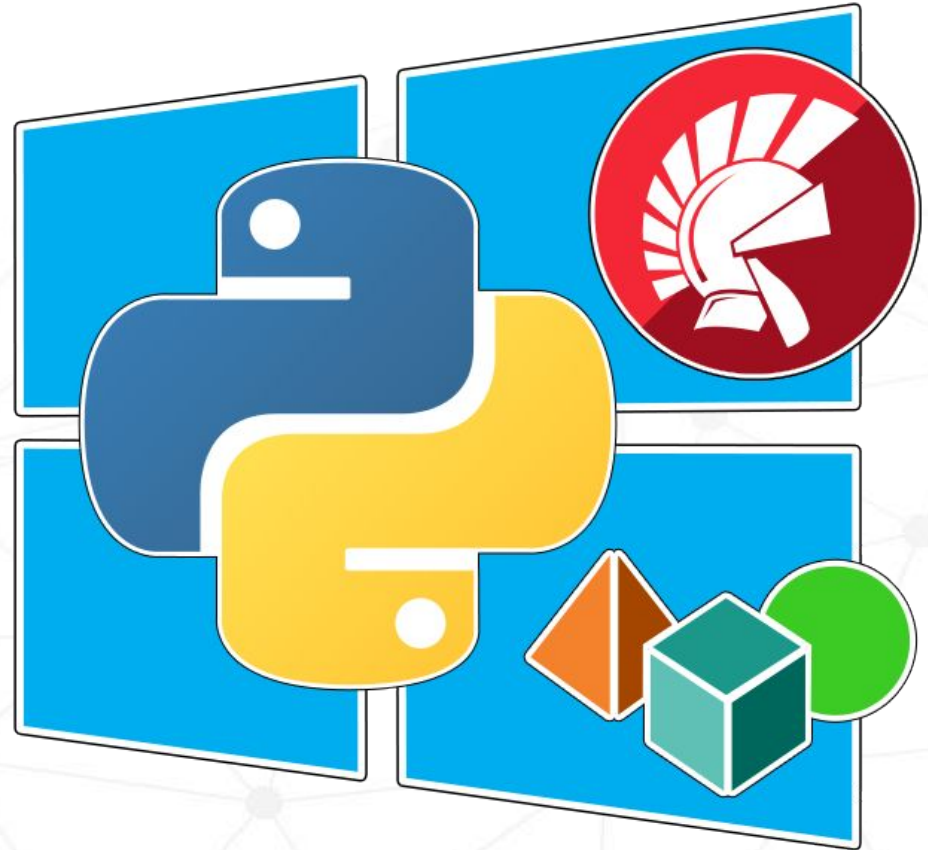


# Python Native Windows GUI with DelphiVCL

Anbarasan Anbazhagan  
Wed, Dec 23<sup>rd</sup>, 2020



# Abstract

Sometimes your application needs a user interface, but what is the best way to make one for Python applications? Enter DelphiVCL for Python. The VCL is a mature Windows native GUI framework with a huge library of included visual components and a robust collection of 3rd party components. It is the premier framework for native Windows applications, but how to use it with Python? Thanks to the DelphiVCL Python package (based on the open-source Python4Delphi library), the VCL is a first-class package for building native Windows GUIs with Python. Need more design tools? You can build the entire GUI in Delphi and then write all the logic in Python.

Join this free webinar to learn how to catapult your Python GUI game to new levels you never imagined possible. DelphiVCL is the fastest, most mature, and complete GUI library for native Windows Python GUI development.

# Agenda

- Why Native Windows GUI is Important
  - Performance, Consistent Behavior, Microsoft Active Accessibility (MSAA), & Handles
- Other common Python GUI frameworks
  - TkInter & PyQt: How they work and their drawbacks
- Introducing DelphiVCL for Python
  - Based on Python4Delphi and Embarcadero Delphi
  - Introduction to Delphi & VCL
  - Using the package directly from Python
  - Using the Delphi visual designers
  - Overview of VCL components
  - Using the DocWiki for documentation
- What about cross platform?
  - DelphiFMX for Python is *coming soon* for Windows, macOS, Linux, and Android

