

Early access pdf edition available from
www.educational-computing.co.uk

Python4Delphi

Information

Why use Python4Delphi?

Python4Delphi combines the strengths of Delphi and Python. It supports Python-based data analytics in Delphi applications and Python GUI development using the VCL.

Information

The programs in this chapter rely on Python4Delphi software, inspiration being derived from the software demos provided with Python4Delphi. For licensing see <https://github.com/pyscripter/python4delphi/blob/master/LICENSE>

Purpose: To become familiar with the use of Python4Delphi

What is Python4Delphi?

Python¹ for Delphi (**Python4Delphi**) is a set of free components that wrap up the **Python dll** so that Python scripts can be executed from Delphi.

Figure 80.1 shows a simple demonstration in which a Python script

BubbleSort.py is loaded and executed from a Delphi program

Demo1Project.exe.

The upper window shows the version of python being used (**Python 3.8**), then the output, [0, 1, 2, 3, 5, 8, 9], from the execution of the python script,

BubbleSort.py, shown in the lower window, acting on the unsorted list, [5, 1, 2, 3, 9, 8, 0].

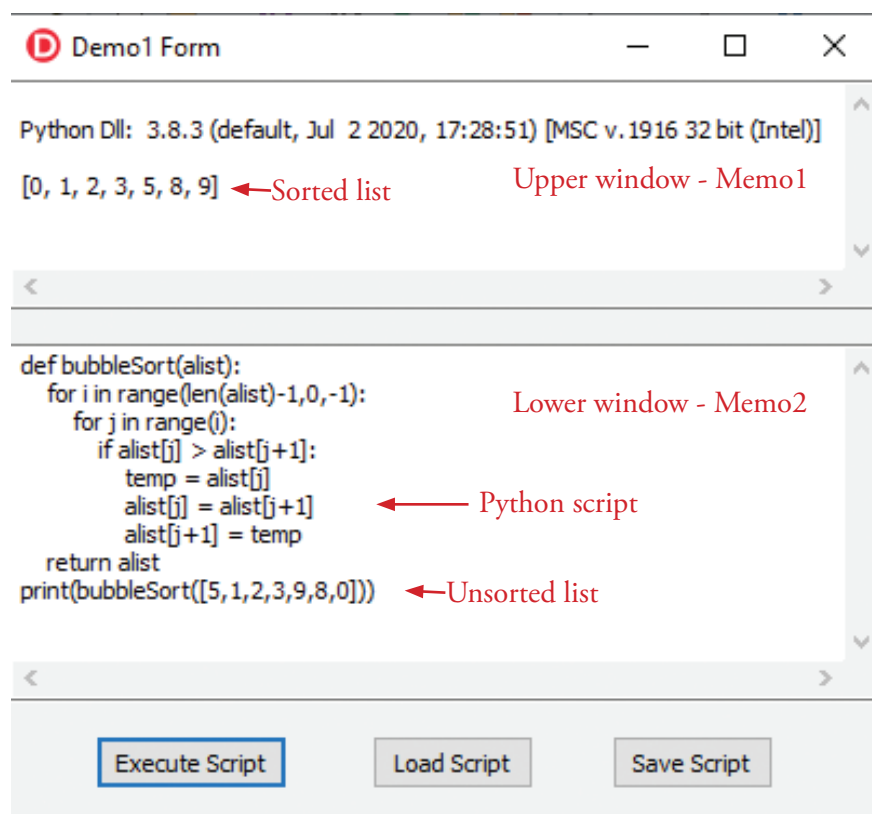


Figure 80.1 Demo1Project.exe executing Python script BubbleSort.py

www.educational-computing.com/DelphiBook/Code/Chapter80/Demo1Project.zip

¹ "Python" is a registered trademark of the Python Software Foundation

Setting up Python4Delphi

Download **python4delphi** code, **python4delphi-master.zip**, or clone this code from the GitHub repository at github.com/pyscripter/python4delphi. Unzip it to create a **python4delphi-master** folder with the contents shown in *Figure 80.2* (September 2020).

1. Start **RAD Studio**.
2. Add the source subdirectory (e.g., **C:\Users\drbond\python4delphi-master\Source**) to the IDE's library path for the targets you are planning to use (**Tools|Options|Language|Delphi|Library**) - *Figure 80.3*.
3. Open the **Python4Delphi** package specific to the version of Delphi being used. For **Delphi 10.4.1** and later, use the package in the **Packages\Delphi\Delphi 10.4+** directory. For earlier versions including **Delphi 10.4** (version before **10.4.1**) use the package in the **Packages\Delphi\Delphi 10.3-** directory - *Figure 80.4*. This will need editing for **Delphi 10.4**. Remove all the **{IFDEF}{LIBSUFFIX}{ENDIF}** statements and replace with **{LIBSUFFIX '270'}**. *Figure 80.5* shows the **Projects** pane in the Delphi IDE after opening the package for **Delphi 10.4**.

Name	Date modified	Type
Demos	27/09/2020 18:05	File folder
ModifiedDemos	28/09/2020 01:52	File folder
Modules	27/09/2020 18:05	File folder
Packages	27/09/2020 18:05	File folder
Source	27/09/2020 18:05	File folder
Tests	27/09/2020 18:05	File folder
Unsupported	27/09/2020 18:05	File folder
.DS_Store	28/09/2020 01:50	DS_STORE File
.gitignore	25/09/2020 09:36	GITIGNORE File
LICENSE	25/09/2020 09:36	File
README.md	25/09/2020 09:36	MD File

Figure 80.2 python4delphi_master folder contents

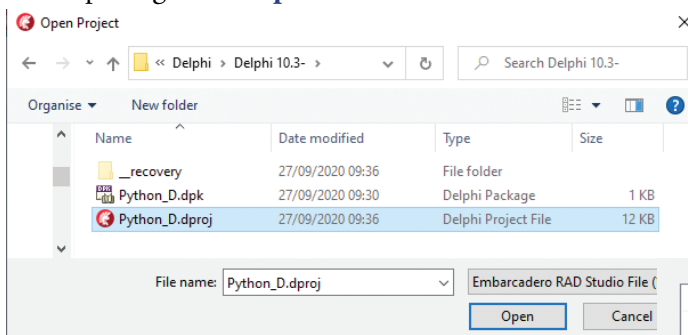


Figure 80.4 Open Project Python_D.dproj

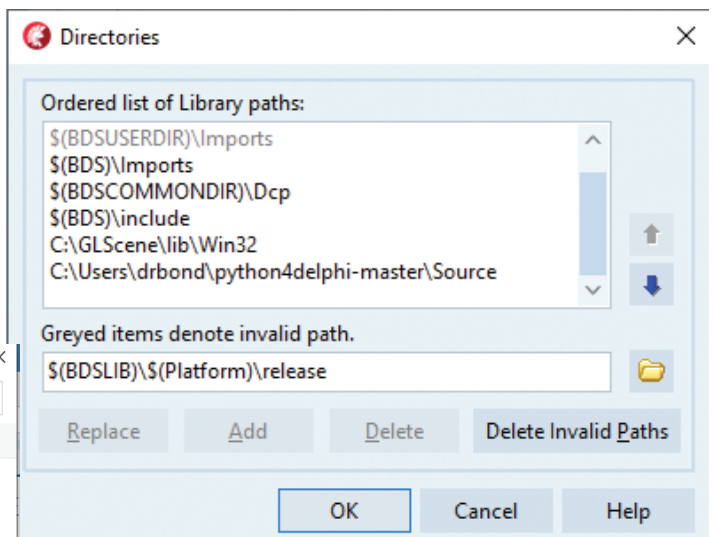


Figure 80.3 Setting Library path

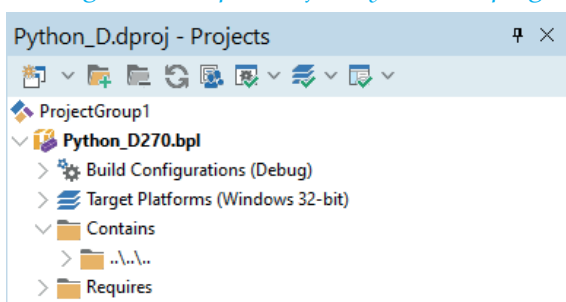


Figure 80.5 Projects pane after opening Python_D.dproj

4. Python was installed as a part of an Anaconda distribution (32-bit), **Anaconda3**, but Python can also be installed directly from [Python.org](https://python.org). You must also add the **\Library\bin** path of the Anaconda distribution to the system environment variable - *Figure 80.6*.

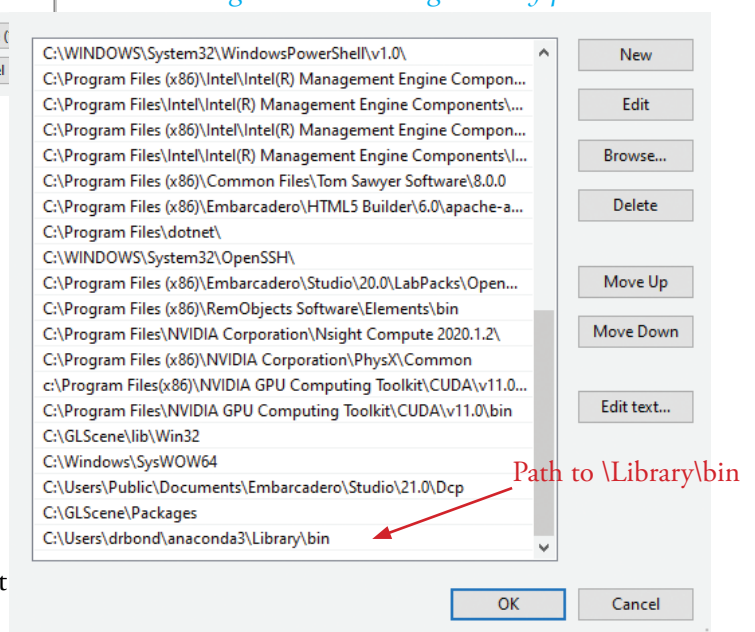


Figure 80.6 System environment variable

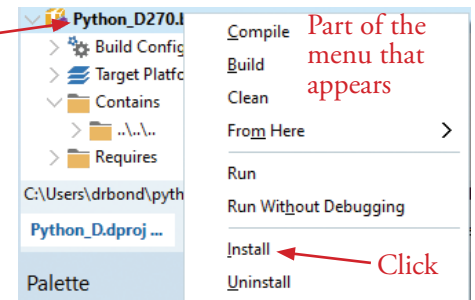
- Right click on **Python_D270.bpl** to bring up the menu shown in [Figure 80.7](#). Click on **Install** to install the python components. If successful, the component palette should now contain a **Python palette** of components as shown in [Figure 80.8](#).

First project

- Create a new **Windows VCL Application for Delphi**.
- Drop a **TPanel** component on the form and set its **Align** property to **alTop** in the **Object Inspector**. Change its **Caption** property to **Python Source Code**.
- Drop a **TMemo** component (**Memo1**) on the form and set its **Align** property to **alTop**. Open its **Lines** property and delete **Memo1** string to leave the memo box empty. Set **ScrollBars** to **ssBoth**.
- Drop a **TSplitter** component from the **Additional** palette on the form and set its **Align** property to **alTop** and its **Height** property to 2.
- Drop a **TPanel** component on the form and set its **Align** property to **alTop** in the **Object Inspector**. Change its **Caption** property to **Python Output**.
- Drop a **TPanel** component on the form and set its **Align** property to **alBottom** in the **Object Inspector**. Empty its **Caption** property.
- Drop a **TMemo** component (**Memo2**) on the form and set its **Align** property to **alClient** so that it fills the gap between the bottom panel and the **TSplitter** component. Open its **Lines** property and delete **Memo2** string to leave the memo box empty. Set **ScrollBars** to **ssBoth**.
- Select the bottom panel and then drop a **TButton** component (**Button1**) onto this panel. Set its caption to **Execute**. Set its **Anchors** property to **[akTop,akRight,akBottom]**.

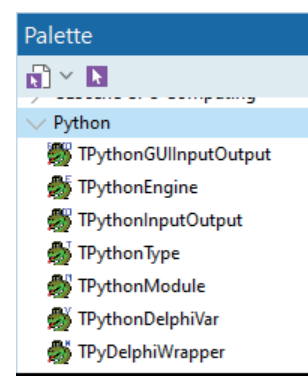
The user interface's appearance in the design window should now be as shown in [Figure 80.9](#).

