Dr. Veikko Krypczyk

Das grafische User-Interface für die Python-Anwendung:

Überblick und Möglichkeiten

#ittage

Agenda

- GUI-Optionen
- Python installieren
- GUI mit TkInter
- GUI mit RAD Studio (Delphi)
- Fazit

Über uns…

LARInet (Learn Read Implement)

IT-Dienstleister

- Development (App/ Web)
- Design
- Workshops/ Seminare
- Unternehmenskommunikation
- Pressearbeit
- Konzepte
- Whitepapers
- Fachartikel

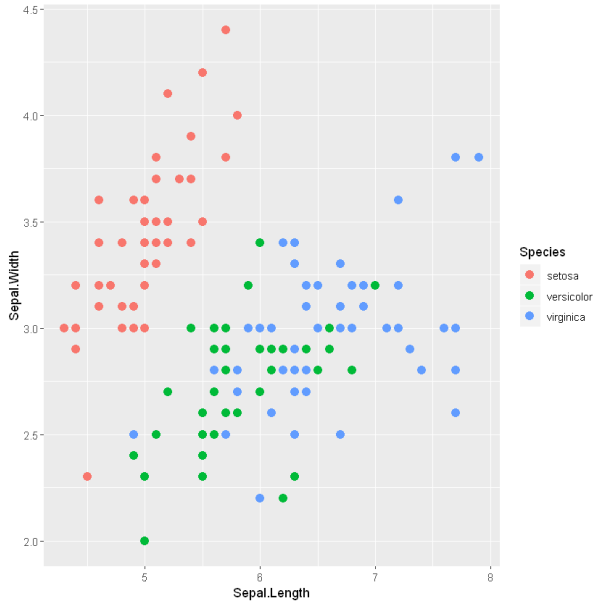
Dr. Veikko Krypczyk

Elena Bochkor

Motivation

Python ist eine leistungsfähige Programmiersprache (Ökosystem) für:

- Mathematik
- Statistik
- wissenschaftliche graphische Darstellungen
- Datenverarbeitung
- Künstliche Intelligenz
- Algorithmen
- …



Bibliotheken rund um Data Science und KI

- *TensorFlow*, https://www.tensorflow.org/: Es ist ein beliebtes Open Source Deep Learning-Framework für numerische Berechnungen. Die Bibliothek wird auch für maschinelles Lernen verwendet. TensorFlow wurde von den Forschern des Google Brain-Teams innerhalb der Google AI-Organisation entwickelt und wird heute von vielen Wissenschaftlern eingesetzt.

- *scikit-learn*, https://scikit-learn.org/: Es handelt sich um eine Bibliothek für maschinelles Lernen. Sie kann für eine Vielzahl von Anwendungen verwendet werden, darunter Klassifizierung, Regression, Clustering und Modellauswahl.

Bibliotheken rund um Data Science und KI

- *Matplotlib*, https://matplotlib.org/: Diese Open Source-Bibliothek wird häufig für die Visualisierung mit Hilfe von Diagrammen verwendet. Mit nur wenigen Codezeilen können Grafiken, Tortendiagramme, Streudiagramme, Histogramme usw. erstellt werden.

- *SymPy*, https://www.sympy.org/: Diese Python-Bibliothek bietet umfassende Methoden für symbolische Berechnungen. Symbolisches Rechnen ermöglicht das Lösen von Gleichungen, zum Beispiel die Auflösung nach einer bestimmten Variablen. Dabei wird eine exakte Lösung ermittelt und nicht nur ein numerischer Näherungswert bestimmt.

GUI-Optionen für Python-Skripte

Python und GUI

- Python ist eine leistungsfähige Programmiersprache (inzwischen Ökosystem) für
  - Mathematik
  - Datenverarbeitung
  - Künstliche Intelligenz
  - Algorithmen
  - …
- Mit einem **T**ool**k**it (Tk) kann man damit auch ein grafisches User Interface (GUI) erstellen

„Eigenes“ oder „fremdes“ GUI?

- „Eigenes“ User Interface:
  - die Benutzeroberfläche wird wie die anderen Programmteile in Python programmiert
  - Einsatz eines Toolkits
- „Fremdes“ User Interface:
  - die Benutzeroberfläche wird durch ein eigenständiges Programm realisiert
  - durch Grafikbibliothek oder -framework
  - das Python Programm stellt der Software seine Dienste in Form einer Bibliothek (API) zur Verfügung
  - empfehlenswert, wenn es sich um umfangreiche oder komplexere Oberflächen handelt

Toolkits

- **TkInter (https://wiki.python.org/moin/TkInter):**
  - es handelt sich um das Standard Interface für grafische Oberflächen für Python-Skripte
  - es bindet das Toolkit Tk, welches ursprünglich für die Sprache TCL (Tool Command Language) entwickelt wurde
  - TkInter ist direkt in der Standardbibliothek von Python enthalten
  - geeignet für kleineren grafische Benutzeroberflächen, ohne eine zusätzliche Bibliothek zu installieren und zu nutzen
- **PyQt [https://riverbankcomputing.com/software/pyqt/intro]**:
  - angebunden wird das Toolkit Qt, welches die Basis der Desktop-Umgebung KDE (K Desktop Environment) ist
  - Qt steht jedoch auch plattformübergreifend zur Verfügung und kann für freie Software kostenfrei verwendet werden
  - für eine kommerzielle Nutzung von Qt fallen Lizenzgebühren an
  - Qt ist in der Programmiersprache C++ entwickelt

Toolkits

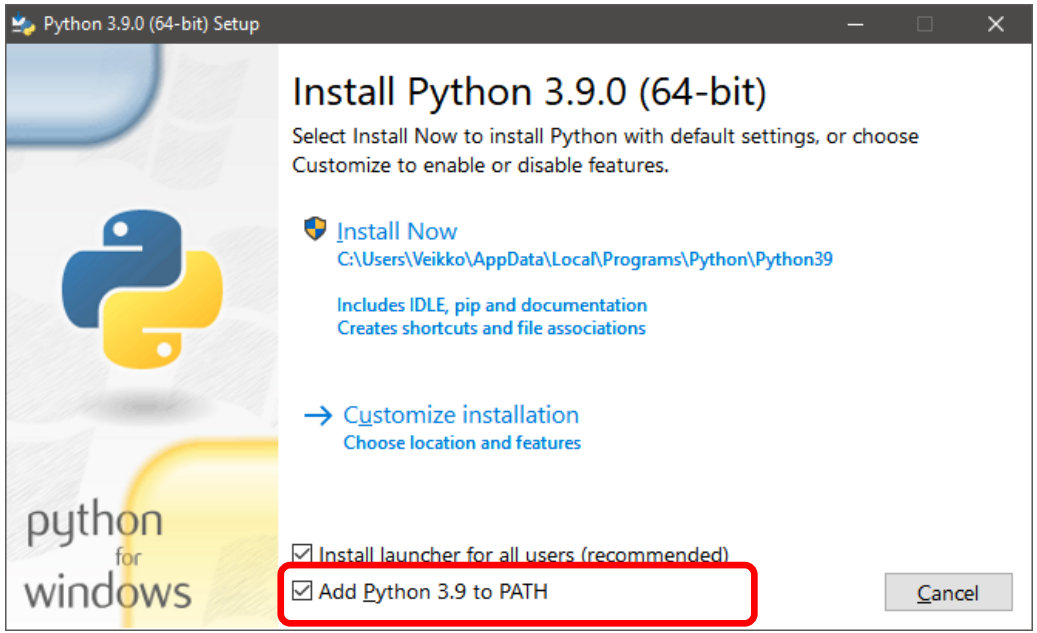
- **PyGObject [https://pygobject.readthedocs.io/en/latest/]:**
  - es wird das Grafik-Toolkit GtK verwendet.
  - GtK (GIMP-Toolkit) wurde ursprünglich für das Grafikprogramm GIMP entwickelt
  - plattformübergreifendes Framework, welches häufig für grafische Benutzeroberflächen eingesetzt wird
  - ist Basis der Desktop-Umgebung GNOME
  - es ist in C programmiert und kann objektorientiert verwendet werden
- **wxPython [https://wxpython.org/]:**
  - es wird mit dem plattformübergreifenden Grafikframework wxWidgets gearbeitet
  - das Ziel von wxWidgets ist es die Besonderheiten der jeweiligen Zielplattform möglichst gut abzubilden
  - Es zeichnet die Controls nicht selbst, sondern verwendet die APIs der Zielsysteme