# Native Windows GUI

- **Performance:** Windows provides hardware acceleration for native controls.
- **Windows Handles:** Native Windows controls have Windows handles providing better OS and intra-application integration.
- **Consistent Behavior:** The use of native controls gives users consistent behavior between all applications they use
- Microsoft Active Accessibility (MSAA) is the framework that leverages native controls to provide accessibility interfaces like screen readers

Check this Simple Explanation By Marco Cantu about Desktop Native

[https://www.youtube.com/watch?v=dCE\\_oz9ZY4U](https://www.youtube.com/watch?v=dCE_oz9ZY4U)

# Other Python GUI Frameworks

## Most Common

- **Tkinter** (*TK Interface*) – Binding for the cross-platform **Tk** widget toolkit
- **PyQt** (*pie-cute*) – Binding for the cross-platform **Qt** widget library

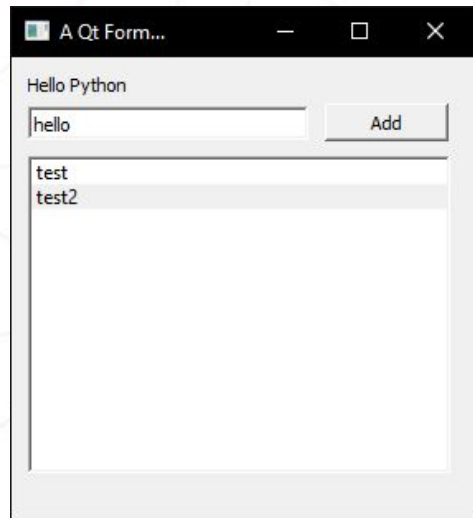
## Others

- **PySide** – Also uses Qt
- **wxPython** – Binding for **wxWidgets**
- **PyGTK** – Binding for **GTK** (GNOME)
- **Kivy** – Focused on mobile and cross-platform

*None provide native Window controls or Windows platform integration.*

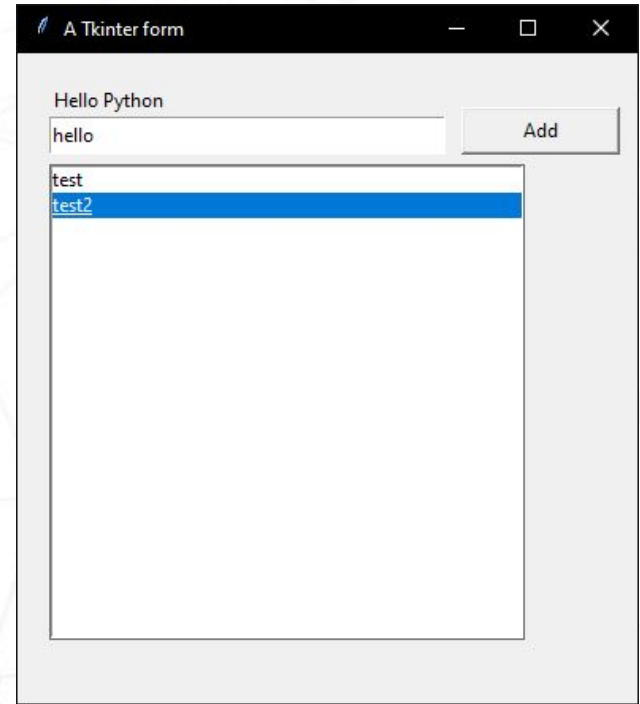
# PyQt and Qt

- Python binding for the Qt widget toolkit
- Originally by Trolltech (1991) and Nokia (2008)
- GPL/LGPL license, or Qt Commercial license
  - Open source or paid license
- Must be installed
  - `pip install PyQt5`
- Fusion style option comes close to matching platform look, but still doesn't use native controls



# Tkinter and the Tk GUI toolkit

- The default Python GUI
  - included with most Python installs
- Available under Python license
- Uses Tcl/Tk cross platform widgets
- Supports various styles
  - No OS style integration
- More information
  - [docs.python.org/3/library/tk.html](https://docs.python.org/3/library/tk.html)