Right mouse click
Python_D270.Bpl

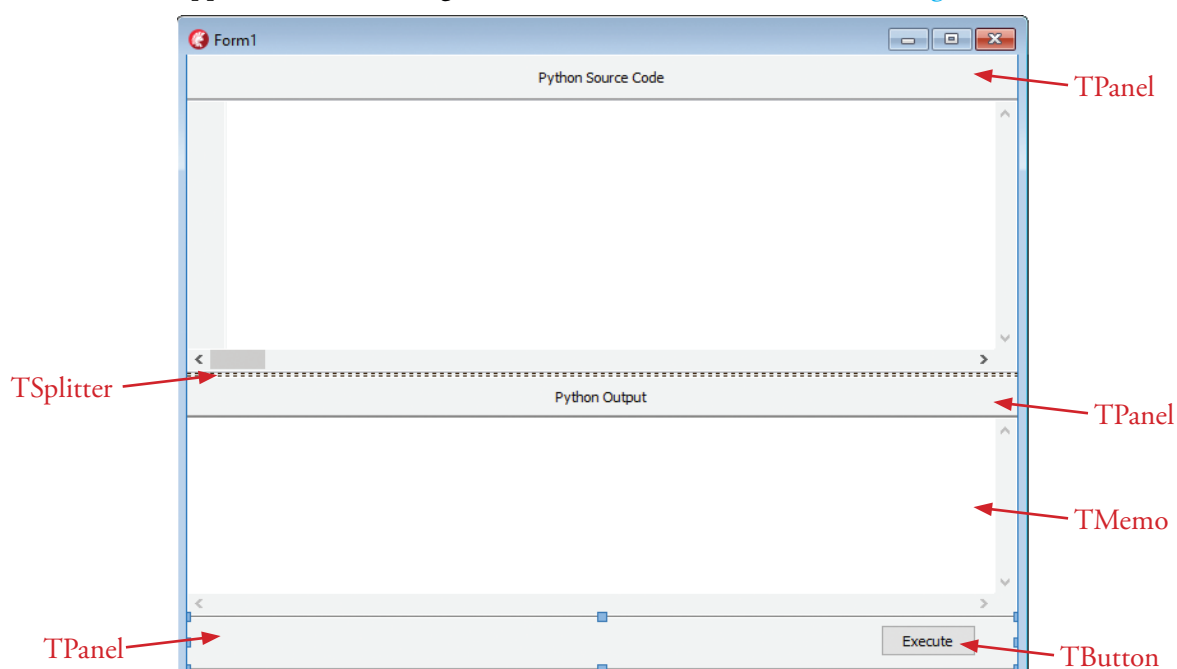


Part of the
menu that
appears

[Figure 80.7](#) Install components from here



[Figure 80.8](#) Component palette showing installed Python components



[Figure 80.9](#) Design of the user interface

9. Drop a **TPythonEngine** component from the **Python** palette on the form. This component provides the connection to **Python** or rather the **Python API**. Its default name is **PythonEngine1**.
10. Drop a **TPythonGUIInputOutput** component on the form. Its default name is **PythonGUIInputOutput1**. This component provides a conduit for routing input and output between the Graphical User Interface (GUI) and the currently executing Python script.
11. As this project uses an **Anaconda** distribution (see later for the same project implemented with Python from www.Python.org) **PythonEngine1**'s properties are required to be set up as follows
 - Set property **DllName** to `python38.dll` (because **Python 3.8** is installed) or the version that you have installed.
 - Set property **DllPath** to the Anaconda distribution, e.g. `c:\users\drbond\anaconda3`.
 - Set property **AutoLoad** to `False`.
 - Set **UseLastKnownVersion** to `False`.
 - Set **PythonEngine1**'s property **IO** to **PythonGUIInputOutput1**.
12. Set **PythonGUIInputOutput1**'s property **Output** to **Memo2**.
13. As this Delphi application relies on an Anaconda distribution, we are required to set up a **Form Create** event handler with the following two lines of code (this is unnecessary with Python from Python.org)

```

Procedure TForm1.FormCreate(Sender: TObject);
Begin
    PythonEngine1.SetPythonHome('c:\drbond\users\anaconda3');
    PythonEngine1.LoadDll;
End;

```

14. Double click **Button1** and add the line of code to the event handler


```
PythonEngine1.ExecStrings(Memo1.Lines);
```
15. Save the project and its unit in folder `FirstExample`. Name project `FirstExampleProject` and its unit `FirstExampleUnit`.
16. Now compile, link and run (**F9**) the executable.
17. Write `print(2 + 2)` in the **Python Source Code** window then click **Execute**. *Figure 80.10* shows the result. The executable calls up the **Python** interpreter which executes the `print` function with the given argument `2 + 2` returning the result `4`. This result is passed to the Delphi executable which then displays it in the **Python Output** window.

The **TSplitter** component allows the sizes of the two windows to be adjusted together so that as one is enlarged the other is reduced in size accordingly seamlessly - *Figure 80.11*.

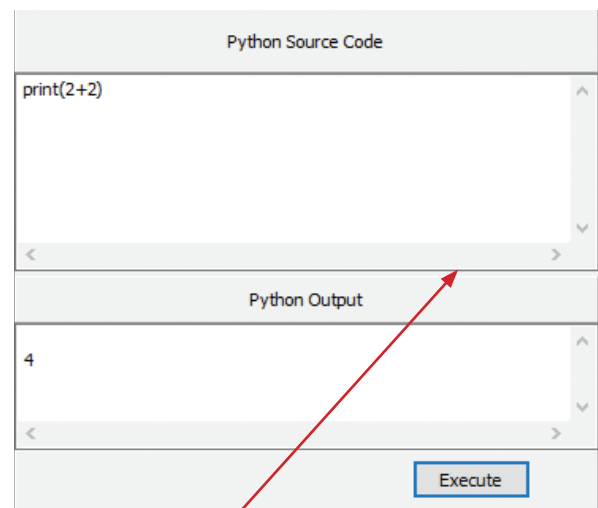


Figure 80.10 FirstExampleProject.exe in execution

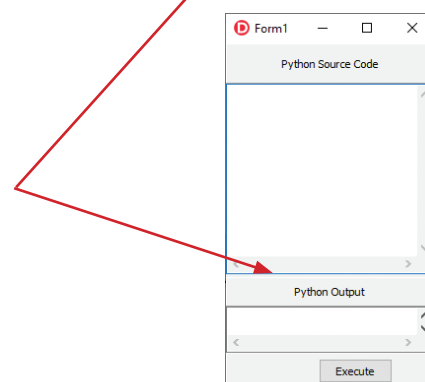


Figure 80.11

HOW TO PROGRAM EFFECTIVELY IN DELPHI

In order to be able to load and save Python scripts for execution by `FirstExampleProject.exe`, two more **TButton** components need to be added to the bottom panel as shown in *Figure 80.12*.

18. Select the bottom panel then add two **TButton** components to this panel, **Button2** and **Button3**. Set the **Caption** property of **Button2** to **Save** and the **Caption** property of **Button3** to **Load**. Position these two buttons as shown in *Figure 80.12*.
19. Set the **Anchors** property of **Button3** to **[akLeft, akTop, akBottom]**.
20. Set the **Anchors** property of **Button2** to **[akTop, akBottom]**.

When the application's window is resized, **Button3** (**Load**) will remain anchored at the same distance from the left, top and bottom edges of the bottom panel. **Button 2** (**Save**) will remain anchored at the same distance from the top and bottom edges of the bottom panel but it will move to the right or to the left, respectively, if the application's window is resized in either of these directions. **Button1** (**Execute**) will remain anchored at the same distance from the right, top and bottom edges of the bottom panel.

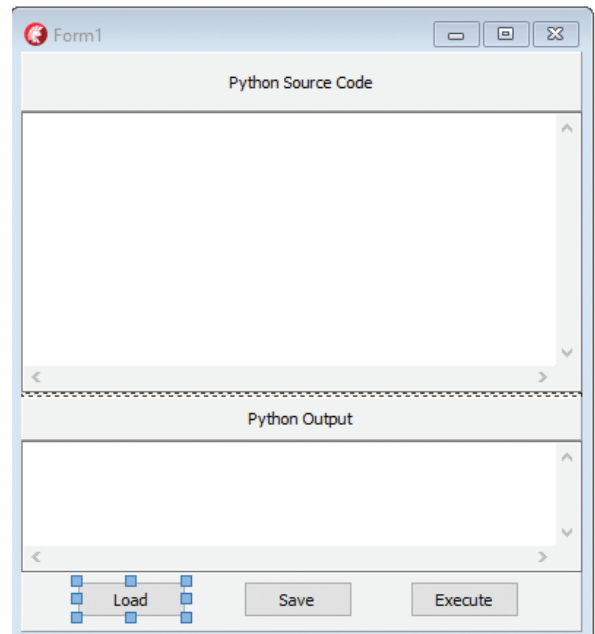


Figure 80.12 FirstExampleProject with two additional buttons Open and Save

21. Add a **TOpenDialog** and a **TSaveDialog** to the form.
22. Double click **Button2** (**Save**) and add the following lines of code to the event handler

```
With SaveDialog1
Do
  Begin
    If Execute
      Then Memo1.Lines.SaveToFile (FileName) ;
    End;
```

Property of SaveDialog1

23. Double click **Button3** (**Load**) and add the following lines of code to the event handler

```
With OpenDialog1
Do
  Begin
    If Execute
      Then Memo1.Lines.LoadFromFile (FileName) ;
    End;
```

Property of OpenDialog1

24. Save project and unit in folder `FirstExample`.
25. Now compile, link and run (**F9**) the executable.
26. Click **Load** and select `BubbleSort.py`.
27. Click **Execute**.

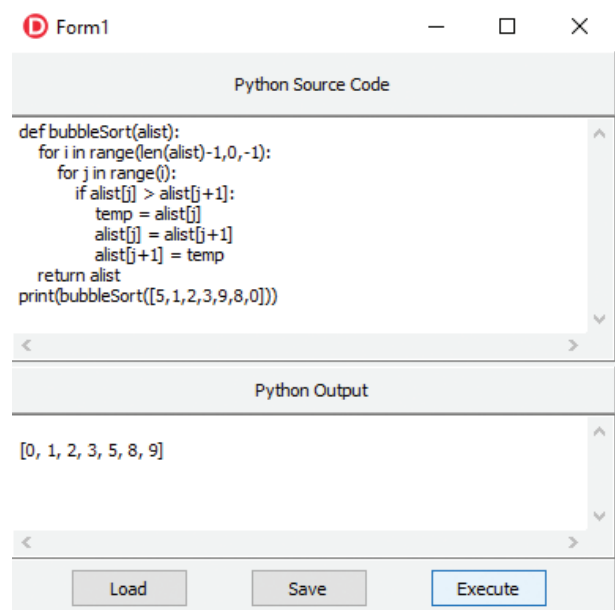


Figure 80.13 FirstExampleProject.exe in execution and with Python script BubbleSort.py loaded

Figure 80.13 shows the result.

www.educational-computing.com/DelphiBook/Code/Chapter80/FirstExample.zip

Second project

Save project `FirstExampleProject.dproj` in a new folder, `SecondExample`, rename the project `SecondExampleProject.dproj` (**Save Project As**).

Save unit `FirstExampleUnit.pas` in new folder `SecondExample`, rename the unit `SecondExampleUnit.pas` (**Save As**).

1. Select bottom panel and add a **TButton** component to this panel. Change its **Caption** property to `Show Var`. Position as shown in [Figure 80.14](#).
2. Add a **TPythonDelphiVar** component from the **Python** palette. Change its **VarName** property to `test`. Set its **Engine** property to **PythonEngine1** if this doesn't happen automatically.
3. Double click `Show Var` and add the following line of code to the event handler

```
ShowMessage('Value = ' +  
PythonDelphiVar1.ValueAsString);
```

4. Save project and unit.
5. Now compile, link and run (**F9**) the executable.
6. Enter the following **Python** code into the **Python Source Code** window
`test.value = 5`
`print(test, test.value)`
as shown in [Figure 80.15](#).

7. Click **Execute**. The line `<PythonDelphiVar: 5> 5` appears in the **Python Output** window. Python function `print` copies the numeric value 5 stored in the Python variable `test.value` and then sends it to Delphi to handle via **PythonGUIInputOutput1** which is also connected to **Memo2**.
8. Now click `Show Var` to execute some Delphi code. The window shown in [Figure 80.15](#) should appear displaying the string `'Value = 5'`. This is occurs because the identifiers `PythonDelphiVar1` and `test` are aliases for the same variable.

Information

The identifiers `test` and `PythonDelphiVar1` operate as aliases for the same variable. It makes better sense therefore to use the same identifier name for each. `PythonDelphiVar1` is the property value of property **Name** for a `TPythonDelphiVar` object and the other, `test`, is the property value of property **VarName**. The latter is the variable name that is accessible to a Python script.

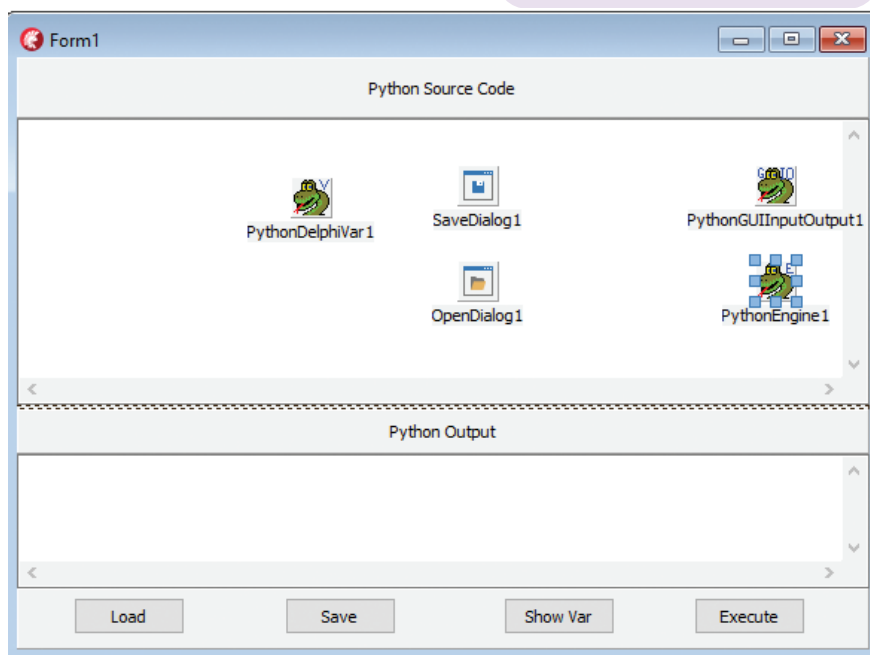


Figure 80.14 SecondExampleProject user interface design

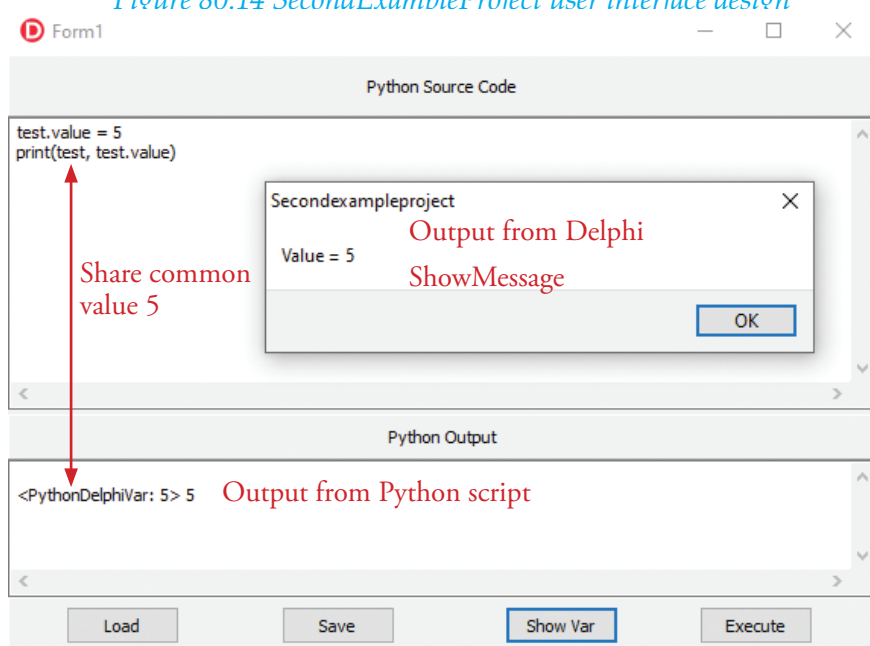


Figure 80.15 SecondExampleProject.exe in execution

The fact that these identify the same variable means it is possible to read and write the variable's content in both Python and Delphi!