Python installieren

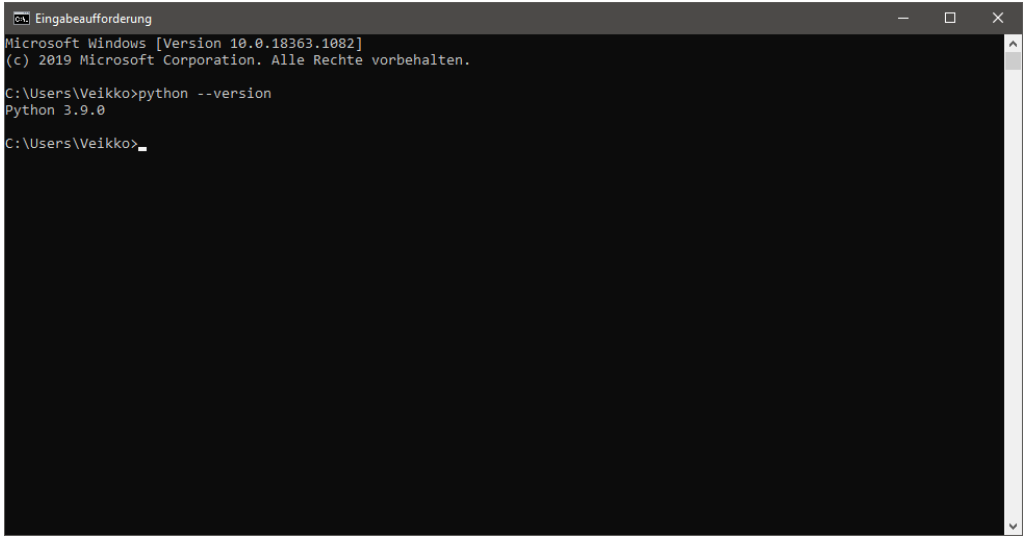
Python: Download

Python installieren: https://www.python.org/ (aktuelle Version)

Python-Installation unter Windows

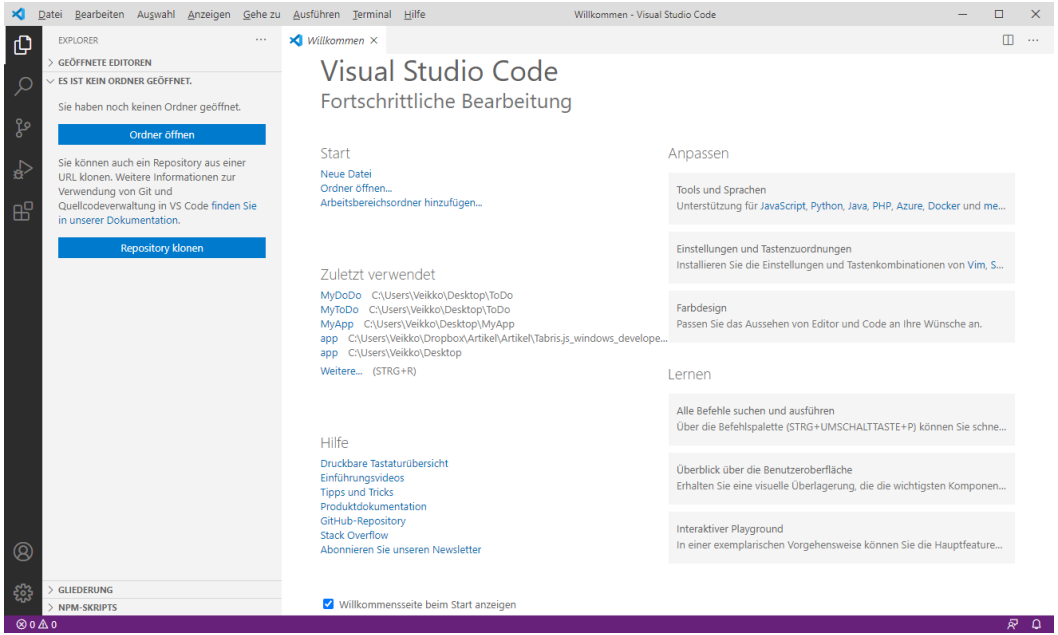


Python: Versions-Prüfung



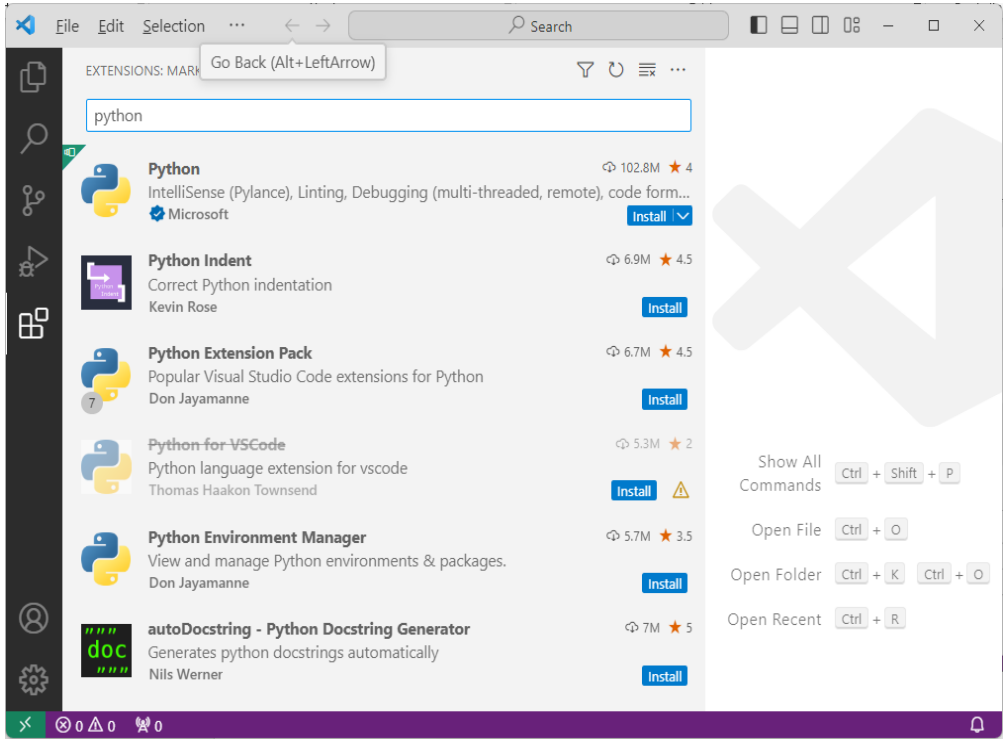
- Reboot
- Check auf der Kommandozeile (CMD): *python --version*