# Tkinter Example

```
import tkinter as tki

class Application(tki.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.list = tki.Listbox(self)
        self.editb = tki.Button(self, text="Add", command=self.add_to_list)
        self.edit = tki.Entry(self)
        self.label = tki.Label(self, text="Hello Python", anchor='w')
        self.master = master
        self.place(anchor='nw', x=10, y=10, relwidth=1, relheight=1,
bordermode='outside')
        self.create_widgets()

    def create_widgets(self):
        self.label.place(x=10, y=10, width=100, height=20)
        self.edit.place(x=10, y=30, width=250, height=24)
        self.editb.place(x=270, y=24, width=100, height=30)
        self.list.place(x=10, y=60, width=300, height=300)

    def add_to_list(self):
        self.list.insert(tki.END, self.edit.get())

root = tki.Tk()
root.minsize(500, 400)
ttl = root.title("A Tkinter form")
app = Application(master=root)
app.mainloop()
```



# PyQt Example

```
import sys
from PyQt5.QtWidgets import *

class PyQtLayout(QWidget):

    def __init__(self):
        super().__init__()
        self.UI()

    def UI(self):
        self.setGeometry(10, 10, 280, 280)
        self.move(10, 10)
        self.setWindowTitle('A Qt Form...')
        label = QLabel('Hello Python', self)
        label.move(10, 10)
        edit = QLineEdit("", self)
        edit.move(10, 30)
        edit.resize(170, 20)
        button = QPushButton("Add", self)
        button.move(190, 28)
        listView = QListWidget(self)
        listView.move(10, 60)
        def buttonClick():
            listView.addItem(edit.text())
        button.clicked.connect(buttonClick)
        self.show()
```

```
def main():
    app = QApplication(sys.argv)
    #QApplication.setStyle(QStyleFactory.create('Windows'))
    ex = PyQtLayout()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

# DelphiVCL.py Example

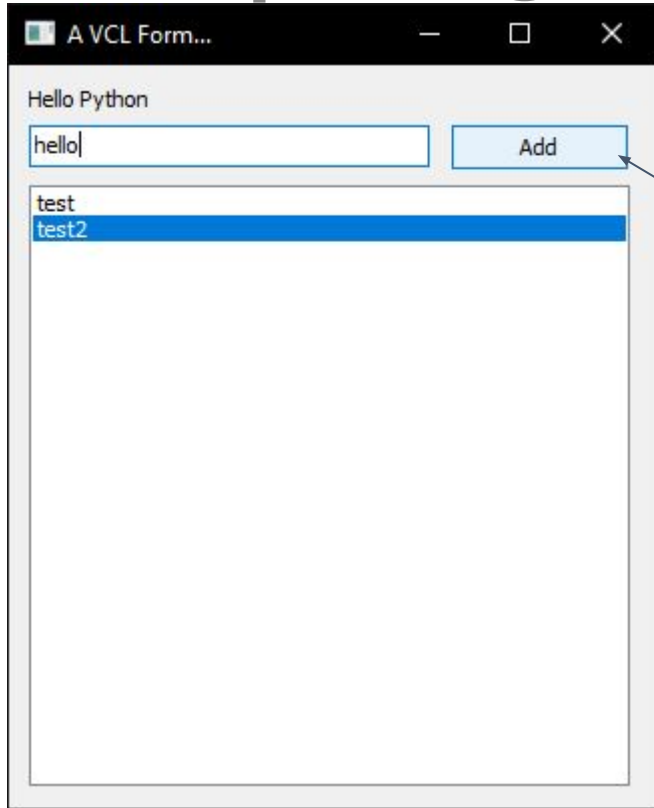
```
from DelphiVCL import *

class MainForm(Form):
    def __init__(self, Owner):
        self.Caption = "A VCL Form..."
        self.SetBounds(10, 10, 340, 410)
        self.lblHello = Label(self)
        self.lblHello.SetProps(Parent=self, Caption="Hello Python", Top=10, Left=10, Width=100, Height=24)
        self.edit1 = Edit(self)
        self.edit1.SetProps(Parent=self, Top=30, Left=10, Width=200, Height=24)
        self.button1 = Button(self)
        self.button1.SetProps(Parent=self, Caption="Add", OnClick=self.Button1Click)
        self.button1.SetBounds(220, 29, 90, 24)
        self.lb1 = ListBox(self)
        self.lb1.SetProps(Parent=self)
        self.lb1.SetBounds(10, 60, 300, 300)

