any given time
- Python switches between python created threads automatically
- Applications using python need to acquire the GIL before running python code and release it soon afterwards

- Welcome to **TPythonThread**
  - TThread descendent
  - Automatically acquires and releases the GIL

Python threads – demo33

- Shows you how to
  - use TPythonThread
  - use new sub-interpreters
  - synchronize with the main thread
  - interrupt running threads with a keyboard interrupt exception



Technical Aside I

- FPU Exception mask
  - Delphi's default FPU exception mask is different from most other Windows apps
  - Incompatible with python libraries written in C or C++
  - If you use numpy, scipy, tensorflow etc. you need to match the FPU mask they expect to operate with
  - PythonEngine.pas provides a function for doing that: MaskFPUExceptions
  - Call MaskFPUExceptions(True) before python is loaded
    - e.g. the initialization section of your main form

  - See the P4D Wiki page for details

Technical Aside II

- Working with different python distributions:
  - P4D makes a good effort to discover registered python distributions automatically
  - But sometimes you want to use python distributions which are not registered
  - One such case is when want to **deploy your application bundled with python**. Python.org offers such a minimal distribution for applications embedding python.
  - Additional considerations apply when using an Anaconda distribution

- Read the Finding Python P4D Wiki page carefully

P4D Python Extension Modules

- Python extension modules are dynamic link libraries that can be used by python in a way similar to modules developed in python.
- They have '`pyd`' extension.
- P4D makes it very easy to create extension modules that contain functions and types developed in Delphi.
- These extension modules can be used by Python, independently of Delphi, and can be packaged with setuptools and distributed through PyPi.

Python extension modules Demo

```
function PyInit_DemoModule: PPyObject;
//function exported by dll - initializes the module
begin
  try
    gEngine := TPythonEngine.Create(nil);
    gEngine.AutoFinalize := False;
    gEngine.UseLastKnownVersion := False;
    // Adapt to the desired python version
    gEngine.RegVersion := '3.8';
    gEngine.DllName := 'python38.dll';

    gModule := TPythonModule.Create(nil);
    gModule.Engine := gEngine;
    gModule.ModuleName := 'DemoModule';
    gModule.AddMethod('is_prime', delphi_is_prime,
     'is_prime(n) -> bool' );

    gEngine.LoadDll;
  except
  end;
  Result := gModule.Module;
end;
```

```
Python test module: test.py

from DemoModule import is_prime
from timeit import Timer

def count_primes(max_n):
    res = 0
    for i in range(2, max_n + 1):
        if is_prime(i):
            res += 1
    return res
```

```
Command line:

> C:\Python\Python38\python test.py
Number of primes between 0 and 1000000 = 78498
Elapsed time: 0.29756570000000004 secs
```

Wrapping VCL as a Python Extension Module I

- We can use the same approach to create an extension module wrapping VCL
- Uses the WrapDelphi and WrapDelphiVCL units
- The whole of VCL (almost) is wrapped in 40 lines of code
- The generated `Delphi.pyd` file is only 5Mbs
- It can be uses to create VCL-based GUIs in python without needing Delphi
- A similar approach could be used to wrap FMX and create a multi-platform python GUI library
- Unclear whether there are licensing restrictions in distributing such a file

Wrapping VCL as a Python Extension Module II

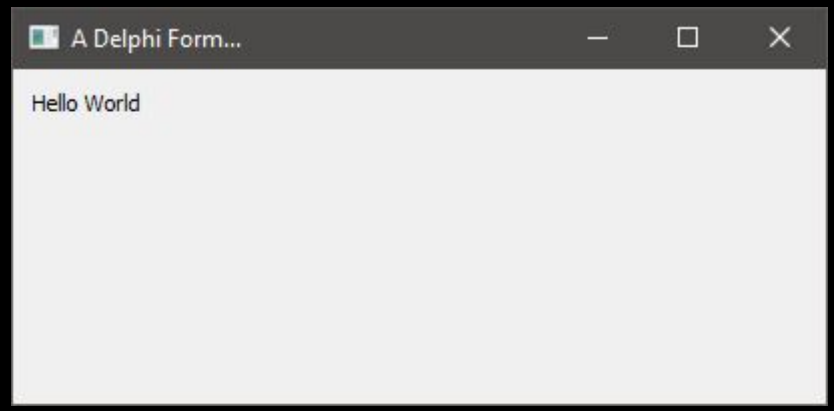
TestApp.py

```
from Delphi import *

class MainForm(Form):
    def __init__(self, Owner):
        self.Caption = "A Delphi Form..."
        self.SetBounds(10, 10, 500, 400)
        self.lblHello = Label(self)
        self.lblHello.SetProps(Parent=self, Caption="Hello World")
        self.lblHello.SetBounds(10, 10, 300, 24)
        self.OnClose = self.MainFormClose
    def MainFormClose(self, Sender, Action):
        Action.Value = caFree

def main():
    Application.Initialize()
    Application.Title = "MyDelphiApp"
    f = MainForm(Application)
    f.Show()
    FreeConsole()
    Application.Run()

main()
```

Command line:

```
> C:\Python\Python38\python .\TestApp.py
```

```
unit uMain;

interface

uses PythonEngine;

function PyInit_Delphi: PPyObject; cdecl;

implementation

uses WrapDelphi, WrapDelphiVCL;

// similar to the previous extension module
```



Summary

- With Python for Delphi you can get the best of both worlds
- P4D makes it very easy to integrate Python into Delphi applications in RAD way, thus providing access to a vast range of python libraries
- Expose Delphi function, objects, records and types to Python using low or high-level interfaces (WrapDelphi)
- Create/Access/Use Python objects/modules in your Delphi code using a high-level interface (VarPyth)
- Run python code in threads
- Create python extensions modules
- Wrap Vcl as a Python extension module to create GUIs with python

Resources

- Python4Delphi library
  - github.com/pyscripter/python4delphi
- PyScripter IDE
  - github.com/pyscripter/pyscripter
- Thinking in CS – Python3 Tutorial
  - openbookproject.net/thinkcs/python/english3e/
- PyScripter blog
  - pyscripter.blogspot.com/
- Webinar blog post
  - blogs.embarcadero.com/python-for-delphi-developers-webinar/