Editor für das Schreiben des Skriptes



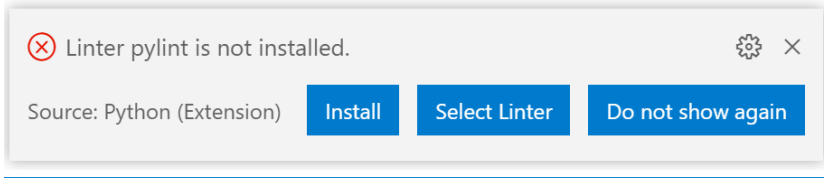
zum Beispiel Visual Studio Code: *https://code.visualstudio.com/*

Extensions für Python in VS Code einbinden



Tool-Tipp: Linting Python in Visual Studio Code

- es werden syntaktische und stilistische Probleme im Python-Quellcode hervorgehoben
- es werden beispielsweise die Verwendung einer nicht initialisierten oder nicht definierten Variablen erkannt, fehlende Klammern korrigiert usw.



- das Python Programm kann dann direkt aus VS Code gestartet werden

TkInter

TkInter

- *TkInter* ist der Standard für grafische Benutzeroberflächen, welche man direkt aus Python erstellt
- keine separate Installation notwendig

Hello World

```
from tkinter import *
# Ein Fenster erstellen
window = Tk()
# Den Fenstertitel erstellen
window.title("Hello World")
# Ereignisschleife auf Reaktion des Benutzers warten.
window.mainloop()
```

`from tkinter import *`
Wir importieren als erstes das Modul *tkinter* mit allen Funktionen und Methoden

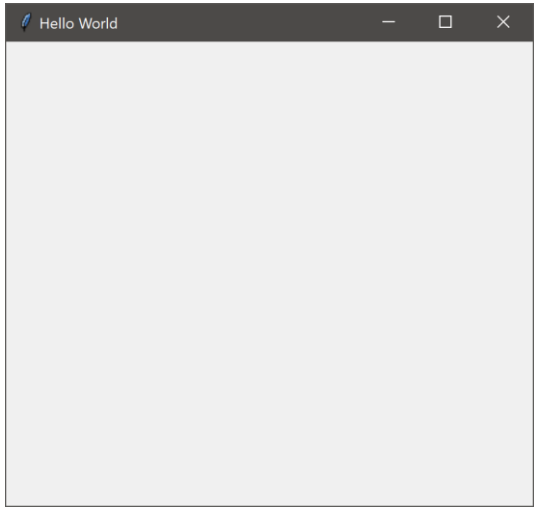
`window = Tk()`
danach wird ein neues Fenster erstellt

`window.title("Hello World")`
die Eigenschaft *title* erhält einen Wert und die Programmschleife wird gestartet.

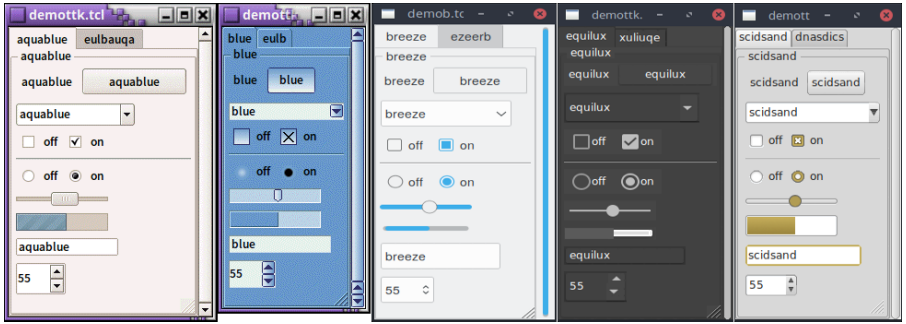
`window.mainloop()`
In dieser Programmschleife verweilt das Programm und wartet auf Reaktionen des Anwenders, ggf. auf ein Beenden, d.h. Schließen des Programmfensters.

Start

- direkt aus dem Editor
- oder über die Kommandozeile: *python Hello.py*
- das Programm arbeitet plattformneutral und kann auf allen gängigen Desktop-Betriebssystemen ausgeführt werden



UI-Widgets

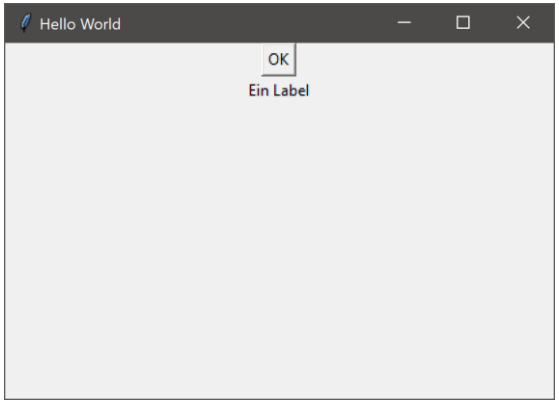


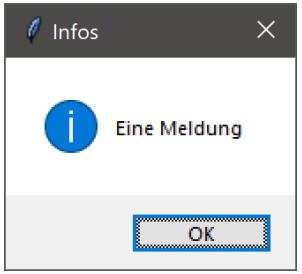
Quelle: https://tkdocs.com/tutorial/styles.html

- *Widget*: Die Basisklasse aller Steuerelemente.
- *Button*: Eine Schaltfläche.
- *Canvas*: Ein Steuerelement für Zeichnungen und Grafiken.
- *Checkbutton*: Ein Steuerelement, das entweder aktiviert oder deaktiviert sein kann.
- *Entry*: Ein einzeiliges Eingabefeld.
- *Label*: Ein Steuerelement für Beschriftungen.
- *LabelFrame*: Ein Steuerelement für beschriftete Rahmen.
- *Listbox*: Eine Liste von Einträgen
- …

Beispiel mit Widgets

```
from tkinter import *
from tkinter import messagebox

# Event-Definition
def button_action():
messagebox.showinfo(message="Eine Meldung", title = "Infos")
window = Tk()
window.title("Hello World")

# Definition von GUI-Elementen
Button = Button(window, text="OK", command=button_action)
label = Label(window, text="Ein Label")

# Hinzufügen der Elemente zum Fenster
button.pack()
label.pack()

# Ereignisschleife auf Reaktion des Benutzers warten.
window.mainloop()
```





Ein Menü in Python

```
…
# Menüleiste erstellen
menuleiste = Menu(window)

# Menü Datei und Help erstellen
datei_menu = Menu(menuleiste, tearoff=0)
help_menu = Menu(menuleiste, tearoff=0)

# Submenüs
datei_menu.add_command(label="Neu")
datei_menu.add_separator() # Fügt eine Trennlinie hinzu
datei_menu.add_command(label="Beenden", command=window.quit)
menuleiste.add_cascade(label="Datei", menu=datei_menu)
menuleiste.add_cascade(label=„Hilfe", menu=help_menu)

# Die Menüleiste mit den Einrägen dem Fenster übergeben
window.config(menu=menuleiste)
…
```

Layout

- *pack():* Relative Positionierung der Elemente
- *grid():* Anordnung der Elemente in einer Tabelle (Zeilen und Spalten)
- (*place():* Absolute Positionierung der Elemente).