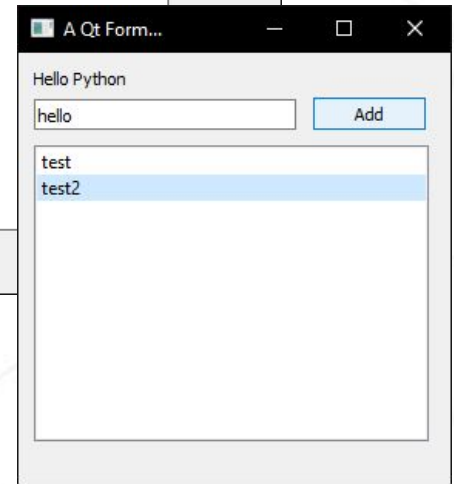
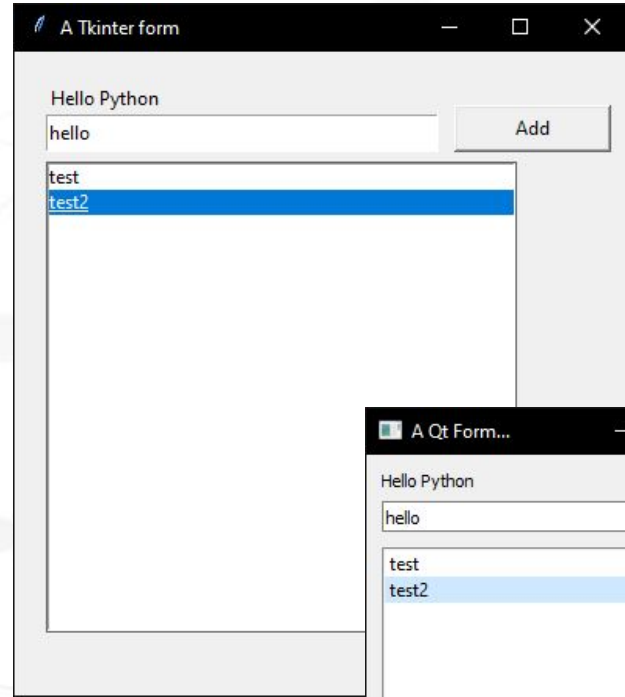
    def Button1Click(self, Sender):
        self.lb1.Items.Add(self.edit1.Text)