Third project

Save project `SecondExampleProject.dproj` in a new folder, `ThirdExample`, renaming the project `ThirdExampleProject.dproj` (**Save Project As**).

Save unit `SecondExampleUnit.pas` in new folder `ThirdExample` renaming the unit `ThirdExampleUnit.pas` (**Save As**).

1. Select bottom panel and add a **TEdit** component, **Edit1**, to this panel as shown in [Figure 80.16](#). Set its **Text** property to 0.
2. Click on **PythonDelphiVar1** and in the **Object Inspector** switch to the **Events** tab.
3. Double-click on attribute **OnGetData** to create its event handler. Add the following line of code to this event handler: `Data := Edit1.Text;`
4. Double-click on attribute **OnSetData**, and add the following line of code to this event handler:

```
Edit1.Text := Data;
```

5. Double-click on the **OnChange** attribute, and add the following line of code to this event handler:

```
With Sender As TPythonDelphiVar
  Do ShowMessage('Var test changed: ' + PythonDelphiVar.ValueAsString);
```

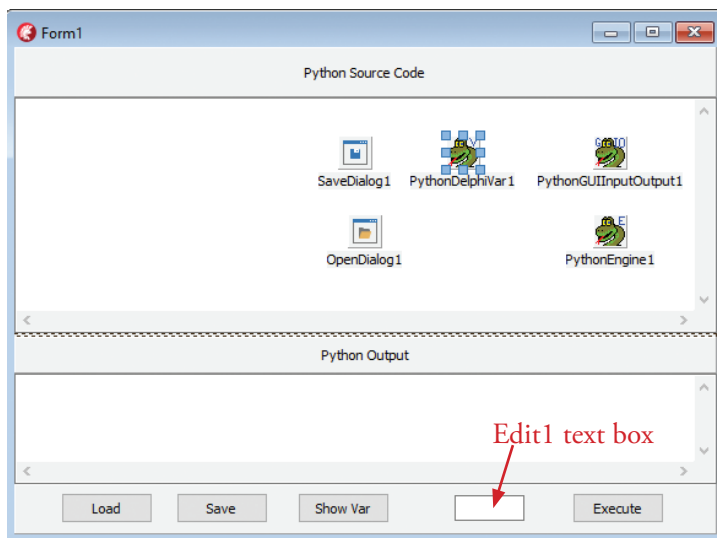


Figure 80.16 ThirdExampleProject user interface design

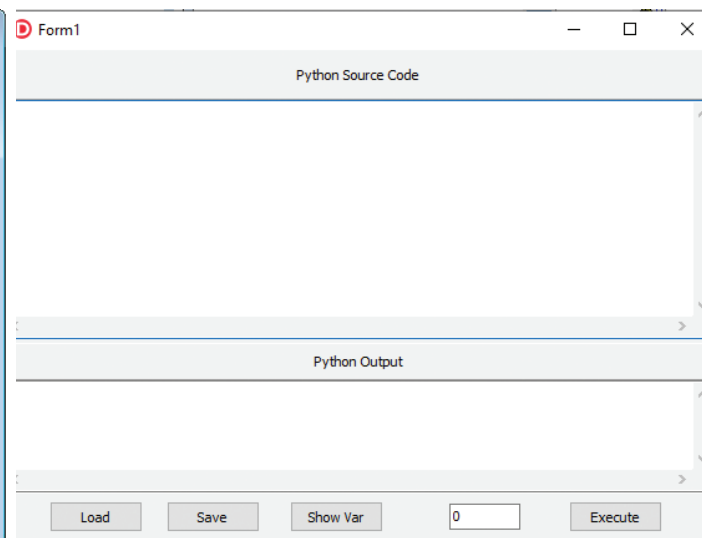


Figure 80.17 ThirdExampleProject just launched

6. Save project and unit.
7. Now compile, link and run (F9) the executable - [Figure 80.17](#).
8. Write the Python script `test.value=45` in the **Python Source Code** window.
9. Click **Execute**. The value shown in the text box **Edit1** will change from its default value 0 to new value 45 as a consequence of executing the given Python script. The outcome demonstrates that Python variable `test` is associated with the variable `Edit1` in the Delphi program.

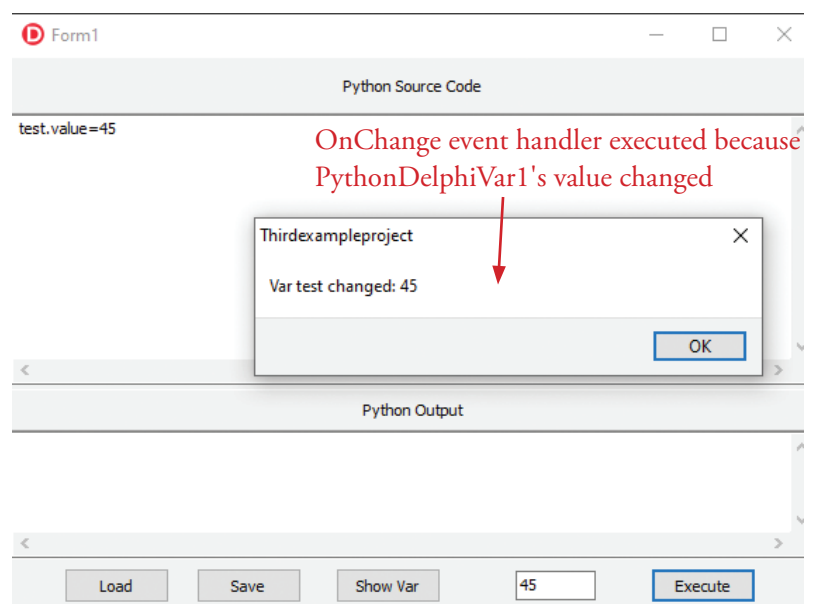


Figure 80.18 ThirdExampleProject executing script `test.value=45`

10. Click **OK** to close the popup window.
11. Delete `test.value=45` in the `Python Source Code` window.
12. Write `print(test.value)` in the blank `Python Source Code` window
13. Change the contents of text box `Edit1` to 30.
14. Click **Execute**. The value 30 now appears in the `Python Output` window reflecting the fact that the value assigned to Python variable `test` has been updated to the value of Delphi text box, `Edit1`.

This exercise has demonstrated that it is possible using components from the Python palette to create two-way communication between a Delphi program in execution and a Python script in execution.

Fourth project

1. Create a new **Windows VCL Application for Delphi**.
2. Drop a **TPanel** component on the form and set its **Align** property to **alTop** in the **Object Inspector**. Change its **Caption** property to `Python Source Code`.
3. Drop a **TPanel** component on the form and set its **Align** property to **alBottom** in the **Object Inspector**. Empty its **Caption** property.
4. Select the bottom panel and then drop a **TButton** component (`Button1`) onto this panel. Set its caption to **Execute**. Set its **Anchors** property to `[akTop,akRight, akBottom]`.
5. Drop a **TMemo** component (`Memo1`) on the form and set its **Align** property to **alClient**. Open its **Lines** property and delete `Memo1` string to leave the memo box empty. Set **ScrollBars** to **ssBoth**.
6. Drop a **TPythonEngine** component from the **Python** palette on the form. This component provides the connection to **Python** or rather the **Python API**. Its default name is **PythonEngine1**.
7. Drop a **TPythonInputOutput** component on the form. Its default name is **PythonInputOutput1**. This component provides a conduit for routing input and output between a console window and the currently executing Python script.

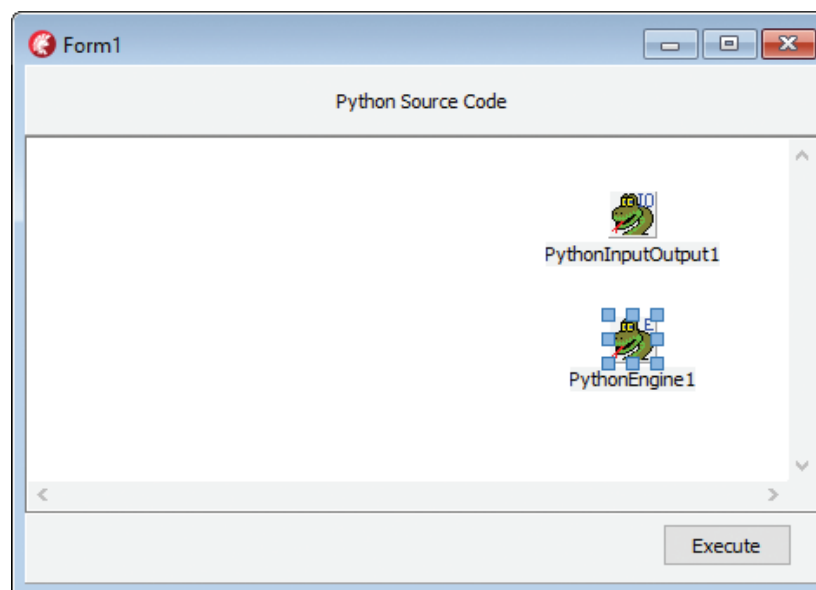


Figure 80.19 FourthExampleProject GUI design

The user interface's appearance in the design window should now be as shown in [Figure 80.19](#). As this project uses an **Anaconda** distribution (see later for the same project implemented with Python from www.Python.org)

PythonEngine1's properties are required to be set up as follows

- Set property **DllName** to `python38.dll` (because **Python 3.8** is installed) or the version that you have installed).
 - Set property **DllPath** to the Anaconda distribution, e.g. `c:\users\drbond\anaconda3`.
 - Set property **AutoLoad** to `False`.
 - Set **UseLastKnownVersion** to `False`.
 - Set **PythonEngine1**'s property **IO** to **PythonInputOutput1**.
8. Select **PythonInputOutput1** and its **Events** page. Double click **OnReceiveData** to create event handler **PythonInputOutputReceiveData**.
 9. Select **PythonInputOutput1** and its **Events** page. Double click **OnSendData** to create event handler **PythonInputOutputSendData**.

10. Add the line of code to **PythonInputOutputReceiveData**

```
Readln(Data);
```

11. Add the line of code to **PythonInputOutputSendData**

```
Writeln(Data);
```

12. In order to read from and write to the console we need to create a console at run time. Click **Project|View Source** to bring up the application's source code. Add the following directive to the application's source code as shown in [Figure 80.20](#)

```
Program FourthExampleProject;
{$APPTYPE CONSOLE} ← Creates Console window
Uses
  Vcl.Forms,
  FourthExampleUnit in 'FourthExampleUnit.pas' {Form1};

{$R *.res}

Begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TForm1, Form1); ← Creates GUI window
  Application.Run;
End.
```

Figure 80.20 Program FourthExampleProject

```
{ $APPTYPE CONSOLE }
```

13. Save the project using filename `FourthExampleProject.dproj` (**Save Project As**) in a folder `FourthExample`.
14. Save its unit using filename `FourthExampleUnit.pas` in the same folder `FourthExample`.

15. Now compile, link and run (**F9**) the executable. Two windows should appear: a console window and an application window as shown in *Figure 80.21*.

16. Enter the following Python script in the Python Source Code window and click **Execute**

```
value = input("Please enter a string\n")
print(f'You entered {value}')
```

17. The prompt **Please enter a string** should appear in the console window as shown in *Figure 80.21*.

18. Click the console window and enter **Hello World!** then press return.

The string **You entered Hello world!** shown in *Figure 80.21* should be echoed to the console window.

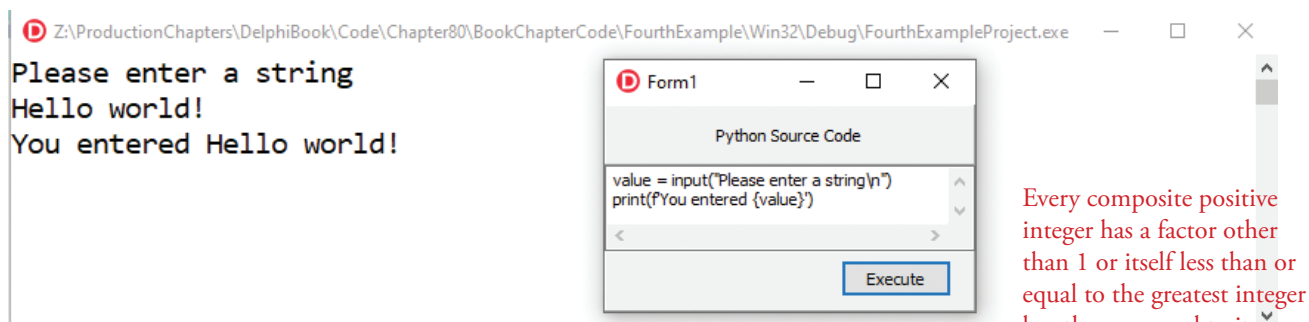


Figure 80.21 FourthExampleProject in execution

www.educational-computing.com/DelphiBook/Code/Chapter80/FifthExample.zip

Fifth project - this project was inspired by Kiriakos Vlahos (aka PyScripter)

Open project **FirstExampleProject.dproj** and save in a new folder,

FifthExample, whilst renaming it **FifthExampleProject.dproj** (**Save Project As**).

