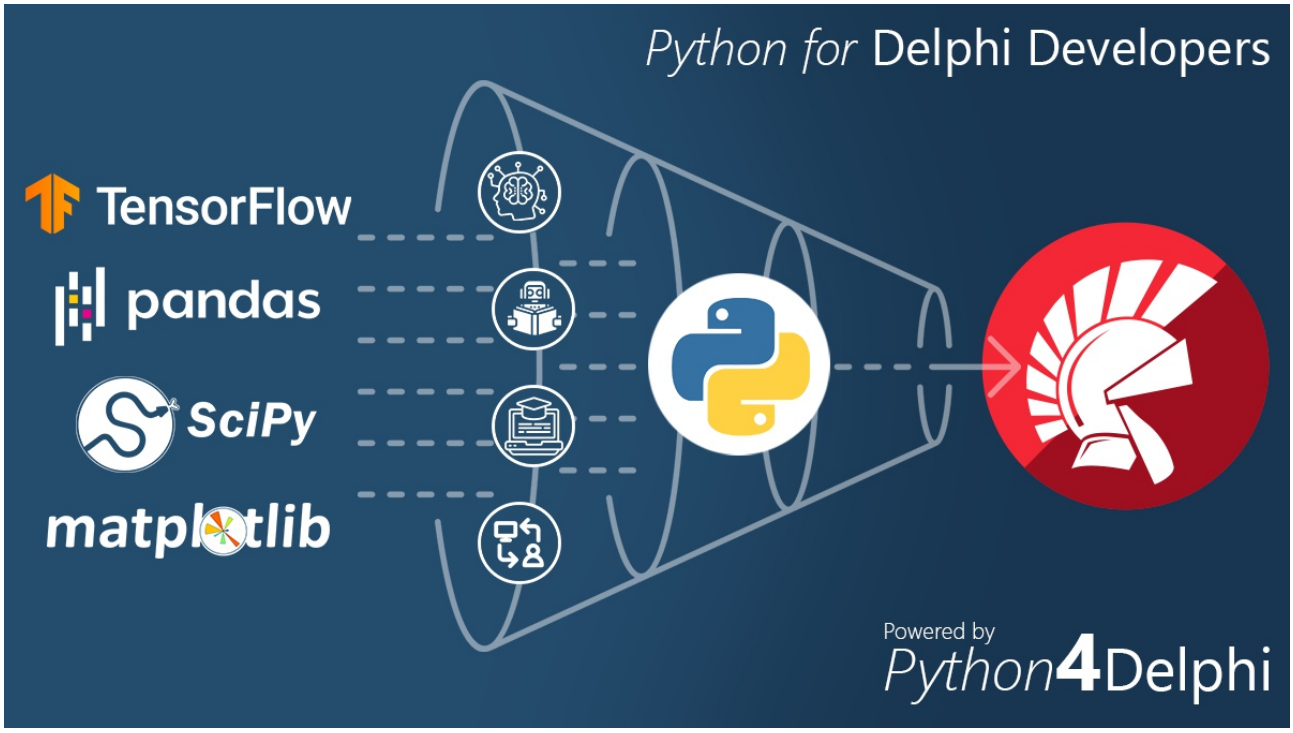
# Python4Delphi

Python für Delphi (P4D) ist ein Set kostenloser Komponenten, welche die Python-DLL in Delphi und Lazarus (FPC) einschließen. Eine DLL könnte beispielsweise die *python37.dll* sein. Sie ermöglicht es, Python-Skripte ziemlich einfach auszuführen, auch neue Python-Module oder neue Python-Typen zu erstellen. Sie können Python-Erweiterungen als DLLs erstellen und einiges mehr wie Skripte ausführen oder die Konsole zu automatisieren. P4D bietet verschiedene Funktionsebenen:

- Low-Level-Zugriff auf die Python-API
- Bidirektionale Interaktion auf hoher Ebene mit Python
- Zugriff auf Python-Objekte mit benutzerdefinierten Delphi-Varianten (VarPyth.pas)
- Wrapping von Delphi-Klassen zur Verwendung in Python-Skripten mit RTTI (WrapDelphi.pas)
- Erstellen von Python-Erweiterungsmodulen mit Delphi-Klassen, -Records und -Funktionen
- Generieren von Skripten in maXbox[1] aus einer Python-Engine.