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

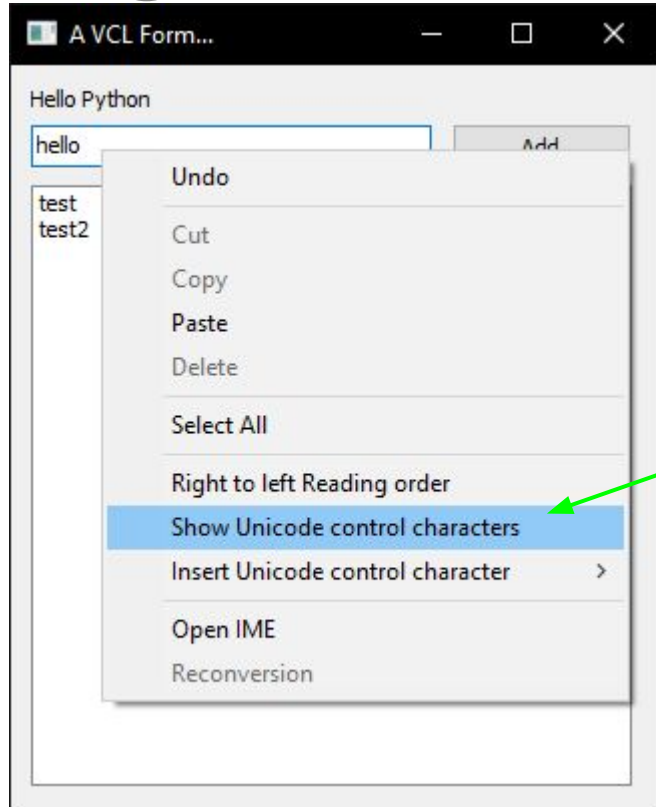
# Comparing the Forms



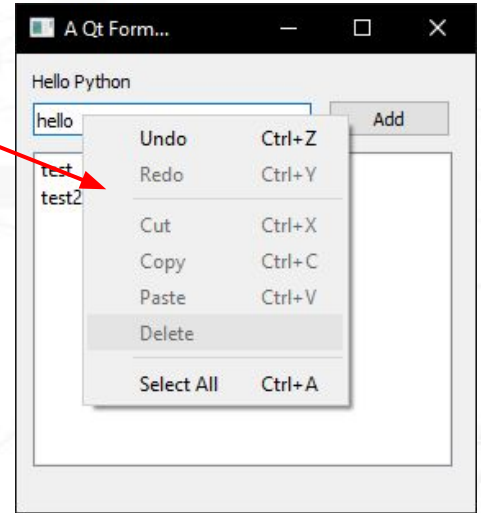
Windows 10 Styling



# Right Click in Edit Box



Operating system  
Extensions by default  
(fully customizable)



Missing OS  
integrations

Tkinter has no Right Click menu by default  
in the Entry box

# VCL

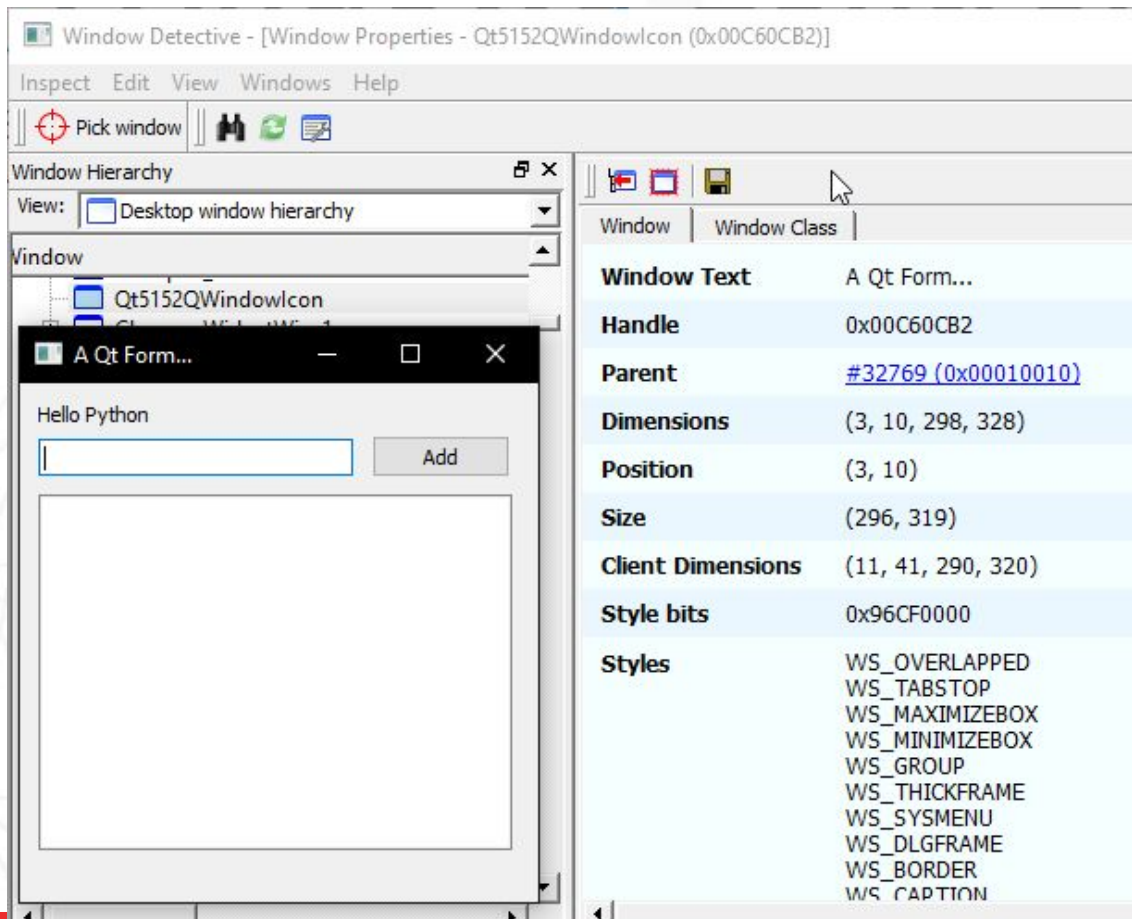
- Uses Windows controls
- Each control has a handle
- Support for UI testing
- Accessibility support

The screenshot displays the Window Detective application. The main window, titled 'A VCL Form...', contains a text input field with the text 'Hello Python' and an 'Add' button. The 'Window Hierarchy' pane on the left shows a tree structure with 'TForm' as the root, containing 'TListBox', 'TButton', and 'TEdit'. The 'Window Properties - TListBox (0x01B121AA)' pane on the right provides detailed information about the selected control.