Save unit **FirstExampleUnit.pas**

in folder **FifthExample** renaming it

FifthExampleUnit.pas (**Save As**).

1. Add a **TPythonModule** to the form (the form should already contain a **TPythonEngine** and a **TPythonGUIInputOutput** component).

2. Add the function **IsPrime** shown in *Figure 80.22* to the implementation section of **FifthExampleUnit.pas**.

3. Select **PythonModule1** and in the Object Inspector click the ellipsis (*Figure 80.23*) to bring up the **Events editor** shown in *Figure 80.24*.

```
Function IsPrime(No : Integer) : Boolean;
Begin
  If (No <= 1) Then Exit(False);
  Var UpperLimit := Floor(Sqrt(No));
  For Var i := 2 To UpperLimit
    Do If (No Mod i = 0) Then Exit(False);
  Exit(True);
End;
```

Figure 80.22 Function IsPrime

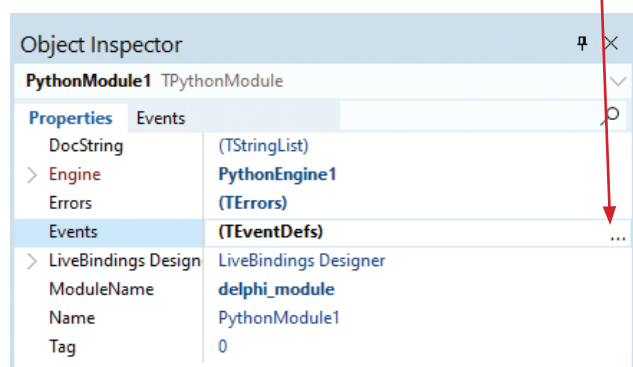


Figure 80.23 Events property PythonModule1

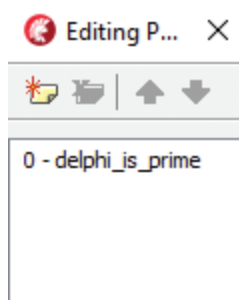


Figure 80.24 delphi_is_prime event

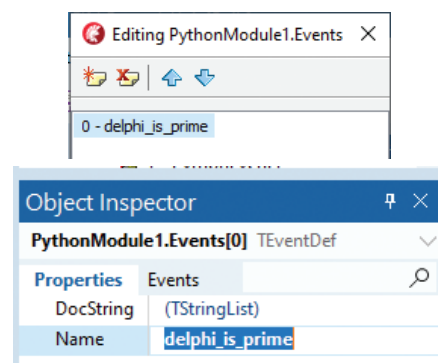

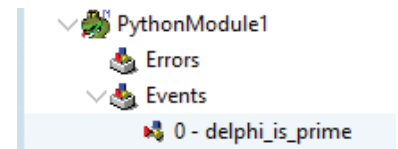
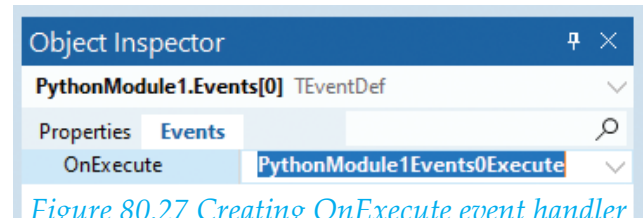


Figure 80.25 delphi_is_prime event

4. Click  in the **Events editor** to add a new event named `delphi_is_prime`. [Figure 80.24](#) already shows that a new event has already been added and then named `delphi_is_prime`. This naming is done by highlighting the default name of this new event in the **Events editor** and then in the Object Inspector, changing the events **Name** property to `delphi_is_prime` - [Figure 80.25](#).
5. In the **Structure pane** select **Events** and `0 - delphi_is_prime` - [Figure 80.26](#).
6. Switch to the **Events** page for `PythonModule1`. `Events[0]` and double click the field for the **OnExecute** event to create the event handler `PythonModule1Events0Execute` - [Figure 80.27](#).
7. Switch to the source code view of `FifthExampleUnit.pas` and add the lines of source code shown in [Figure 80.28](#) to the event handler `PythonModule1Events0Execute`.
8. Select **PythonModule1** in the Object Inspector and set its **ModuleName** property to `delphi_module`.
9. Save all (**Shift+Ctrl+S**).
10. Now compile, link and run (**Shift+Ctrl+F9**) the executable.



[Figure 80.26](#)



[Figure 80.27](#) Creating OnExecute event handler

```

Procedure TForm1.PythonModule1Events0Execute(Sender: TObject; PSelf,
                                           Args: PPyObject;
                                           Var Result: PPyObject);

Var
  N : Integer;
Begin
  With GetPythonEngine
    Do
      Begin
        If PyArg_ParseTuple(Args, 'i:delphi_is_prime', @N) <> 0
          Then
            Begin
              If IsPrime(N)
                Then Result := PPyObject(Py_True)
                Else Result := PPyObject(Py_False);
              Py_INCREF(Result);
            End
          Else Result := Nil;
        End;
      End;
  End;
End;

```

[Figure 80.28](#) Event handler `PythonModule1Events0Execute`

The following exercise was inspired by a Webinar given by Kiriakos Vlahos (aka PyScripter):

11. Click **Load** and open `countofprimesinpythononly.py` ([Figure 80.29](#)).
12. Click **Execute** to run `countofprimesinpythononly.py`. Repeat this two more times.
13. The Python program counts the number of primes between 0 and 1000000, measures the time that elapses and then prints both results - [Figure 80.30](#). The elapsed time is approximately 6.5 seconds.
14. Click **Load** and open `pythondelphi_prime.py` - see [Figure 80.31](#). This script uses a function, `delphi_is_prime`, linked to `PythonModule1` via module `delphi_module`, to test for primality in place of the Python `is_prime` function in `countofprimesinpythononly.py`. [Figure 80.28](#) shows that `delphi_is_prime` in turn relies on function `IsPrime` written in Delphi. When the python script is run it invokes the event handler `PythonModule1Events0Execute` which in turn calls the Delphi function `IsPrime`.

```

from timeit import Timer
import math
def is_prime(n):
    if n <= 1:
        return False
    upperlimit = math.floor(math.sqrt(n))
    for i in range(2, upperlimit + 1):
        if (n % i == 0):
            return False
    return True
def count_primes(max_n):
    result = 0
    for i in range(2, max_n + 1):
        if is_prime(i):
            result += 1
    return result
def test():
    max_n= 1000000
    print(f'Numberof primes between 0 and {max_n} = {count_primes(max_n)}')
def main():
    print(f'Elapsedtime: {Timer(stmt=test).timeit(1)} secs')
if __name__ == '__main__':
    main()

```

Figure 80.29 countofprimesinpythononly.py

PythonEngine1 and PythonModule1

provide the "wiring" between the Python script and the Delphi event handler

`PythonModule1Events0Execute`. The

remainder of the code of the latter packs and unpacks the communication between Python script and the Delphi program.

15. Click `Execute` to run `pythondelhiprime.py`. Repeat this two more times.
16. The Python program counts the number of primes between 0 and 1000000, measures the time that elapses and then prints both results - [Figure 80.32](#). The elapsed time is approximately 0.32 seconds. Twenty times faster than script `countofprimesinpythononly.py`.

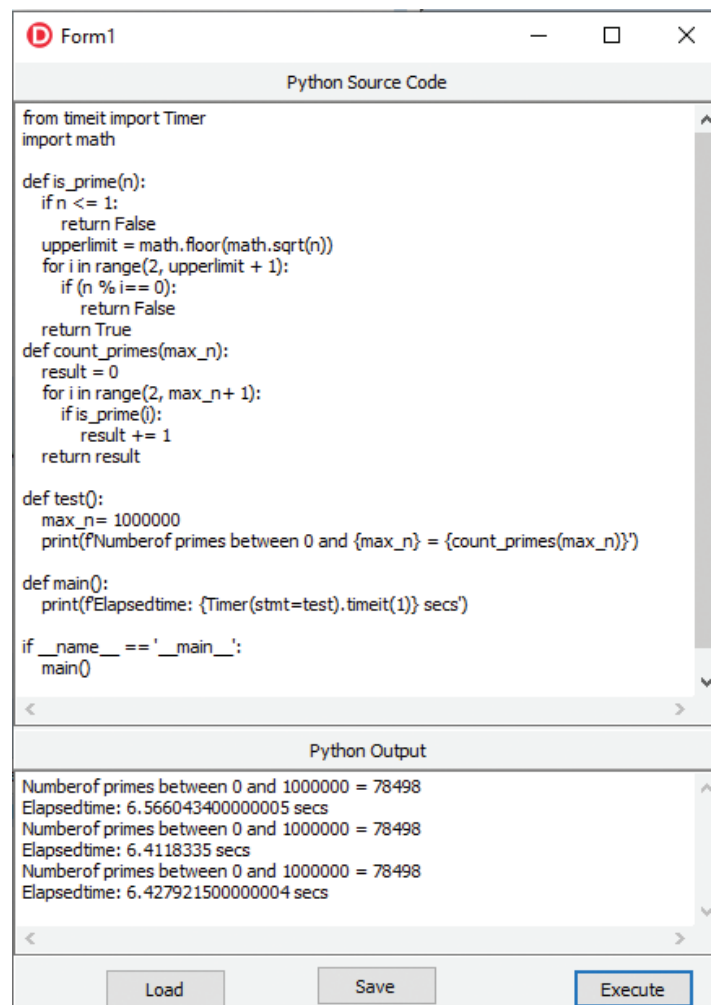


Figure 80.30 countofprimesinpythononly.py executed three times

```

from delphi_module import delphi_is_prime
from timeit import Timer
import math

def count_primes(max_n):
    result=0
    for i in range(2,max_n+1):
        if delphi_is_prime(i):
            result +=1
    return result
def test():
    max_n=1000000
    print(f'Number of primes between 0 and {max_n}={count_primes(max_n)}')
def main():
    print(f'Elapsed time:{Timer(stmt=test).timeit(1)}secs')
if __name__=='__main__':
    main()

```

Information

Threading issue in Python


Python is not suitable for parallelising computationally intensive Python code because threaded Python code is locked to one thread executing at a time. If C extensions and I/O, however (e.g. PIL or numpy operations) and any C code can run in parallel with one active Python thread. One solution is to delegate to a dedicated external library.

Figure 80.31 *pythondelhiprime.py*

Sixth project

www.educational-computing.com/DelphiBook/Code/Chapter80/SixthExample.zip

This project was inspired by Kiriakos Vlahos (aka PyScripter):

1. Save project `FifthExampleProject.dproj` in a new folder, `SixthExample`, renaming it `SixthExampleProject.dproj` (**Save Project As**). Save unit `FirstExampleUnit.pas` in folder `SixthExample` renaming it `SixthExampleUnit.pas` (**Save As**).
2. Select **PythonModule1** and in the Object Inspector click the ellipsis (*Figure 80.23*) to bring up the **Events editor**.
3. Click  in the **Events editor** to add a new event `delphi_count_primes` - *Figure 80.33*.
4. In the **Structure pane** select **Events** and `1 - delphi_count_primes`.
5. Switch to the **Events** page for `PythonModule1`. `Events[1]` and double click the field for the **OnExecute** event to create the event handler `PythonModule1Events1Execute`.
6. Switch to the source code view of `SixthExampleUnit.pas` and add the function `CountPrimes` - *Figure 80.34*.
7. Add the lines of source code shown in *Figure 80.35* to the event handler `PythonModule1Events1Execute`.
8. Save all (**Shift+Ctrl+S**).
9. Now compile, link and run (**Shift+Ctrl+F9**) the executable.
10. Click **Load** and open `CountToPrimesParallel.py` (*Figure 80.36*).

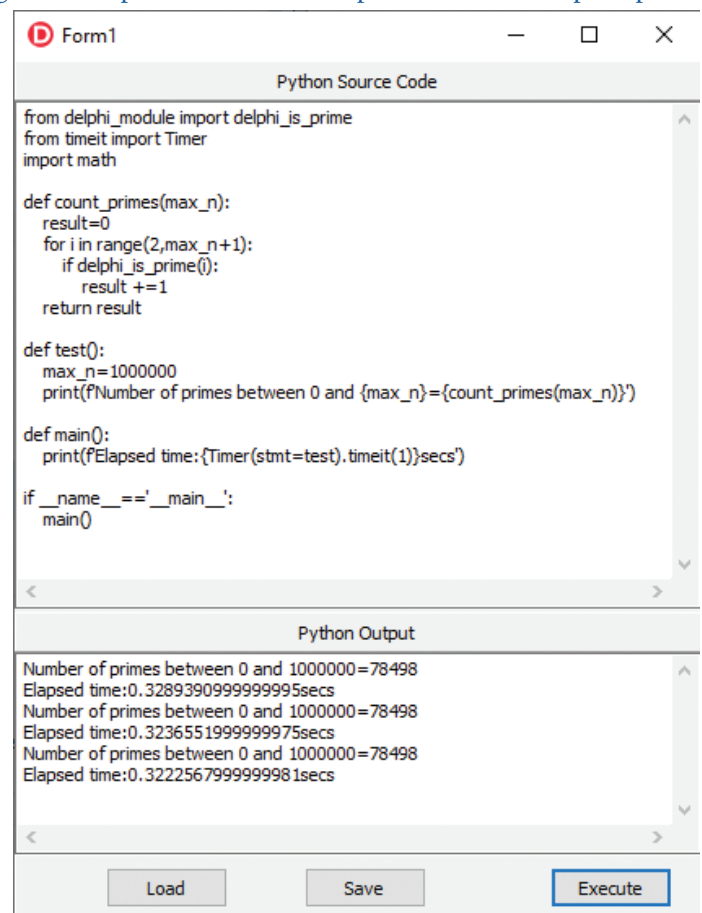


Figure 80.32 *pythondelhiprime.py* executed three times

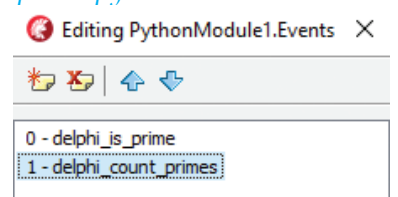


Figure 80.33 Events editor, new event `delphi_count_primes`

11. Click **Execute** to run `CountToPrimesParallel.py`. Repeat this two more times.
12. The Python program counts the number of primes between 0 and 1000000, measures the time that elapses and then prints both results - [Figure 80.37](#). The elapsed time is approximately 0.06 seconds which is a 100 times faster than the python script `countofprimesinpythononly.py`.