Hinweis: absolute Positionierung nur bedingt für plattformübergreifende Lösungen geeignet.

Layout mit Pack()

- Die Steuerelemente sind in Tk hierarchisch angeordnet. Das bedeutet, dass jedes Steuerelement über ein übergeordnetes Element verfügt. Außerdem darf jedes Element beliebig viele Kind-Elemente enthalten.
- Dem Packer kann ein Layout vorgegeben werden, nachdem er die Elemente anzuordnen hat.
- Der Packer arbeitet stets in einem rechteckigen Teilbereich des Fensters. Sofern noch kein Element vorhanden ist, umfasst dieser Bereich das gesamte Fenster.

Pack() - Beispiel

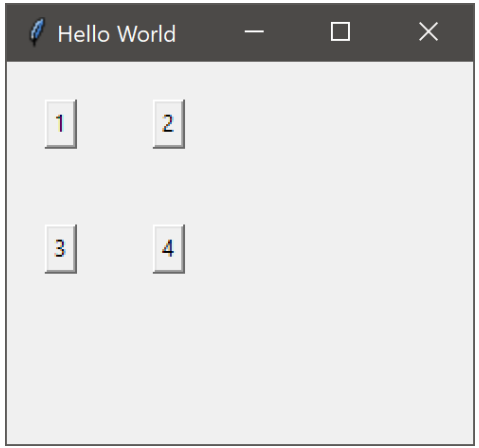
```
# Definition von GUI-Elementen
button=Button(window, text="OK")
button2=Button(window, text="Abbrechen")

# Hinzufügen der Elemente zum Fenster
button.pack(side=LEFT, padx="20", pady="20")
button2.pack(side=LEFT, padx="20", pady="20")
```

Layout mit Grid()

- Dieser platziert die Komponenten in einer Tabelle. Die Position einer Komponente wird durch einen Zeilen- (*row*) und einen Spaltenwert (*column*) bestimmt. Mit der Methode *grid()* übergibt man die Information, wo die betreffende Komponente platziert werden soll.
- Die Optionen der Methode *grid()* sind:
  - *row*: Bestimmt in welcher Zeile man die Komponente setzen möchte.
  - *column*: Bestimmt in welcher Spalte man die Komponente setzen möchte.
  - *padx*: Diese Option kann man gebrauchen, wenn man in der Horizontalen noch zusätzlich Abstand an die jeweilige Komponente haben möchte.
  - *pady*: Analog wie *padx*, jedoch in vertikaler Richtung.
- Mittels *grid()* ist ein einfacher Aufbau auch komplexerer Layouts möglich.

Grid()-Beispiel

```
# Definition von GUI-Elementen
button1=Button(window, text="1")
button2=Button(window, text="2")
button3=Button(window, text="3")
button4=Button(window, text="4")

# Hinzufügen der Elemente zum Fenster
button1.grid(row=0, column=0, padx=20, pady =20)
button2.grid(row=0, column=1, padx=20, pady =20)
button3.grid(row=1, column=0, padx=20, pady =20)
button4.grid(row=1, column=1, padx=20, pady =20)
```



Mehrere Formulare

```
import tkinter as tk

def create_window():
        window = tk.Toplevel(root)

root = tk.Tk()
b = tk.Button(root, text="Create new window", command=create_window)
b.pack()
root.mainloop()
```

Weitere Features

- In Bezug auf TkInter gibt es noch viele weitere Funktionen, beispielsweise
- Zeichnen
  - mittels des *Canvas*-Widgets komplette geometrische Zeichnungen erstellen
  - diese können dann zum u.a. dynamisch aus den Daten generiert werden
  - es stehen die üblichen Zeichenbefehle wie: *create_oval(), create_line(), create_image()* zur Verfügung
- Standarddialoge: *tkinter.filedialog, tkinter.font, tkinter.messagebox*

Zwischenfazit

- TkInter ist das Standard-Toolkit zum Erstellen von Benutzeroberflächen für Python-Programme
- Vorteile:
  - direkt aus Python verfügbar
  - plattformübergreifend
  - einfacher Ansatz
- Nachteile:
  - nur Standard-Controls
  - keine komplexen und modernen UIs
  - manuelles Erstellen des UIs

GUI mit RAD Studio

User Interface-Gestaltung mit Delphi

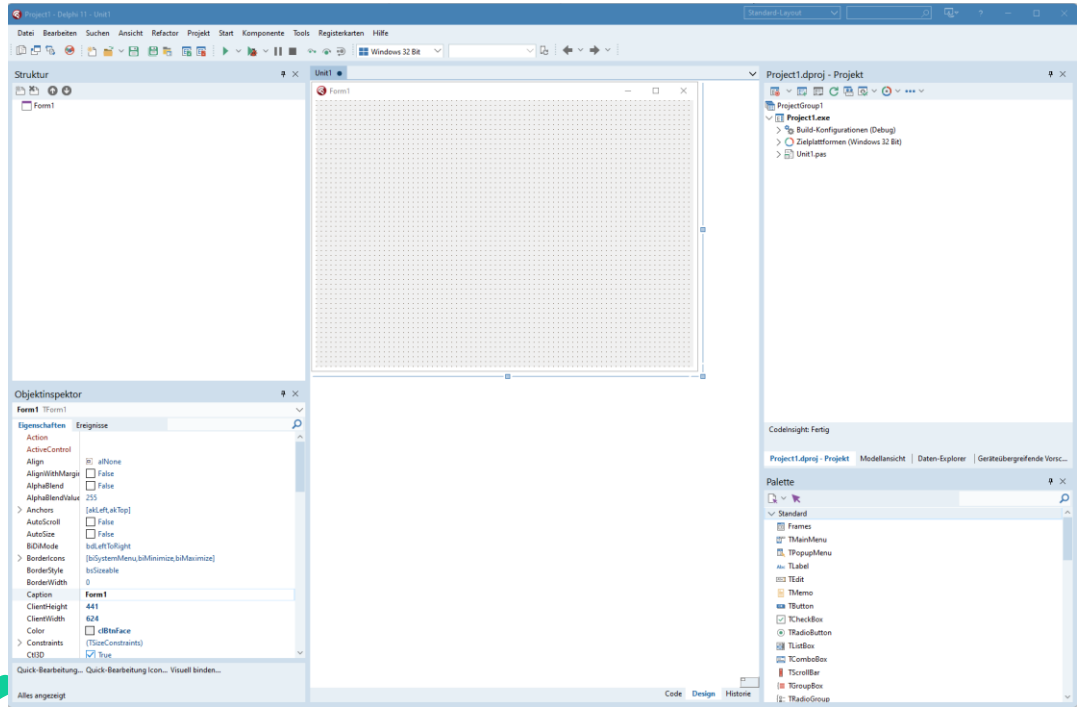
- mit Delphi kann man native Applikationen für alle gängigen Client-Systeme erstellen und diese mit modernen grafischen Benutzeroberflächen ausstatten
- dazu stehen leistungsfähige Steuerelemente (UI-Controls) zur Verfügung
- statt die Benutzeroberfläche im Quellcode per Hand zu codieren, arbeitet man mit einem grafischen Designer
- Programmiersprache: Delphi (Object Pascal)
- Community Edition verfügbar