Window	Window Class
<b>Window Text</b>	
Handle	0x01B121AA
Parent	<a href="#">TForm (0x010A236E)</a>
Dimensions	(205, 351, 504, 650)
Position	(205, 351)
Size	(300, 300)
Client Dimensions	(207, 353, 502, 648)
Relative Dimensions	(10, 60, 309, 359)
Style bits	0x54010141
Styles	WS_OVERLAPPED WS_TABSTOP WS_MAXIMIZEBOX WS_CLIPSIBLINGS WS_VISIBLE WS_CHILD
Extended Style bits	0x00000200
Extended Styles	WS_EX_LEFT WS_EX_LTRREADING

# PyQt

- Fusion style looks native
- No individual controls
- No OS integration
- No Windows handles



The screenshot shows the Window Detective application. The main window is titled "A Qt Form..." and contains the text "Hello Python", a text input field, and an "Add" button. The "Window Hierarchy" pane shows the window structure, with "Qt5152QWindowIcon" selected. The "Properties" pane on the right displays the following information:

Property	Value
Window Text	A Qt Form...
Handle	0x00C60CB2
Parent	#32769 (0x00010010)
Dimensions	(3, 10, 298, 328)
Position	(3, 10)
Size	(296, 319)
Client Dimensions	(11, 41, 290, 320)
Style bits	0x96CF0000
Styles	WS_OVERLAPPED WS_TABSTOP WS_MAXIMIZEBOX WS_MINIMIZEBOX WS_GROUP WS_THICKFRAME WS_SYSMENU WS_DLGFRAME WS_BORDER WS_CAPTION

# Delphi & VCL

- Object Pascal(Delphi) is an extremely powerful language based upon core foundations such as a good program structure and extensible data types. These foundations are partially derived from the traditional Pascal language, but even the core language features have seen many extensions from the early days.

[Download a copy of the Object Pascal Handbook by Marco Cantu...FREE!](#)

- Windows-specific VCL (Visual Component Library) is a set of components and classes for the rapid development of Windows applications in the Delphi and C++ languages.
- The GUI components of the VCL are native Windows GUI components.

# Overview of VCL Components

- Visual Components

Visual components, such as TForm and TSpeedButton, are called controls and descend from TControl. Controls are used in GUI applications, and appear to the user at run time.

- Non Visual components

Non Visual components, such as TDataSource. Allows you to manipulate their properties and events just as you would a visual control at design time.

- Utility classes

Classes that are not components (that is, classes that descend from TObject but not TComponent) are also used for a variety of tasks. Typically, these classes are used for accessing system objects (such as a file or the clipboard) or for transient tasks (such as storing data in a list).



# VCL Visual Designers

- The **Form Designer** (or Designer) is displayed automatically when you are creating or editing a graphical application that uses a form file, either a .dfm or an .fmx file.
- A VCL form displays the standard Windows buttons for Minimize, Resize, and Close commands. Drag and Drop components from **ToolPallette** and you can view and set properties and events visually using **ObjectInspector**.
- Ultimate Design Guidelines Support with VCL Components.
- Taking a Snapshot of Your Form (VCL Only)
- **LiveBindings** is a data-binding feature supported by both the VCL and FireMonkey frameworks. It is an **expression-based** framework, which means it uses bindings expressions to bind objects to other objects or to dataset fields.
  - Create new LiveBindings between various visual components you have on your form.
  - Edit the existing LiveBindings.
  - Visualize all the LiveBindings that you have created.
  - Export your binding diagram as an image file.