P4D macht es nach einigen (aufwendigen) Studien sehr einfach, Python als Skriptsprache für Delphi-Anwendungen zu verwenden. Das Framework kommt auch mit einer umfangreichen Palette von Demos und nützlichen (Video-)Tutorials daher.




1 Scripting Engine für Pascal und Delphi-Anwendungen

Ein allererster einfacher Ansatz besteht darin, die Python-DLL direkt ohne Wrapper oder Mapper aufzurufen, z.B. die Copyright-Funktion:

```
//if fileExiststst(PYDLLPATH+ 'python37.dll';
  function getCopyRight: PChar;
     external 'Py_GetCopyright@C:\maXbox\EKON25\python37.dll stdcall';
```

Dann der Aufruf mit einer Test-Bedingung:

```
function IsDLLOnSystem(DLLName:string): Boolean;
var ret: integer;
    good: boolean;
begin
  ret:= LoadLibrary(pchar(DLLNAME));
  Good:= ret>0;
  if good then FreeLibrary(ret);
  result:= Good;
end;


if isDLLOnSystem(PYDLLPATH+PYDLLNAME) then begin
  showmessage('Py dll available');
  writeln(getCopyRight)
end;
```

```
Copyright (c) 2001-2019 Python Software Foundation.
All Rights Reserved.
Copyright (c) 2000 BeOpen.com.
All Rights Reserved.
Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.
Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.
```

Wir verwenden auch Objekte, um ein Python-Skript als Einbettungs-Konstante aufzurufen, mit der DLL-Funktion von PyRun_SimpleString:

```
procedure InitSysPath;
  var _path: PPyObject;
  const Script =
    'import sys' + sLineBreak+
    'sys.executable = r"%s"' + sLineBreak+
    'path = sys.path' + sLineBreak+
    'for i in range(len(path)-1, -1, -1):' + sLineBreak+
    '    if path[i].find("site-packages") > 0:' + sLineBreak+
    '        path.pop(i)' + sLineBreak+
    'import site' + sLineBreak+
    'site.main()' + sLineBreak+
    'del sys, path, i, site';
  begin
     if VenvPythonExe <> '' then
       ExecString(AnsiString(Format(Script, [VenvPythonExe])));
    _path := PySys_GetObject('path');
    if Assigned(FOnSysPathInit) then
      FOnSysPathInit(Self, _path);
  end;
```

## So verwendet man Python4Delphi

Der beste Weg, um mehr über die Verwendung von P4D zu erfahren, besteht darin, das umfangreiche Angebot an verfügbaren Demos auszuprobieren. Auch das Studium der Unit-Tests für VarPyth und WrapDelphi kann helfen zu verstehen, was mit diesen beiden Units und den wichtigsten PythonEngine.pas möglich ist.

Beginnen wir mit der Unit VarPyth.pas. Auf diese Weise lassen sich Python-Objekte wie COM-Automatisierungsobjekte in Ihrem Delphi-Quellcode verwenden. Dies ist ein Ersatz für die frühere *PythonAtom.pas*, welche die neuen benutzerdefinierten Variantentypen verwenden, die seit Delphi6 eingeführt wurden.

Sie können diese Python-Varianten in Ausdrücken genauso verwenden wie alle anderen Varianten oder Automatisierungen, z.B.:

```
var
 a: Integer; v: Variant;
begin
 v:= VarPythonEval('2 ** 3');
 a:= v;
```

Die Funktionen dieser Unit *VarPyth.pas* sind weitgehend dokumentiert und selbsterklärend.