Grafikframeworks in Delphi

- Visual Component Library (VCL):
  - Erstellen von grafischen Benutzeroberflächen für Windows Betriebssysteme
  - Aufbau des UI mittels Controls
  - auch die neuen Design-Features von Windows 11 werden unterstützt

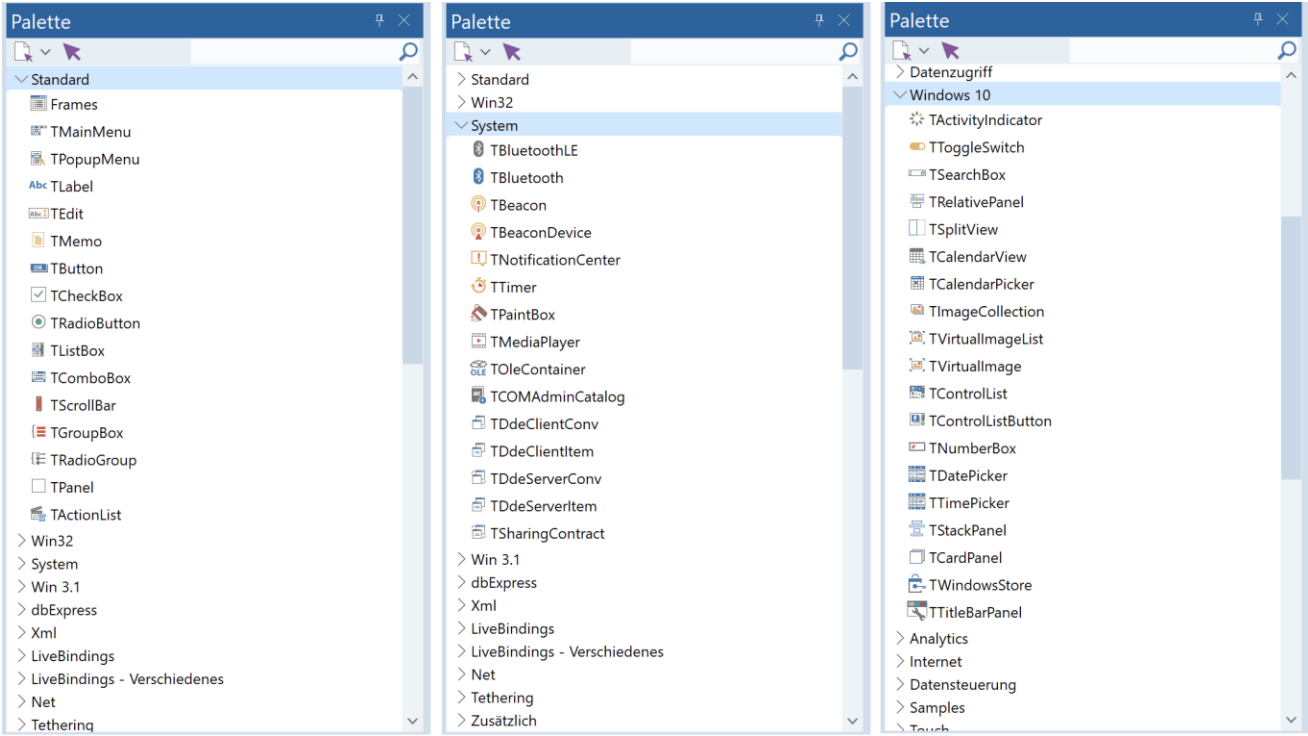
- FireMonkey (FMX):
  - für plattformübergreifende Applikationen, d.h. Windows, macOS, Linux, Android, iOS
  - gemeinsame Quellcodebasis für die Businesslogik und die grafische Oberfläche

Delphi: grafischer Designer



- integrierte Entwicklungsumgebung
- RAD-Ansatz
- graphischer Designer
- visuelle und nicht visuelle Komponenten

Delphi: Palette von UI-Controls



Python mit Delphi kombinieren

- Delphi: App und UI-Entwicklung
- Python: Nutzung der Bibliotheken aus den Bereichen Data Science, maschinelles Lernen, künstliche Intelligenz usw., beispielsweise: *TensorFlow*, *Scikit-Learn, SciPy, …*
- Anwendungsprogramme können auf diese Weise sehr einfach um Fähigkeiten der KI angereichert werden

Kombination von Delphi und Python: Zielgruppe

Datenwissenschaftler

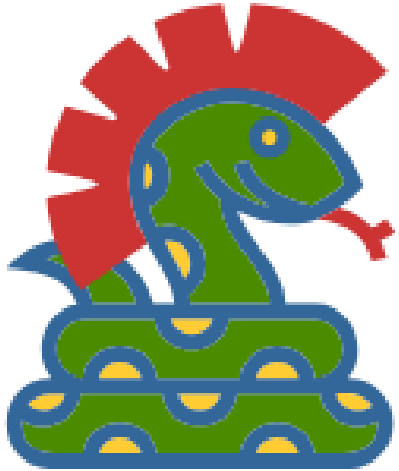
- diese arbeiten mit der Programmiersprache Python
- Expertise liegt insbesondere in den Bereichen KI, Deep Learning und ähnlicher Technologien
- hat ein Modell einen bestimmten Reifegrad erreicht, dann soll es oft in ein Anwendungssystem eingebettet werden
- IDE mit einem grafischen Designer erleichtert die Umsetzung einer App mit einer grafischen Oberfläche
- Zugriff auf Datenbanken und Systemfunktionen notwendig