TParallel provides a class for-loop, **&For**, which makes efficient use of all the CPU cores in the system.

```
Function CountPrimes(MaxN : Integer): Integer;
Begin
    Var Count := 0;
    TParallel.&For(2, MaxN, Procedure(i : Integer)
        Begin
            If IsPrime(i)
            Then AtomicIncrement(Count);
            End
        );
    Result := Count;
End;
```

Figure 80.34 Function CountPrimes

```
Procedure TForm1.PythonModule1Events1Execute(Sender: TObject; PSelf,
    Args: PPyObject;
    Var Result: PPyObject);

Var
    N : Integer;
Begin
    With GetPythonEngine
    Do
        Begin
            If PyArg_ParseTuple(Args, 'i:delphi_count_primes', @N) <> 0
            Then
                Begin
                    Result := PyLong_FromLong(CountPrimes(N));
                    Py_INCREF(Result);
                End
            Else Result := Nil;
        End;
    End;
End;
```

Figure 80.35 Event handler PythonModule1Events1Execute

```
from delphi_module import delphi_count_primes
from timeit import Timer
import math
def test():
    max_n = 1000000
    print(f'Number of primes between 0 and {max_n} = {delphi_count_primes(max_n)}')
def main():
    print(f'Elapsed time:{Timer(stmt=test).timeit(1)}secs')
if __name__=='__main__':
    main()
```

Figure 80.36 CountPrimesParallel.py

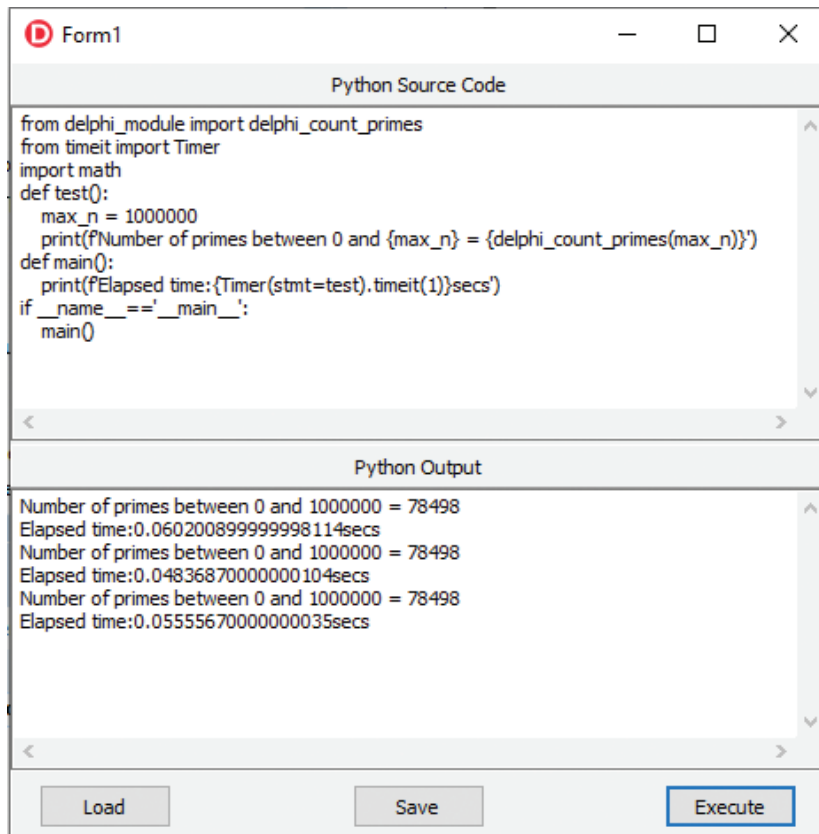


Figure 80.37 CountPrimesParallel.py

Stable Releases

Python 3.9.0 - Oct. 5, 2020

Note that Python 3.9.0 cannot be used on Windows 7 or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)

Figure 80.38 Python 3.9 download versions

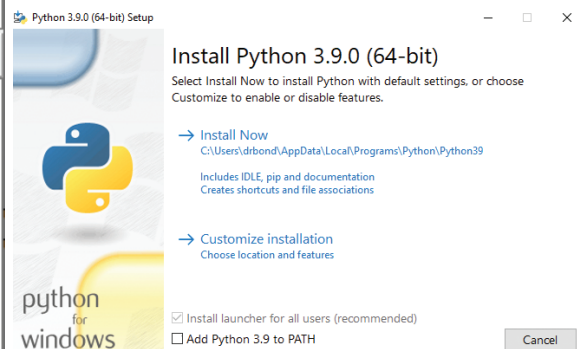


Figure 80.39 Python 3.9.0(64-bit) install

Using Python4Delphi with a version of Python installed from Python.org

Download **Windows x86-64 executable installer** for Python from python.org/downloads/windows - Figure 80.38.

Click on **python-3.9.0-amd64.exe** (or the version that you downloaded) to launch the install window. Tick **Add Python 3.9 to PATH** then click **Install Now** to install **Python 3.9** - Figure 80.39.

Python4Delphi should find **Python39.dll** automatically.

Create a new VCL application and add a **TPythonEngine** component to the form. Figure 80.40 shows the Object Inspector pane for **PythonEngine1**. For this version of **Python4Delphi** the default settings of interest for **DelphiPython1** are

- UseLastKnownVersion** = **True**
- DllName** = **python39.dll**
- AutoLoad** = **True**.

When earlier in this chapter we worked with an Anaconda installation of Python - **Python 3.8** we needed the application to initialise **PythonEngine1** as follows

```
PythonEngine1.SetPythonHome('c:\Users\drbond\anaconda3');
PythonEngine1.LoadDll;
```

Using **AutoLoad** set to **True** takes care of this automatically.

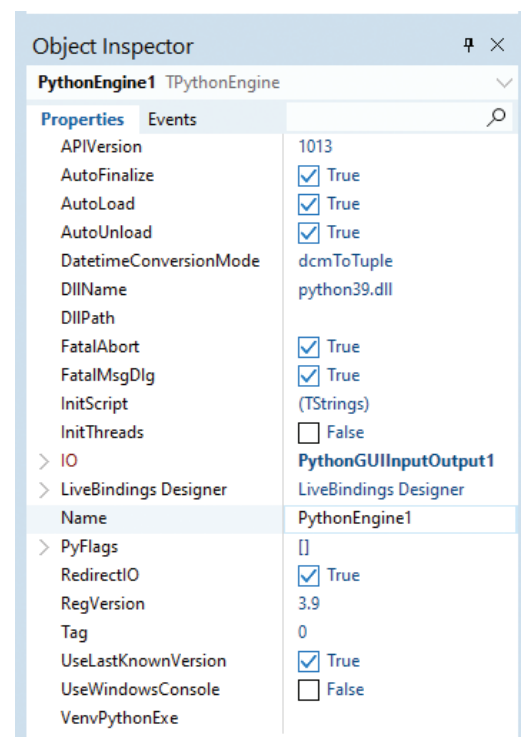


Figure 80.40 Object Inspector

When Python is installed in Windows, the user has the option to register it, either for all users or just for this user. Registration involves writing information to the registry about the location of the installation, the name and location of the help file etc.

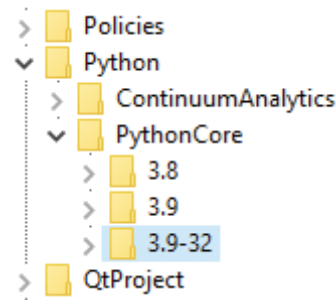
Setting **UseLastKnownVersion** = **True** forces Delphi to use the latest **registered** version of Python.

If the programmer requires a different **registered**² version then they need to set the following properties:

- **DLLName** e.g. python38.dll
- **RegVersion** e.g 3.8
- Set **UseLastKnownVersion** property to **False**.

Figure 80.41 Registry entry for

Computer\HKEY_CURRENT_USER\SOFTWARE\Python\PythonCore



A little more work needs to be done to use a specific **unregistered** version:

Set

- **DLLName** e.g. python38.dll
- **RegVersion** e.g 3.8
- Set **UseLastKnownVersion** property to **False**
- Set **DLLPath** to the path where the DLL is located, e.g. C:\Users\drbond\anaconda3
- Set **AutoLoad** property to False
- The event handler for the **OnCreate** event of **Form1** (or whatever the form launched by the application is called), or in another suitable place, must contain the following code (assuming that **PythonEngine1** is the name of the component)

```
PythonEngine1.SetPythonHome('Python installation directory');
PythonEngine1.LoadDLL;
```

32-bit Delphi applications only work with 32-versions of Python and 64-bit Delphi applications only work with 64-bit versions of Python.

The project **FirstExampleProject** created using the Anaconda installed Python may now be recreated Python 3.9 downloaded directly from www.python.org.

² Anaconda distributions require that **SetPythonHome** is called as shown above even if they are registered.

Seventh project

www.educational-computing.com/DelphiBook/Code/Chapter80/SeventhExample.zip

This project manipulates images in Python which are loaded, displayed and saved using Delphi.

The Python side of things requires a module called pillow (PIL). The easiest way to install pillow is automatically as an integral part of an Anaconda installation, installing it directly is quite tricky because it depends on other modules being pre-installed. Anaconda automatically takes care of everything. For this reason, the Anaconda installation of Python - Python 3.8 - is used in this project. It will therefore be necessary to set up the Python Engine as per the instructions for unregistered versions of Python - see previous page.

1. Create a new **Windows VCL Application for Delphi** and save in a new folder `SeventhExample` with filenames `SeventhExampleProject.dproj` and `SeventhExampleUnit.pas`.
2. Drop a **TPanel** component on the form and set its **Align** property to **alBottom** in the **Object Inspector**. Set its **Height** property to 41 and clear its **Caption** property.
3. With this panel selected drop three buttons on the panel and set the **Top** property of each to 8. Position the buttons as shown in *Figure 80.42*.
4. Label the buttons `Load Image`, `Save Image` and `Execute` according to *Figure 80.42* adjusting the width of each accordingly.
5. Set the **Anchors** property of `Load Image` to **[akLeft, akTop, akBottom]**.
6. Set the **Anchors** property of `Save Image` to **[akTop, akBottom]**.
7. Set the **Anchors** property of `Execute` to **[akTop, akRight, akBottom]**.
8. Drop a **TMemo** component (change **Name** to `Memo2`) on the form and set its **Align** property to **alBottom**. Open its **Lines** property and delete `Memo2` string to leave the memo box empty for the moment. Set **ScrollBars** to **ssBoth**.
9. Drop a **TSplitter** component from the **Additional** palette on the form and set its **Align** property to **alBottom** and its **Height** property to 2.
10. Drop a **TPanel** component on the form and set its **Align** property to **alBottom** in the **Object Inspector**. Change its **Caption** property to `Python Script`.
11. Drop a **TMemo** component (`Memo1`) on the form and set its **Align** property to **alBottom**. Open its **Lines** property and delete `Memo1` string to leave the memo box empty. Set **ScrollBars** to **ssBoth**.
12. Drop a **TPanel** component on the form and set its **Align** property to **alBottom** in the **Object Inspector**. Set its **Caption** property to `Image information (Don't exceed 2.5 MB)`.
13. Drop a **TImage** component set its **Align** property to **alClient** and its **Proportional** property to **True**.
14. Drop a **TPythonEngine** and a **TPythonGUIInputOutput** component on the form.

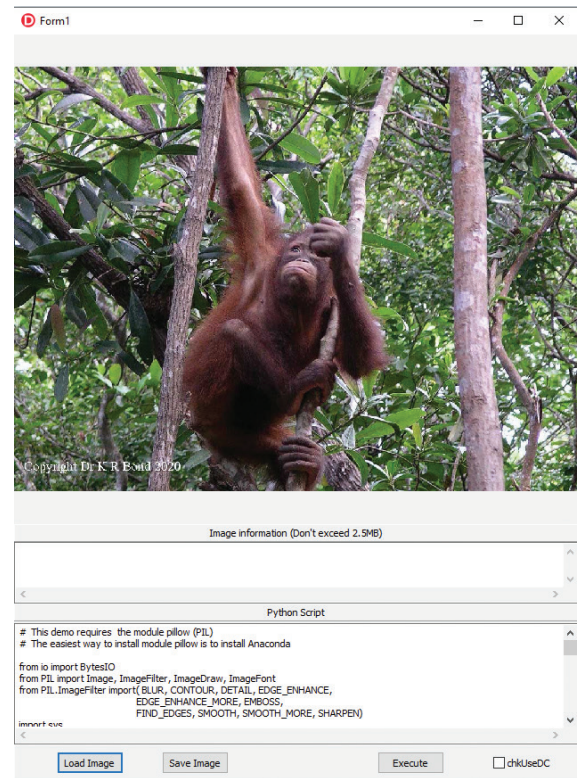


Figure 80.42 SeventhExampleProject in execution

15. As the author's Python installation was done via Anaconda, **PythonEngine1** needs to be set up as follows
- **DLLName** e.g. python38.dll
 - **RegVersion** e.g 3.8
 - Set **UseLastKnownVersion** property to **False**
 - Set **DLLPath** to the path where the DLL is located, e.g. C:\Users\drbond\anaconda3
 - Set **AutoLoad** property to False
 - The event handler for the **OnCreate** event of **Form1** (or whatever the form launched by the application is called), or in another suitable place, must contain the following code (assuming that **PythonEngine1** is the name of the component)

```
PythonEngine1.SetPythonHome('Python installation directory');
PythonEngine1.LoadDLL;
```

16. Set **PythonEngine1**'s **IO** property to **PythonGUIInputOutput1**.

17. Set **PythonGUIInputOutput1**'s **Output** property to Memo1.

18. Add a **TOpenDialog** and a **TSaveDialog** component to the form.

19. Double click **Save Image** button and add the following lines of code to the event handler

```
With SaveDialog1
Do
Begin
    If Execute
        Then Memo1.Lines.SaveToFile (FileName) ;
    End;
```

Property of SaveDialog1

20. Double click **Load Image** button and add the following lines of code to the event handler

```
With OpenFileDialog1
Do
Begin
    If Execute
        Then Memo1.Lines.LoadFromFile (FileName) ;
    End;
```

Property of OpenFileDialog1

21. Save project and unit in folder **SeventhExample**.

22. Select **Memo2** and open its **Lines** property so that the contents of script **ImageProcessingScript.py** shown in [Figure 80.45](#) can be pasted into the **String List Editor**.