## Was ist mit der WrapDelphi-Unit möglich?

P4D hilft auch, um Python-Erweiterungsmodule zu erstellen, die Ihre eigenen Klassen und Funktionen für den Python-Interpreter bereitstellen. Sie können Ihre Erweiterung mit Setuptools verpacken und über **PyPi** verteilen. Wenn Sie also ein vorhandenes Objekt oder eine vorhandene Unit in Delphi haben, die Sie in Python verwenden möchten, dieses Objekt jedoch nicht modifizieren wollen, um es Python-fähig zu machen, kann es als Python-Package existieren.

Mit *TPyDelphiObject* können Sie jedes Delphi-Objekt umschließen, das veröffentlichte Eigenschaften und Methoden verfügbar macht. Beachten Sie, dass die Bedingungsdefinitionen TYPEINFO und METHODINFO aktiviert sein müssen.
Wenn Sie beispielsweise eine vorhandene Delphi-Klasse haben, die TRGBColor:

```
 TRGBColor = class
  private
    fRed, fGreen, fBlue: Integer;
  public
    property Red: read fRed write fRed;
    property Green: read fGreen write fGreen;
    property Blue: read fBlue write fBlue;
  end;
```

Sie möchten diese Farbklasse nun in Python-Code verwenden, aber nichts in der Klasse selbst ändern. Dann erstellt man einen von *TPyObject* geerbten Wrapper, der einige sehr grundlegende Dienste und Tools bereitstellt, z. B. Abrufen und Festlegen von Attributen einer Farbe oder das Ausführen einer Zeichenfolgedarstellung:

```
TPyColor = class(TPyObject)
 private
    fColor: TRGBColor;
 public
    constructor Create( APythonType: TPythonType ); override;
    // Py Basic services
    function  GetAttr(key: PChar): PPyObject; override;
    function  SetAttr(key: PChar; value:PPyObject): Integer; override;
    function  Repr: PPyObject; override;
 end;
```

Das Projekt im Unterverzeichnis Delphi generiert ein Python-Erweiterungsmodul (eine DLL mit der Erweiterung „pyd" in Windows), mit dem Sie Benutzeroberflächen mit Delphi aus Python heraus erstellen können. Die ganze VCL (ja fast) ist in ein paar Zeilen Code verpackt! Die kleine Demo *TestApp.py* gibt Ihnen einen Vorgeschmack auf das, was möglich ist. Eine Maschinerie, mit der dies erreicht wird, ist die WrapDelphi-Unit.

Das Unterverzeichnis *DemoModule* zeigt, wie Sie Python-Erweiterungsmodule mit Delphi erstellen, die es Ihnen ermöglichen, in Delphi definierte Funktionen in Python zu verwenden. Kompilieren Sie das Projekt und führen Sie *test.py* über die Eingabeaufforderung aus (z.B. *py test.py*). Die generierte pyd-Datei sollte sich im selben Verzeichnis wie die Python-Datei befinden.
Dieses Projekt sollte leicht auf die Verwendung mit Lazarus und FPC angepasst werden. Das Unterverzeichnis *RttiModule* enthält ein Projekt, das dasselbe wie *DemoModule* ermöglicht, jedoch erweiterte RTTI (RunTimeTypeInformation) verwendet, um Delphi-Funktionen zu exportieren. Dies wird derzeit von FPC nicht unterstützt.

Die Unit *PythonEngine.pas* ist die zentrale Core-Unit des Frameworks. Sie sind dafür verantwortlich, eine und nur eine *TPythonEngine* zu erstellen. Der Großteil der Python/C-API wird als Memberfunktionen der Engine präsentiert.

```
type
  TPythonVersionProp = record
    DllName     : string;
    RegVersion  : string;
    APIVersion  : Integer;
  end;

  ...
  Py_BuildValue             := Import('Py_BuildValue');
  Py_Initialize             := Import('Py_Initialize');
```

```
  PyModule_GetDict           := Import('PyModule_GetDict');
  PyObject_Str               := Import('PyObject_Str');
  PyRun_String               := Import('PyRun_String');
  PyRun_SimpleString         := Import('PyRun_SimpleString');
  PyDict_GetItemString       := Import('PyDict_GetItemString');
  PySys_SetArgv              := Import('PySys_SetArgv');
  Py_Exit                    := Import('Py_Exit');
  ...
```