Anwendungsentwickler

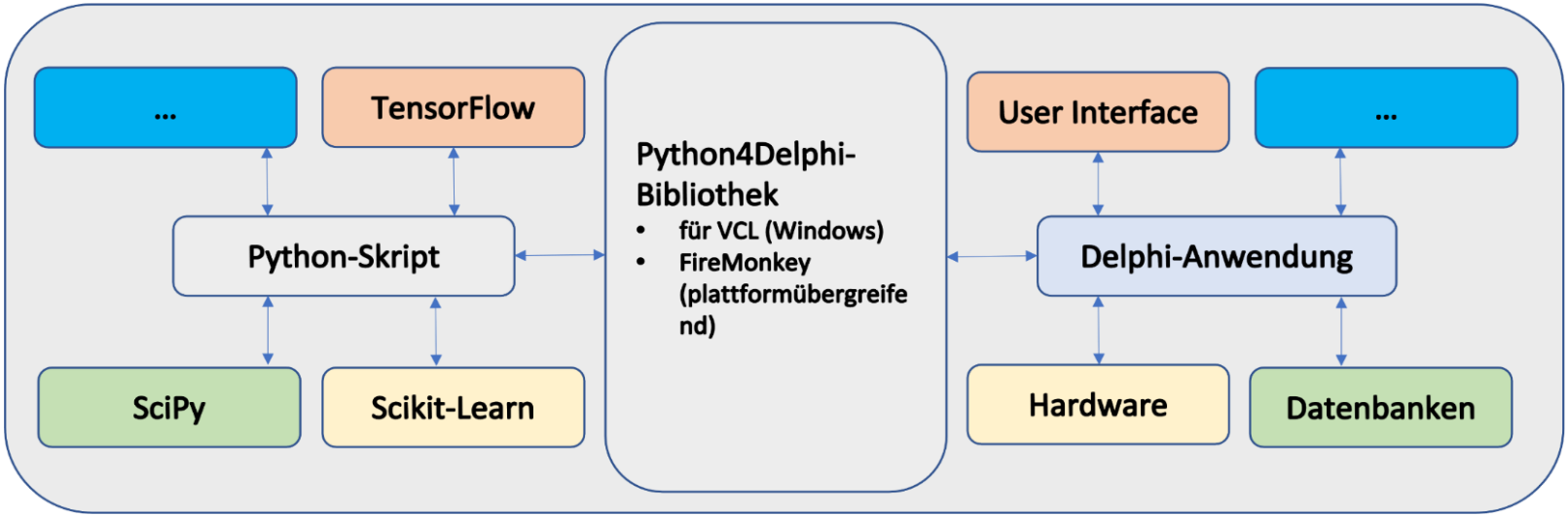
- Nutzung der Bibliotheken und der Skripte der Programmiersprache Python in eigenen Anwendungen
- Funktionen auf der Basis von KI, Deep Learning, Datenauswertung werden direkt nutzbar

Bibliothek Python4Delphi (Open Source)

- Installation und Nutzung aus Delphi
- Low-Level-Zugriff auf das Python API
- bidirektionale Kommunikation zwischen Python- und Delphi-Programm
- Zugriff auf Python-Objekte über benutzerdefinierte Datentypen in Delphi
- Nutzung von Delphi-Objekten aus einem Python-Skript mittels Wrappers
- Erstellen von Python-Extension Modulen mittels Klassen und Funktionen aus Delphi

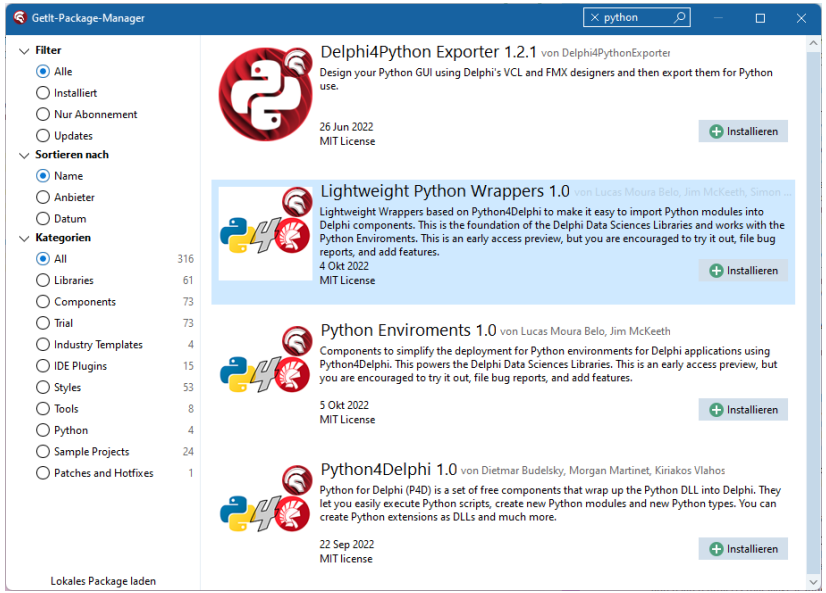


Kombination von Python und Delphi



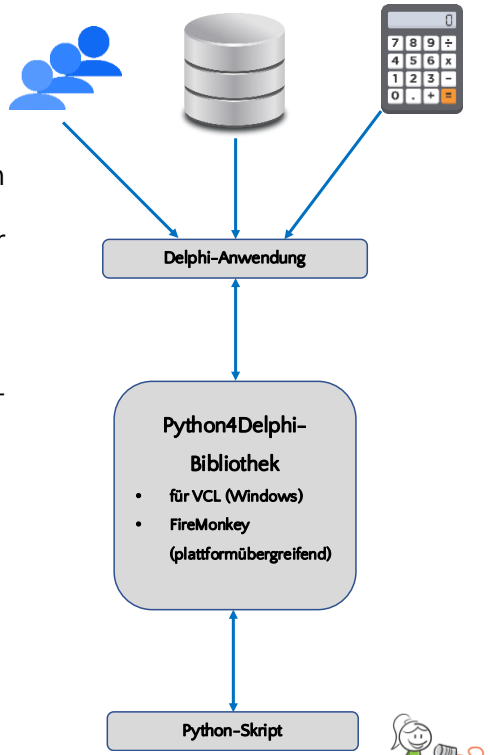
Voraussetzungen/ Installation

- Windows 10/ Windows 11
- Delphi, Community Edition
- Python, aktuelle Version
- *Python4Delphi*-Bibliothek
  - manuelle Installation
  - *GetIt* (Paketmanager von Delphi)