23. Add a **Uses** clause to the **Implementation** section (place below **{R *.dfm}**) as follows

```
Uses
    VarPython,
    Math,
    jpeg;
```

24. Add the source code shown in [Figure 80.46](#) to the implementation section.

25. Save all (**Shift+Ctrl+S**).

26. Now compile, link and run (**Shift+Ctrl+F9**) the executable.

27. The form shown in [Figure 80.43](#) should show.

28. Use the splitter bar to enlarge the Python script window.

29. The default setting of Python script is `new_im = im.convert('L')`. This Python statement when executed converts an RGB image to a black and white image one.

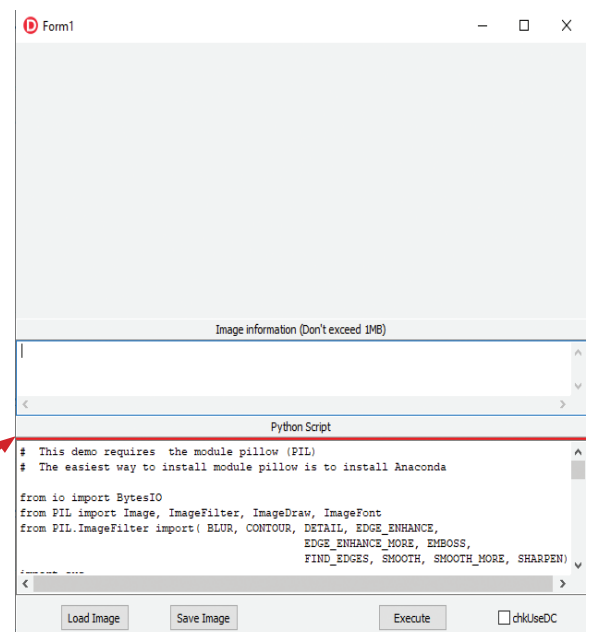


Figure 80.43 SeventhExampleProject in execution

30. Click **Load Image** and select from the images folder `BorneoOrangUtan1SmallRes1.jpg` and open - *Figure 80.42*.
31. Click **Execute** to convert the image to a black and white one - *Figure 80.44*.
32. Comment out `new_im = im.convert('L')` in the Python Script window and uncomment
`new_im = im.rotate(90, expand=True).`
33. Click **Load Image** and select from the images folder `FarmPortDicksonLowResSmall.jpg` and open.
34. Click **Execute** to rotate this image through 90° counterclockwise. *Figure 80.47* shows the rotated image.
35. Click **Load Image** and select from the images folder `CockatooSmallLowRes.jpg` and open.
36. Comment out `new_im = im.rotate(90, expand=True)` in the Python Script window and uncomment `new_im = im.filter(FIND_EDGES).`
37. Click **Execute** to find the images edges - *Figure 80.48*.

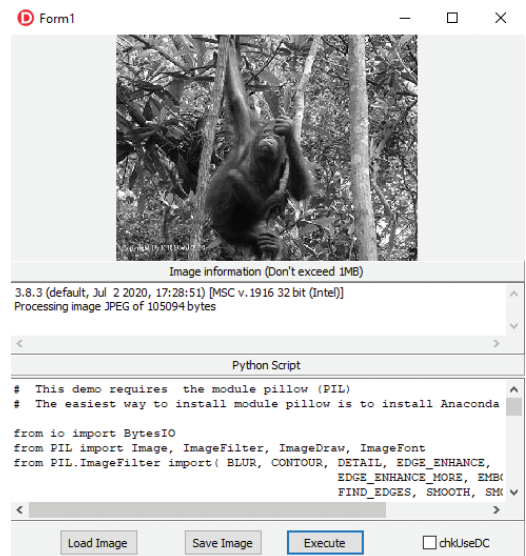


Figure 80.44 Black & white conversion

```
# This demo requires the module pillow (PIL)
# The easiest way to install module pillow is to install Anaconda
from io import BytesIO
from PIL import Image, ImageFilter, ImageDraw, ImageFont
from PIL.ImageFilter import ( BLUR, CONTOUR, DETAIL, EDGE_ENHANCE,
                             EDGE_ENHANCE_MORE, EMBOSS,
                             FIND_EDGES, SMOOTH, SMOOTH_MORE, SHARPEN)

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 = im.filter(ImageFilter.BLUR)
    # new_im = im.filter(ImageFilter.BoxBlur(5))
    # new_im = im.filter(ImageFilter.GaussianBlur(5))
    # new_im = im.crop((100,200,400,500))
    # new_im = im.transpose(Image.FLIP_LEFT_RIGHT)
    # new_im = im.transpose(Image.FLIP_TOP_BOTTOM)
    # new_im = im.transpose(Image.ROTATE_90)
    # new_im = im.resize(round(im.size[0]*0.5), round(im.size[1]*0.5))
    # width, height = im.size
    # draw = ImageDraw.Draw(im)
    # text = "sample watermark"
    # font = ImageFont.truetype('arial.ttf', 36)
    # textwidth, textheight = draw.textsize(text, font)
    # calculate the x,y coordinates of the text
    # margin = 10
    # x = width - textwidth - margin
    # y = height - textheight - margin
    # draw watermark in the bottom right corner
    # draw.text((x, y), text, font=font)
    new_im = im.convert('L')
    # new_im = im.filter(CONTOUR)
    # new_im = im.filter(EMBOSS)
    # new_im = im.filter(FIND_EDGES)
    new_im.format = im.format
    return new_im

def ImageToString(image):
    stream = BytesIO()
    image.save(stream, image.format)
    return stream.getvalue()
```

Figure 80.45 ImageProcessingScript.py

```

Function ImageToPyStr(AGraphic : TGraphic) : Variant;
Var
  _Stream : TMemoryStream;
  _Str : PPyObject;
Begin
  _Stream := TMemoryStream.Create();
  Try
    AGraphic.SaveToStream(_Stream);
    _Str := GetPythonEngine.PyString_FromStringAndSize(_Stream.Memory, _Stream.Size);
    Result := VarPythonCreate(_Str);
    GetPythonEngine.Py_DECREF(_Str);
  Finally
    _Stream.Free;
  End;
End;

Procedure TForm1.Button1Click(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(Memo2.Lines);
  _im := MainModule.ProcessImage(ImageToPyStr(Image1.Picture.Graphic));
  If Not chkUseDC.Checked
  Then
    Begin
      // We have to call PyString_AsStringAndSize because the image may contain zeros
      With GetPythonEngine
      Do
        Begin
          pargs := MakePyTuple([ExtractPythonObjectFrom(_im)]);
          Try
            Try
              presult := PyEval_CallObjectWithKeywords(ExtractPythonObjectFrom(MainModule.ImageToString),
                                                         pargs, nil);
              If (PyString_AsStringAndSize(presult, P, Len) < 0) or (P = nil)
              Then
                Begin
                  ShowMessage('This does not work and needs fixing');
                  Abort;
                End;
            Finally
              Py_XDECREF(pResult);
            End;
          Finally
            Py_DECREF(pargs);
          End;
        End;
      _Stream := TMemoryStream.Create();
      Try
        _Stream.Write(P^, Len);
        _Stream.Position := 0;
        Image1.Picture.Graphic.LoadFromStream(_stream);
      Finally
        _Stream.Free;
      End;
    End
  Else
    Begin
      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;
  End;

```

Figure 80.46 Source code from Python4Delphi Demo 29

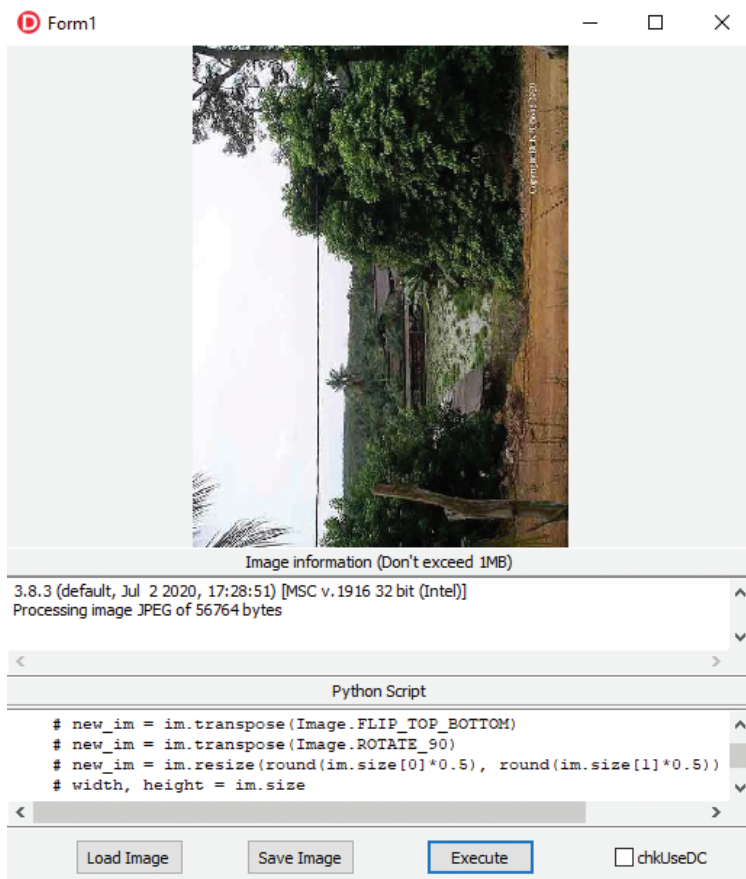


Figure 80.47 Rotated image

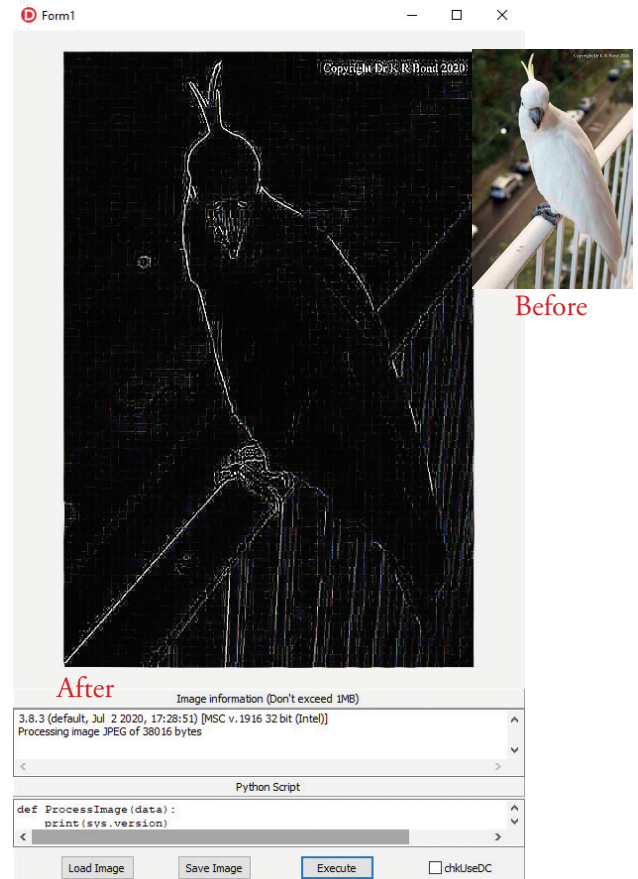


Figure 80.48 Finding edges

Programming Task

- 1 Create a new VCL application which implements image processing based on the Python script shown in Figure 80.45.

The user interface should enable a user to

- load an image
- save an image
- select and apply a particular image processing effect, e.g. convert a coloured image to monochrome (black and white). The application should then automatically execute the corresponding Python code to achieve this effect. No visible reference to this Python code should appear in the user interface when operating normally.

The application should

- check that an image is loaded before attempting to apply any image processing
- prevent images from loading that are above 1MB in size.

Deploying Python4Delphi Application

The minimum that is required to deploy a **Python4Delphi** application consists of

1. the application's exe, e.g. `FirstExampleProject.exe`
2. The dll of the version of Python that was used when developing the application, e.g. `python38.dll`.

Figure 80.49 and *Figure 80.50* show the first two stages of creating a software installer for the two files mentioned above. A free Microsoft Windows® installer **Inno Setup** by [JRSoftware.org](http://www.jrsoftware.org) was used to create an installer `setup.exe` for `FirstExampleProject`.

Figure 80.51 shows that `FirstExampleProject` has been added to the **Start Menu** after running its installer `setup.exe`.

Information

Download Inno Setup QuickStart pack - it is much easier to use because it uses a wizard to guide you through setting up an installer.