# Python4Delphi & Embarcadero Delphi

- **Python for Delphi (P4D)** is a set of free components that wrap up the Python DLL into Delphi.
- Easily execute Python scripts, create new Python modules and new Python types.
- Can create Python extensions as DLLs and provides different levels of functionality:
  - Low-level access to the python API
  - High-level bi-directional interaction with Python
  - Access to Python objects using Delphi custom variants (VarPyth.pas)
  - Wrapping of Delphi objects for use in python scripts using RTTI (WrapDelphi.pas)
  - Creating python extension modules with Delphi classes and functions
- Design Beautiful Desktop and Mobile App UIs with **Embarcadero Delphi**
- Code Faster and Smarter with Delphi in built and Rich 3rd party Ecosystem.
- Debug Faster with Integrated Native Debugging
- Compile and Deploy High-Performance Native Apps
- Easily Improve the Quality of Your Code, Collaborate and Extend the IDE

# Using DelphiVCL For Python

- DelphiVCL for Python is nothing but wrapping of Delphi objects for use in python scripts using RTTI (WrapDelphi.pas) explicitly for VCL Components(WrapDelphiVCL.pas)
- WrapDelphiVCL.pas uses following unit files.
  - WrapDelphiTypes, WrapDelphiClasses,
  - WrapDelphiWindows, WrapDelphiControls,
  - WrapDelphiGraphics, WrapDelphiForms,
  - WrapDelphiActnList, WrapDelphiStdCtrls,
  - WrapDelphiComCtrls, WrapDelphiExtCtrls,
  - WrapDelphiButtons, WrapDelphiGrids;
- Wrappers helps to create and access Delphi Objects quickly from python.
- These wrappers contains container classes to extend and expose your custom events,methods,variables which can used in python script.

# Using DelphiVCL For Python, Steps:

- Extend this two units (WrapDelphi.pas) and (WrapDelphiVCL.pas) as a Python Extension Module e.g) **Delphi4Python.pyd**
- Create PythonEngine, PythonModule, PyDelphiWrapper and expose the function as PyInit\_<ExtensionDllName>
- Import the module in a Python script e.g) **from Delphi4Python import \***
- Use the Python variables, get/set methods which is exposed in WrapDelphiVCL classes like Form,Application,Button,PageControl,Edit,CheckBox etc to create Python Objects
- Set the properties of Python Objects using **SetProps**(propertyname = propertyvalue,...N) or each property one by one. Define events and custom methods required to manipulate the GUI events and operations in the script.
- Initialize the Application, show the GUI app. Save the PythonScript and Run the script from command prompt. e.g) D:\DelphiVclWebinar>C:\Python\Python39\Python.exe Sample.py

# DocWiki for Documentation

Delphi VCL DockWiki's

[http://docwiki.embarcadero.com/RADStudio/en/Main\\_Page](http://docwiki.embarcadero.com/RADStudio/en/Main_Page)

<http://docwiki.embarcadero.com/RADStudio/en/VCL>

[http://docwiki.embarcadero.com/RADStudio/en/Form\\_Designer](http://docwiki.embarcadero.com/RADStudio/en/Form_Designer)

[http://docwiki.embarcadero.com/RADStudio/en/How\\_To\\_Build\\_VCL\\_Forms\\_Applications](http://docwiki.embarcadero.com/RADStudio/en/How_To_Build_VCL_Forms_Applications)

[http://docwiki.embarcadero.com/RADStudio/en/VCL\\_Styles\\_Overview](http://docwiki.embarcadero.com/RADStudio/en/VCL_Styles_Overview)

Other Resources: Python4Delphi

<https://github.com/pyscripter/python4delphi>

Blog Resources:

<https://blogs.embarcadero.com/?s=python4delphi>

Youtube Embarcadero channel link:

<https://www.youtube.com/channel/UCMmsCQhkz-WIJ-IVBzPhbgA>

Python4Delphi

<https://www.youtube.com/watch?v=aCz5h96ObUM> (Webinar 1)

<https://www.youtube.com/watch?v=ssIKb3nJw5c> (Webinar 2)

<https://www.youtube.com/watch?v=hjY6lBgrHhM> (Getting started with Python4Delphi)

# Summary

- Learning Delphi and Building Native GUI apps is Easier and faster.
- Quite lot of resources with Rich third party ecosystem is available to get started.
- Python has vast packages to ease developers from basic programming task to advanced Scientific,ML,AI,Deep learning related tasks.
- Combining the Strengths of Delphi and Python helps users to build Ultra fast Native GUI Apps.

# Q & A

- Anbarasan : [naanbu.dreamy@gmail.com](mailto:naanbu.dreamy@gmail.com)