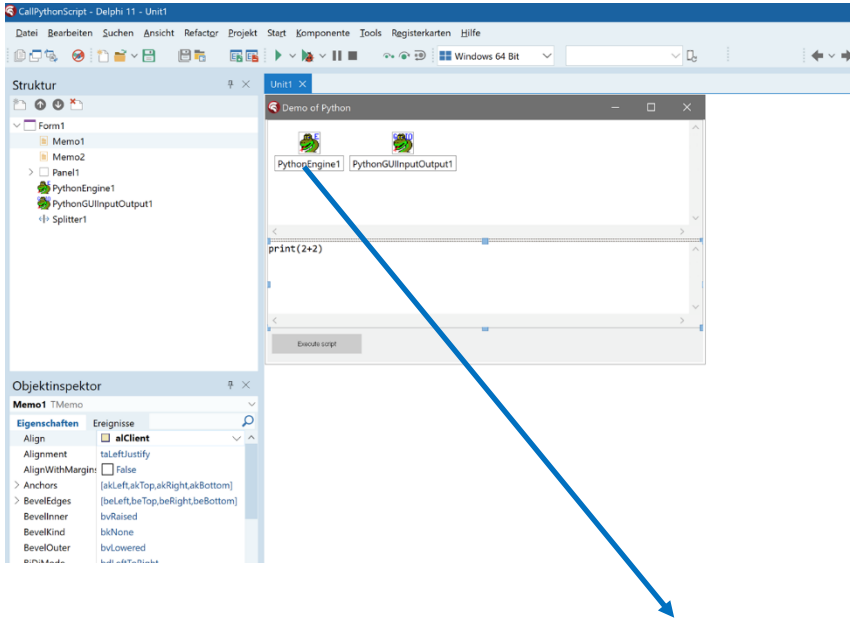


Python-Skript aus Delphi aufrufen: Funktionsweise

- Ziel: Python-Skript aus Delphi-Anwendung ausführen
- das Skript kann zum Beispiel über die Benutzeroberfläche eingegeben werden oder durch berechnete Werte und Ergebnisse aus einer Datenbankabfrage generiert werden
- das Skript wird aus der Delphi-Anwendung mit Hilfe der Python4Delphi-Bibliothek an den Python-Interpreter übermittelt
- die ermittelten Ergebnisse werden an das aufrufende Programm gesendet



Python-Skript aus Delphi aufrufen: Umsetzung




```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PythonEngine1.ExecStrings( Memo1.Lines );
end;
```

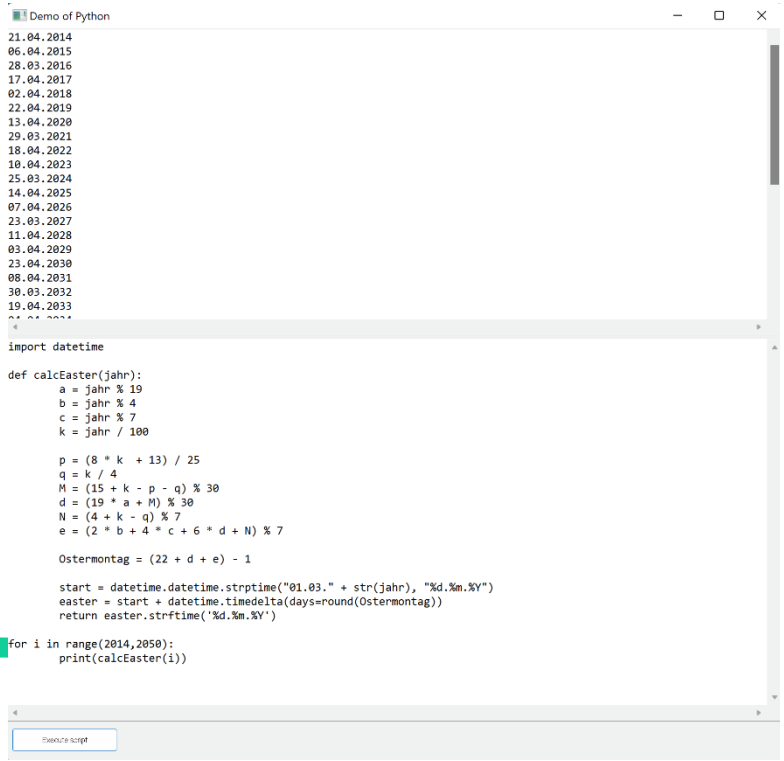
- VCL-Anwendung mit folgenden Komponenten:
  - *TMemo*: für die Erfassung des Python-Skriptes
  - *TMemo*: für die Ausgabe der Ergebnisse nach der Ausführung des Python-Skriptes.
  - *TButton*: dient dem Start des Python Skriptes, d.h. der Übermittlung der Daten an den Python-Interpreter.
  - *TPython-Engine*: das ist eine Komponente aus der Bibliothek *Python4Delphi*, sie dient der Interaktion mit dem Python-Interpreter.
  - *TPythonGUIInputOutput*: diese Komponente stammt ebenfalls aus der Python4Delphi-Bibliothek und dient dem Datenaustausch zwischen Delphi und Python.

Python-Skript (Osteralgorithmus)

```
import datetime
def calcEaster(jahr):
a = jahr % 19
b = jahr % 4
c = jahr % 7
k = jahr / 100
p = (8 * k  + 13) / 25
q = k / 4
M = (15 + k - p - q) % 30
d = (19 * a + M) % 30
N = (4 + k - q) % 7
e = (2 * b + 4 * c + 6 * d + N) % 7
Ostermontag = (22 + d + e) - 1
start = datetime.datetime.strptime("01.03." + str(jahr), "%d.%m.%Y")
easter = start + datetime.timedelta(days=round(Ostermontag))
return easter.strftime('%d.%m.%Y')
for i in range(2014,2050):
print(calcEaster(i))
```

Quelle: https://www.uweziegenhagen.de/?p=3093

Ausführen der Anwendung

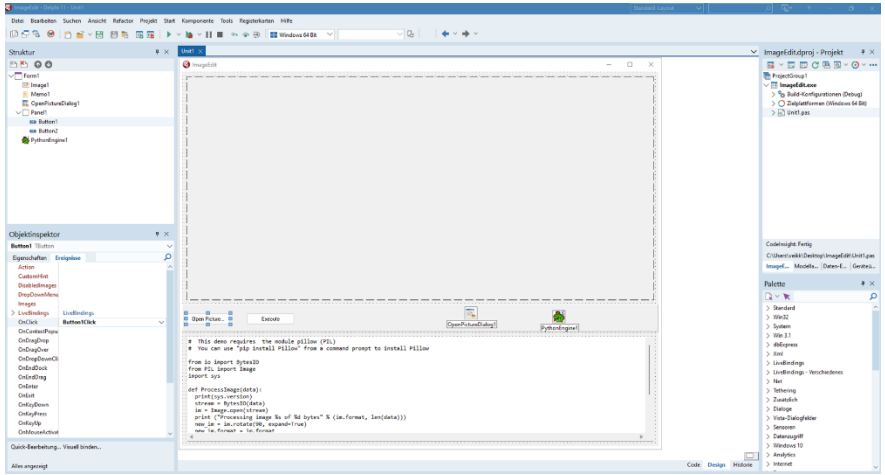


- mit Klick auf den Button wird das Skript an den Python-Interpreter übermittelt
- dort werden die Berechnungen durchgeführt und an das aufrufende Programm zurückgegeben
- die Ergebnisse werden in der Komponente des Typs *PythonGUIInputOutput* angezeigt

Bildbearbeitung mit Python und Delphi

- Programm inklusive grafischer Benutzeroberfläche zur Bildmanipulation
- die Anzeige des Originalbildes und des veränderten Bildes erfolgen im Programm

Bildbearbeitung: User Interface



- Komponenten für den Aufbau der Benutzeroberfläche:
  - *TImage*: Anzeige des Bildes
  - *TPanel*: nimmt zwei Komponenten des Typs *TButton* (Schaltfläche) auf
  - *TMemo*: eine mehrzeilige Eingabeoberfläche für die Erfassung des Python-Skriptes
  - *TButton*: zwei Schaltflächen, für die Auswahl des Bildes und für die Ausführung des Skriptes
  - *TOpenPictureDialog*: ermöglicht Auswahl einer Bilddatei aus dem Dateisystem
  - *TPythonEngine*: Verbindung zwischen Delphi- und Python-Anwendung