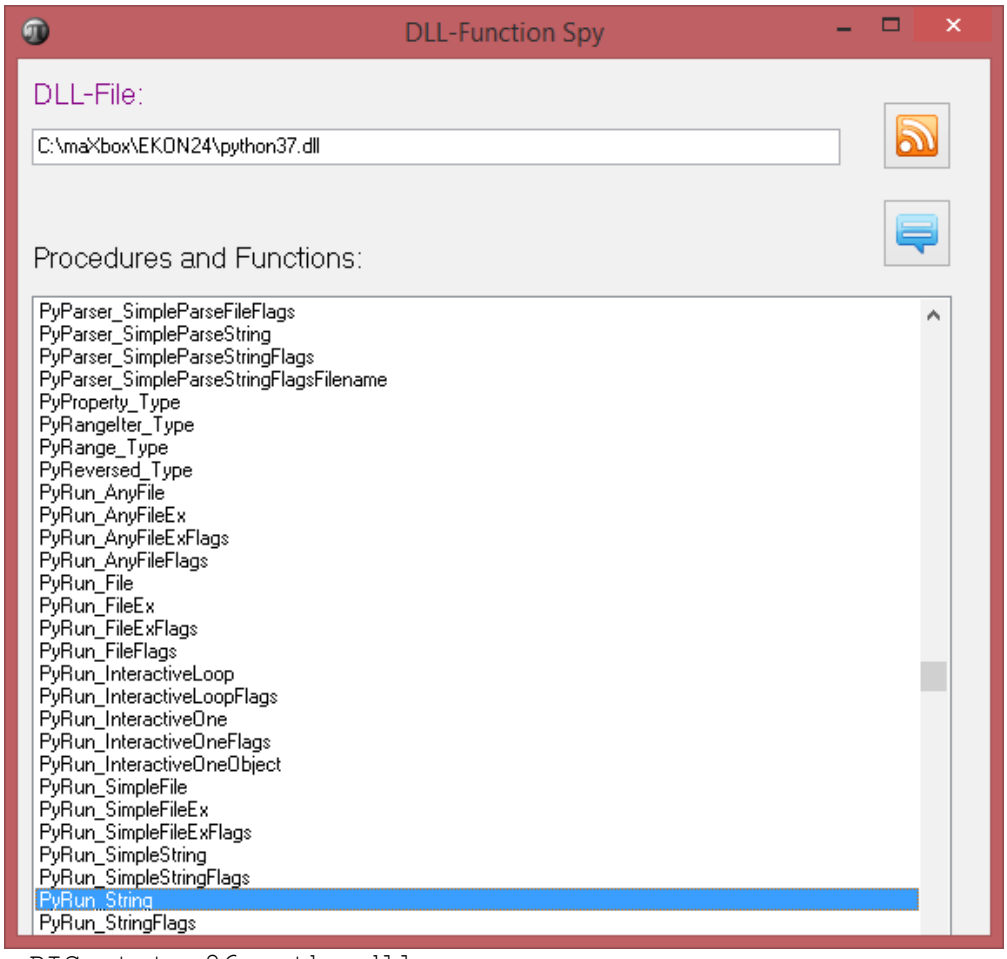
Werfen wir einen letzten Blick auf die Funktionalität von *PyRun_SimpleString*, die zuerst im const-Skript erwähnt wird.

http://www.softwareschule.ch/examples/1016_newsfeed_sentiment_integrate2.txt

```
PyRun_SimpleString: function(str: PAnsiChar): Integer; cdecl;
```