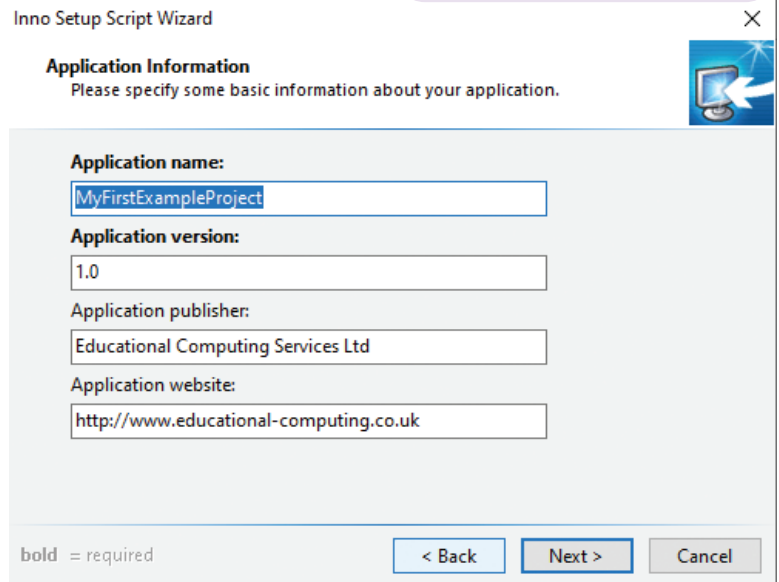


Figure 80.49 Creating an installer for FirstExampleProject.exe using Inno Script Studio Community edition

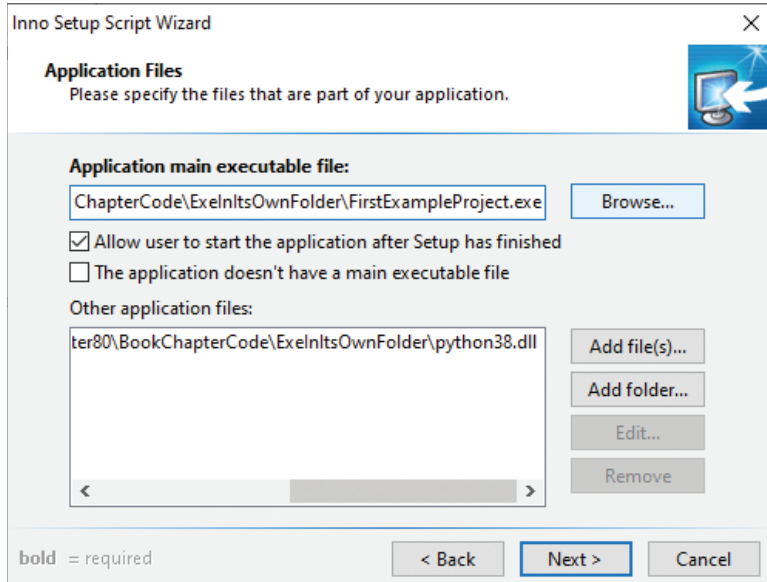


Figure 80.50 Adding the application files

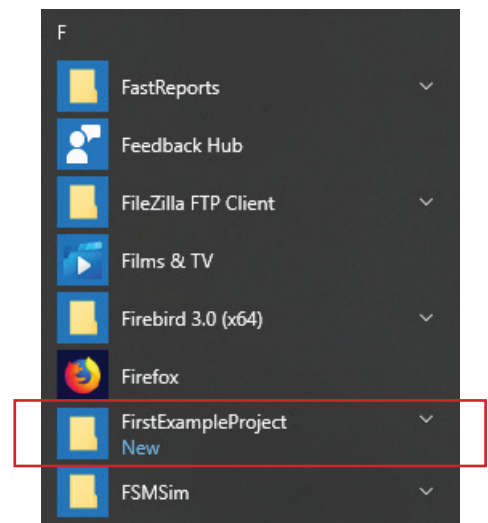


Figure 80.51 Start menu showing FirstExampleProject

TPythonEngine InitScript

TPythonEngine has a property called **InitScript** of type **TStrings** - *Figure 80.52*. Valid Python source code assigned to this property is executed when the application is launched as shown in *Figure 80.53*. In this example, the Python source code assigned to **InitScript** is the string

```
print ("This FirstExampleProject enables
Python scripts in the window\r\n above to be
executed with output appearing here.")
```

`\r` is the escape code for carriage return.

`\n` is the escape code for new line.

Eighth project

This project makes use of the package **SynEdit** which provides syntax highlighting. The package may be downloaded:

1. As a zip file from <https://github.com/SynEdit/SynEdit>.
2. Or using **Tools|GetIt Package Manager**.
3. Or from within Delphi - **File|Open From Version Control...** if you have installed git (<https://gitforwindows.org>) on your computer.

Figure 80.54 shows the expanded **SynEdit** download.

Open **SynEdit.groupproj** from the **Packages** folder in the Delphi IDE choosing the version corresponding to your version of Delphi, e.g. **104S** for **Delphi 10.4 Sydney**.

Highlight the runtime package **SynEdit_R104S.bpl** in the **Projects** pane (or highlight your particular version of **SynEdit**) and click the right mouse button to bring up the menu to use to build the package. Select **From Here|Build All From Here**.

Do the same for the design time package **SynEdit_D104S.bpl**. Once built, click the right mouse button over **SynEdit_D104S.bpl** to bring up the menu again. Select the **Install** option to install the **SynEdit** components.

Figure 80.55 shows that menus for the installed **SynEdit** and **SynEdit Highlighters** components have been installed in the **Component Palette**.

Open project **FirstExampleProject.dproj** and save in a new folder, **EighthExample**, rename the project **EighthExampleProject.dproj** (**Save Project As**).

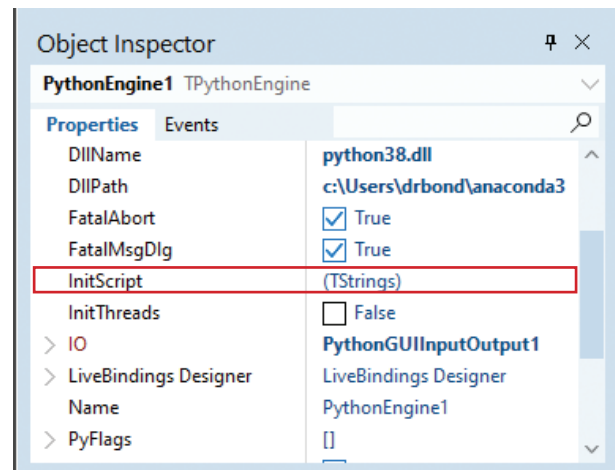


Figure 80.52 Object Inspector view of PythonEngine1 in FirstExampleProject

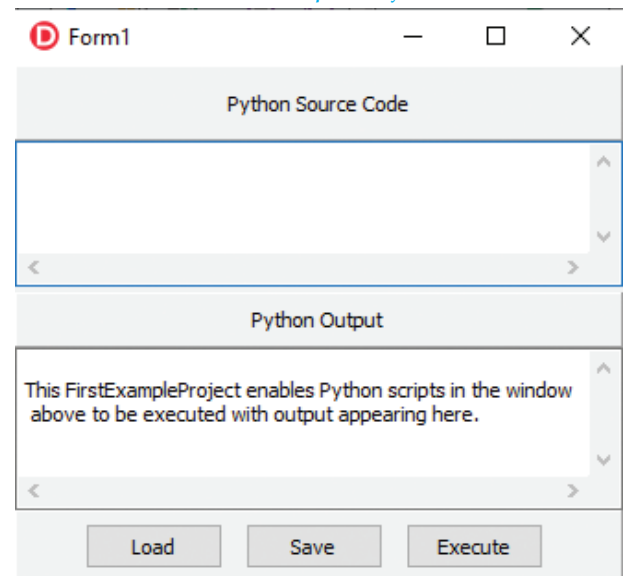


Figure 80.53 FirstExampleProject in execution

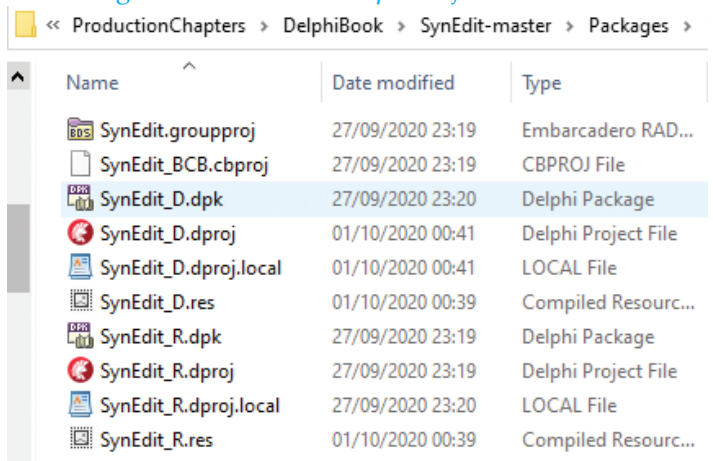


Figure 80.54 Expanded SynEdit download

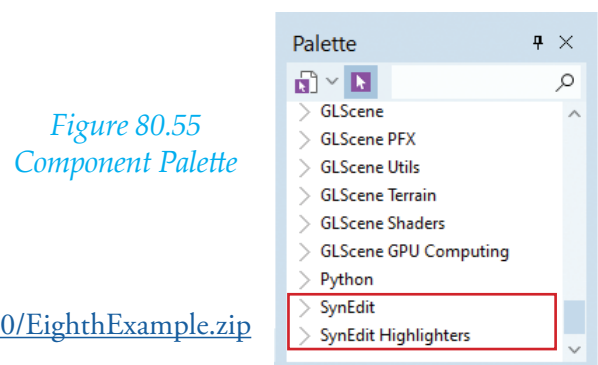


Figure 80.55 Component Palette

Save unit `FirstExampleUnit.pas` in new folder `EighthExample`, rename the unit `EighthExampleUnit.pas` (**Save As**). Add a **TSynEdit** component from the **SynEdit Palette** to the form. Set its **Align** property to **alTop**. Now move it so that it is immediately below the panel with caption `Python Source Code`. Select `Memo1` in the **Structure** pane and click the right mouse button and select from **Edit** from the popup menu then **Delete**. The form should look similar to that shown in [Figure 80.56](#).

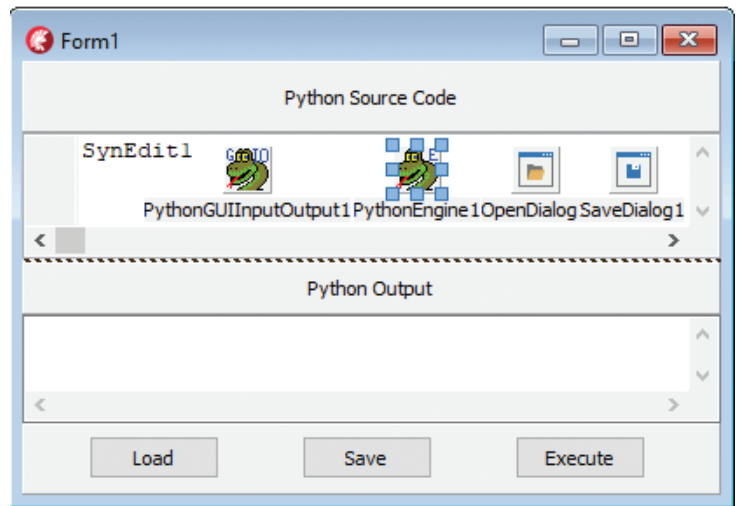


Figure 80.56 Form design for EighthExampleProject

Open the **Lines** property of `SynEdit1` in **Object Inspector** and delete the line `SynEdit1`.

Add a **TSynEditPythonBehaviour** component from the **SynEdit Palette** to the form. Set its **Editor** property to `SynEdit1`.

Add a **TSynPythonSyn** component from the **SynEdit Highlighters Palette** to the form. Set the **Highlighter** property of the **Editor** property of `SynEditPythonBehaviour1` to `SynPythonSyn1`.

Change the event handler for the button labelled `Execute` to

```
PythonEngine1.ExecStrings(UTF8Encode(SynEdit1.Text)).
```

Replace `Memo1` in the event handler for the button labelled `Load` with `SynEdit1`.

Replace `Memo1` in the event handler for the button labelled `Save` with `SynEdit1`.

Select `SynPythonSyn1` in the **Object Inspector** and change the **Foreground** of **KeyAttri** property to **clRed** - [Figure 80.57](#).

Change the **Foreground** of **IdentifierAttri** property to **clHighlight**.

Save project and unit in folder `EighthExample`.

Now compile, link and run (**Shift+Ctrl+F9**) the executable.

Click **Load** and select `BubbleSort.py`.

Click **Execute**.

[Figure 80.58](#) shows the result.

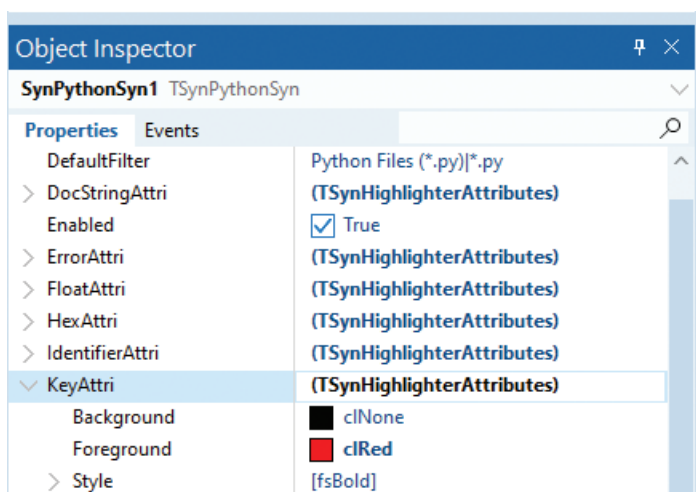


Figure 80.57 Section of Object Inspector showing how to change attribute colour

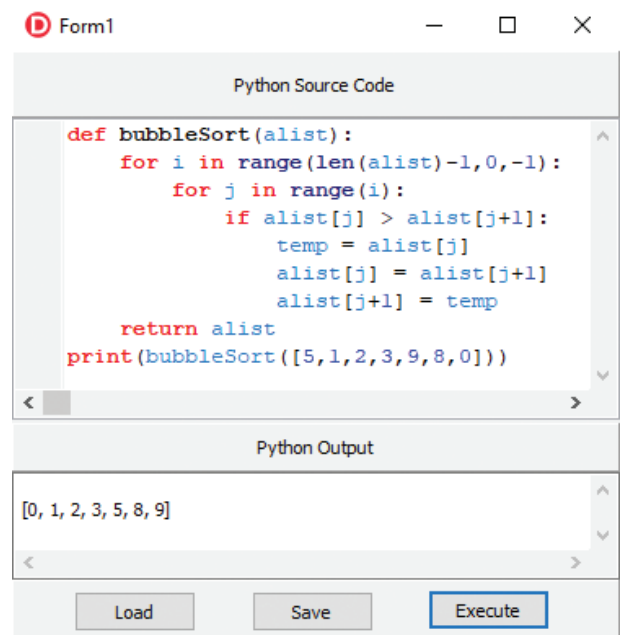


Figure 80.58 EighthExampleProject in execution with BubbleSort.py loaded and run

Ninth project

Save project `EighthExampleProject.dproj` in a new folder, `NinthExample`, rename the project `NinthExampleProject.dproj` (**Save Project As**).