Bildbearbeitung: Delphi-Programm

```
procedure TForm1.Button2Click(Sender: TObject);
var
  _im: Variant;
  _stream: TMemoryStream;
  _dib: Variant;
  pargs: PPyObject;
  presult: PPyObject;
  P: PAnsiChar;
  Len: NativeInt;
begin
  if (Image1.Picture.Graphic = nil) or Image1.Picture.Graphic.Empty then
    raise Exception.Create('You must first select an image');
  PythonEngine1.ExecStrings(Memo1.Lines);
  _im := MainModule.ProcessImage(ImageToPyBytes(Image1.Picture.Graphic));
  Image1.Picture.Bitmap.SetSize(Image1.Width, Image1.Height);
  _dib := Import('PIL.ImageWin').Dib(_im);
  Image1.Picture.Bitmap.SetSize(Image1.Height, Image1.Width);
  _dib.expose(NativeInt(Image1.Picture.Bitmap.Canvas.Handle));
end;
```

- das Python-Skript wird der *Lines-Eigenschaft* der *Memo*-Komponente entnommen
- es wird mittels *PythonEngine1.ExecStrings(…)* an den Python-Interpreter übermittelt
- das Ergebnis ist der Datenstrom des modifizierten Bildes
- das modifizierte Bild wird auf der Programmoberfläche angezeigt

Bildbearbeitung: Python-Skript

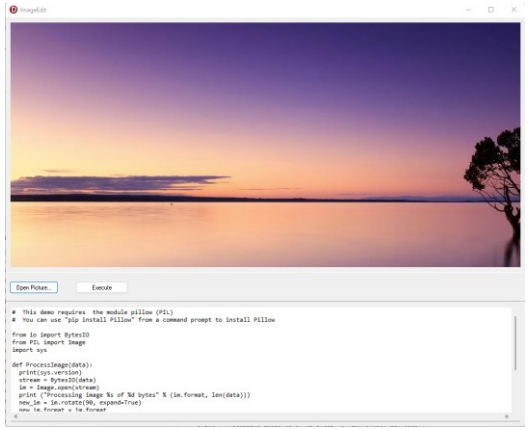
```
from io import BytesIO
from PIL import Image
from PIL import ImageOps
import sys

def ProcessImage(data):
  print(sys.version)
  stream = BytesIO(data)
  im = Image.open(stream)
  print ("Processing image %s of %d bytes" % (im.format, len(data)))
  new_im = im.rotate(90, expand=True)
  #new_im=ImageOps.mirror(im)
  new_im.format = im.format
  return new_im
```

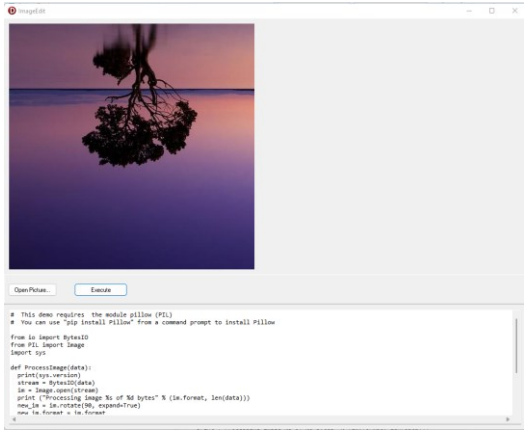
- der Befehl für eine Bildrotation lautet : *new_im = im.rotate(90, expand=True)*
- Spiegeln des Bildes: *new_im=ImageOps.mirror(im)*

Python Imaging Library (PIL):

- Bild öffnen und anzeigen
- Bildmanipulation, z.B. drehen, skalieren, zuschneiden, spiegeln, konvertieren
- Bildfilter und Effekte, z.B. Unschärfe, Schärfen, Helligkeit, Kontrast, Farbkorrekturen
- Text und Annotationen hinzufügen, z.B. Wasserzeichen, Beschriftungen, Markierungen
- Bildspeicherung, z.B. in den Formaten JPEG, PNG, BMP und TIFF

Bildbearbeitung: Ergebnis

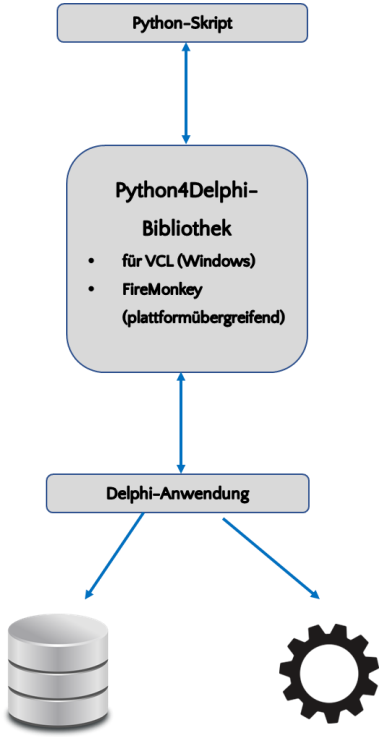




- Laden einer Bilddatei im Delphi-Programm
- Ausführen des Python-Skriptes aus Delphi
- Bilddatei wird durch Python-Library bearbeitet (Drehung um 90°)
- Rückgabe der bearbeiteten Bilddatei
- Anzeige des modifizierten Bildes durch das Delphi-Programm

Delphi-Objekte aus Python nutzen: Funktionsweise

- es geht auch der umgekehrte Weg
- aus einem Python-Skript kann eine Delphi-Funktion aufgerufen werden, dabei kann es sich um Berechnungen, Zugriffe auf Datenbanken oder Interaktionen mit weiteren Systembibliotheken und Hardware handeln



Fazit

Zusammenfassung

- Delphi und Python können mittels der *Python4Delphi*-Bibliotheken kombiniert werden
- typische Anwendung ist das Erstellen der App in Delphi mit einem umfassenden UI, Zugriff auf Datenbanken und Web-Service und den Aufruf des Python-Skriptes aus der Delphi-Applikation
- die Ergebnisse der Ausführung des Python-Skriptes werden an das Programm zurückgegeben
- auch der Zugriff auf Delphi-Objekte ist aus dem Python-Skript über die Python4Delphi-Bibliotheken möglich (bidirektionale Kommunikation)

Fragen und Diskussion

Materialien

Foliensatz und Quellcodebeispiele	• https://larinet.com
Python	• https://www.python.org/ • https://py-tutorial-de.readthedocs.io/de/python-3.3/
Delphi	• https://www.embarcadero.com/products/delphi • https://delphilernen.de/
Python4Delphi	• https://www.embarcadero.com/new-tools/python/delphi-4-python • https://github.com/pyscripter/python4delphi • https://blogs.embarcadero.com/getting-started-wit-python4delphi/

Vielen Dank für Ihre Aufmerksamkeit

Dr. Veikko Krypczyk

v.krypczyk@larinet.com

https://larinet.com/

Bilder von: https://pixabay.com