Wir sehen, dass wir ein *PAnsiChar* in der cdecl-Konvention übergeben und auf die ExecString (*PyRun_String:*) abbilden müssen:

```
procedure TPythonEngine.ExecString(const command : AnsiString);
begin
  Py_XDecRef( Run_CommandAsObject( command, file_input ) );
end;
```



_PIC: tutor86_pythondll_spy.png

## Konklusion

Die P4D-Bibliothek bietet eine bidirektionale Brücke zwischen Delphi und Python. Es ermöglicht Delphi-Anwendungen, auf Python-Module zuzugreifen und Python-Skripte auszuführen.
Auf der anderen Seite macht es Delphi/Lazarus-Objekte, -Records, -Interfaces und -Klassen für Python zugänglich und gibt Python die Möglichkeit, diese Methoden zu verwenden. Bevor Sie die Demos ausprobieren, lesen Sie bitte das Wiki-Thema „Wie Python für Delphi Ihre Python-Distribution findet" unter:

https://github.com/pyscripter/python4delphi/wiki/FindingPython

Sie müssen die Demos entsprechend anpassen, um die Python-Distribution, die Sie auf Ihrem PC installiert haben, erfolgreich zu laden oder in der Shell zu starten, Bspw.

C:\Users\max\AppData\Local\Programs\Python\Python36\.

- **PythonEngine:** Der Kern von Python für Delphi. Bietet die Python API mit DLL-Mapper und Laufzeitkonfiguration.
- **VarPyth:** VarPyth verpackt Python-Typen als benutzerdefinierte Delphi-Varianten.
- **WrapDelphi:** Verwendet RTTI (in einer DLL), damit Sie Delphi-Objekte aus Python verwenden können, ohne einen individuellen Wrapper zu schreiben.
- **TPythonGUIInputOutput:** Stellt eine Python-Konsole bereit, die Sie in ein Formular einfügen und ein Python-Skript aus einem Memo ausführen können.

_PIC: tutor86_Python4Delphilogo.png

## Wiki P4D topics

- Installation
- Supported Platforms
- How Python for Delphi finds your Python distribution (read before trying the demos)

## Learn about Python for Delphi

- Tutorials
- Demos
  https://github.com/maxkleiner/python4delphi

Das Skript ist zu finden:

http://www.softwareschule.ch/examples/1016_newsfeed_sentiment_integrate2.txt

Ref Video:

https://www.youtube.com/watch?v=jLuxTfct3CU

https://github.com/pyscripter/python4delphi/wiki/FindingPython

https://maxbox4.wordpress.com

## Appendix: Katalog der Demos:

```
Demo01  A simple Python evaluator
Demo02  Evaluate a Python expression
Demo03  Defining Python/Delphi vars
Demo04  Defining Python/Delphi vars (advanced case)
Demo05  Defining a new Module
Demo06  Defining a new Type
Demo07  Using Delphi methods as Python functions
Demo08  Using Delphi classes for new Python types
Demo09  Making a Python module as a Dll
Demo10_FireDAC Database demo using FireDAC
Demo11  Using Threads inside Python
Demo16  Using a TDelphiVar or Module methods
Demo17  Using variant arrays of 2 dimensions
Demo19  C++ Builder: this is a replicate of the Delphi Demo05
Demo20  C++ Builder: this is a replicate of the Delphi Demo08
Demo21  Using Events in TPythonModule or TPythonType
Demo22  Using Threading, Windows Console and Command line arguments
Demo23  Using Threading and Delphi log window
Demo25  Using VarPyth.pas
Demo26  Demo8 revisited to allow the new Python type to be subclassed
Demo27  Container indexing
Demo28  Iterator
Demo29  Using Python Imaging Library (PIL)
Demo30  Using Named Parameters
Demo31  Using WrapDelphi to access Delphi Form attributes
Demo32  Demo08 revisited using WrapDelphi
Demo33  Using Threads inside Python
Demo34  Dynamically creating, destroying and recreating PythonEngine.
```