Save unit `EighthExampleUnit.pas` in new folder `NinthExample`, rename the unit `NinthExampleUnit.pas` (**Save As**).

1. Add a **TPyDelphiWrapper** from the **Python Palette** to the form.
2. Check in Object Inspector that **Engine** property of `PyDelphiWrapper1` (default identifier of **TPyDelphiWrapper component**) is set to `PythonEngine1` and **Module** property is set to `PythonModule1`.
3. Add the additional source code shown in [Figure 80.59](#) to the **FormCreate** event handler in `NinthExampleUnit.Pas`.

```

Procedure TForm1.FormCreate(Sender: TObject);
Begin
    PythonEngine1.SetPythonHome('c:\Users\drbond\Anaconda3');
    PythonEngine1.LoadDll;
    Var Py := PyDelphiWrapper1.WrapRecord(@DelphiFunctions,
        TRttiContext.Create.GetType(TypeInfo(TDelphiFunctions))
        As TRttiStructuredType);
    PythonModule1.SetVar('delphi_functions', Py);
    PythonEngine1.Py_DecRef(Py);
End;

```

Figure 80.59 Event handler TForm1.FormCreate

The properties and events displayed at design time in the **Object Inspector** for an instance of a class have the scope **published**. **Published** means that runtime type information (**RTTI**) is generated about these through the application's published interface. External applications can also access this **RTTI** to get information that enables each application to determine the fields, methods, and properties an otherwise indeterminate object has at runtime.

4. Add the following to the **Implementation** section

```

Uses
    System.Rtti,
    System.Threading
    System.Math

Type
    TDelphiFunctions = Record
        Class Function count_primes(MaxN : Integer) : Integer; Static;
    End;

Var
    DelphiFunctions : TDelphiFunctions;

```

For a Python program to access a type defined in a Delphi program we need to supply a published interface, i.e. an **RTTI** structure.

The inline variable declaration(introduced in Delphi 10.3):

```

Var Py := PyDelphiWrapper1.WrapRecord(@DelphiFunctions,
    TRttiContext.Create.GetType(TypeInfo(TDelphiFunctions))
    As TRttiStructuredType);

```

wraps up the published type information in variable `Py` which

```
PythonModule1.SetVar('delphi_functions', Py);
```

associates with the class variable `delphi_functions`, the identifier to be used in the Python program side of things as follows

```
from delphi_module import delphi_functions
```

Check that the **ModuleName** property of `PythonModule1` is `delphi_module`.

The final added line of source code shown in [Figure 80.59](#) frees the object reference stored in `Py` by calling `Py_DecRef`

```
PythonEngine1.Py_DecRef(Py);
```

5. Add the source code shown in [Figure 80.60](#) to the end of `NinthExampleUnit.pas`.

A class method is a method (other than a constructor) that operates on classes instead of objects. To be able to access the function `count_primes` independently of any object reference, it is made a **static** class function.

A **class static method** therefore has no **Self** parameter. The **Self** parameter identifies the object to which the method call applies or is to operate on but there won't be one in this example.

```
Function IsPrime(x : Integer) : Boolean;
Begin
  If (x <= 1) Then Exit(False);
  Var UpperLimit := Floor(Sqrt(x));
  For Var i := 2 To UpperLimit
    Do If (x Mod i = 0) Then Exit(False);
  Exit(True);
End;

Class Function TDelphiFunctions.count_primes(MaxN : Integer) : Integer;
Begin
  Var Count := 0;
  TParallel.&For(2, MaxN, Procedure(i : Integer)
    Begin
      If IsPrime(i) Then AtomicIncrement(Count);
    End);

  Result := Count;
End;
```

Count accessible by more than one thread
so must ensure that transaction is atomic

Figure 80.60 More Source code for NinthExampleUnit.Pas

6. Save project and unit in folder `NinthExample`.
7. Now compile, link (**Shift + F9**) to build the project.
8. Create the Python script `NinthExample.py` as shown in [Figure 80.61](#).

```
from delphi_module import delphi_functions
from timeit import Timer
import math
def test():
    max_n = 1000000
    print(f'Number of primes between 0 and {max_n} = {delphi_functions.count_primes(max_n)}')
def main():
    print(f'Elapsed time: {Timer(stmt=test).timeit(1)} secs')
if __name__ == '__main__':
    main()
```

Figure 80.61 NinthExample.py

9. Press **Shift + Ctrl + F9** to run `NinthExampleProject` without debugging.
10. Click **Load** and open `NinthExample.py`
11. Click **Execute**.

12. The outcome from `NinthExample.py` when run from Delphi `NinthExampleProject` is shown in [Figure 80.62](#) - `delphi_functions.count_primes` with multiple threads takes to five sig. figs only 0.042383 seconds.

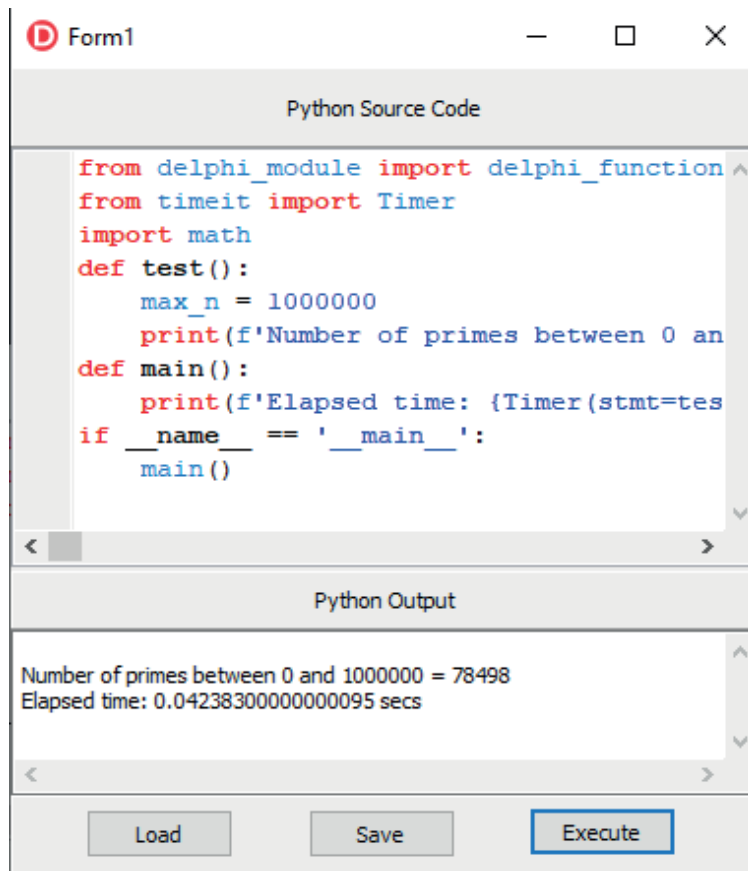


Figure 80.62 NinthExampleProject in execution

Tenth project

www.educational-computing.com/DelphiBook/Code/Chapter80/TenthExample.zip

NumPy is a Python programming language library which adds support for large, multi-dimensional arrays and matrices, as well as a large collection of high-level mathematical functions to operate on these arrays.

NumPy can be installed by using the installer **pip** from the command line. However installing some of the larger Python libraries, particularly those such as **NumPy** which depend on complex low-level C and Fortran packages, is made easier with **Anaconda**. **Anaconda** does all the dependency checking and binary installs that are required. **Anaconda** is free to download and install from <https://www.anaconda.com/products/individual>.

Once the **NumPy** library is installed, an array, for example, is created after importing the numpy module (alias **np**) in Python as follow:

```
import numpy as np
vector = np.array([0, 1, 2, 3])
```

Python list `[0, 1, 2, 3]` is converted and returned by function `np.array`. Python has no native array construct so we rely on libraries such as **NumPy** for array support.

The module **numpy** can be imported into Delphi as follows

```
Var np := Import('numpy');
```

For this statement to compile the **VarPyth** unit from **Python4Delphi** must appear in the **Uses** clause.

A delphi array is created and returned by Delphi function `np.array` is as follows

```
Var np_array : Variant := np.array(VarPythonCreate([1,2,3,4,5,6,7,8,9,10]))
```

Information

Modules are individual .py files from which we can import functions and objects. Packages are collections of such modules.

The variable `np_array` is a variant data structure. A variant data structure is one that allows types that cannot be determined at compile time or whose data structure values can change their type at run time. In this example, `TenthExample`, using a variant for the variable `np_array` allows it to work in both Delphi and Python. We need a **TPythonModule**, `PythonModule1`, with which to set up shared access to `np_array` for Delphi and Python as follows

```
PythonModule1.SetVar('np_array', ExtractPythonObjectFrom(np_array));
```

This setting up may be done in the `FormCreate` event handler as shown in [Figure 80.63](#).

```
Procedure TForm1.FormCreate(Sender : TObject);
Begin
    PythonEngine1.SetPythonHome('c:\Users\drbond\Anaconda3');
    PythonEngine1.LoadDll;
    Var np := Import('numpy');
    Var np_array : Variant := np.array(VarPythonCreate([1,2,3,4,5,6,7,8,9,10]));
    PythonModule1.SetVar('np_array', ExtractPythonObjectFrom(np_array));
End;
```

These two statements needed because Anaconda installed Python 3.8 which is used for the Python program.

Figure 80.63 TenthExampleUnit.pas

With the **ModuleName** property `delphi_module` assigned to `PythonModule1`, the Delphi constructed array `np_array` may be imported into the Python program `np_array.py` as shown in [Figure 80.64](#).

`np_array` may then be accessed in Python, e.g. `print("np_array = ", np_array)`

In order to use what has been described, a graphical user interface is needed.

Save project `NinthExampleProject.dproj` in a new folder, `TenthExample`, rename the project `TenthExampleProject.dproj` (**Save Project As**).

Save unit `TenthExampleUnit.pas` in new folder `TenthExample`, rename the unit `TenthExampleUnit.pas` (**Save As**).

```
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)
```

Figure 80.64 np_array.py

1. Delete `PyDelphiWrapper1`, Function `IsPrime` and Class Function `TDelphiFunctions.count_primes`.
2. Edit `FormCreate` so that it conforms to [Figure 80.63](#).
3. Edit the **Uses** clause under **Implementation** as follows

```
Uses
    System.Threading,
    System.Math,
    VarPyth;
```

4. Drop a **TListBox** component on the form and set its **Align** property to **alRight** in the **Object Inspector** so it aligns between the `Python Output` panel and the bottom panel.
5. Drop a **TSplitter** component on the form and set its **Align** property to **alRight** so it aligns to the left of the **TListBox** component.

The user interface at design time should look similar to [Figure 80.65](#).

6. Edit `Button1Click` so that its body now is as shown in [Figure 80.66](#).

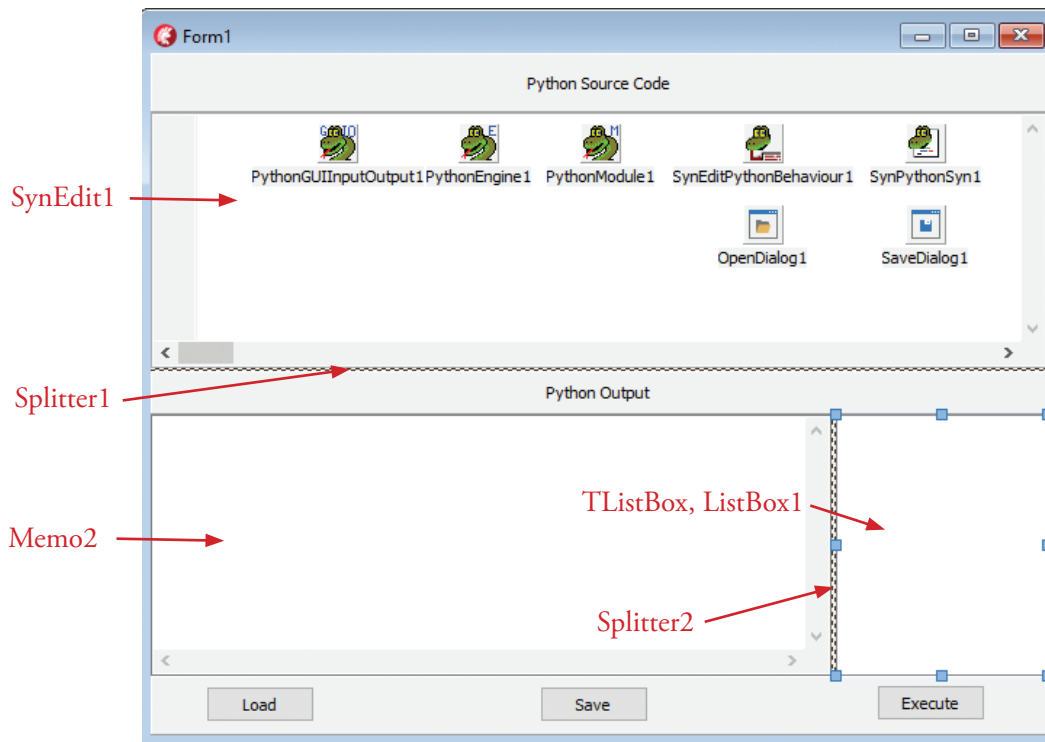


Figure 80.65 TenthExampleProject user interface at design time

```

Procedure TForm1.Button1Click(Sender : TObject);
Begin
  // PythonEngine1.ExecStrings(SynEdit1.Lines);
  GetPythonEngine.ExecString(UTF8Encode(SynEdit1.Text));
  For Var V In VarPyIterate(MainModule.res_array)
  Do ListBox1.Items.Add(V);
End;
    
```

Alternative way of executing the contents of SynEdit1

MainModule always refers to a Python module, in this case np_array.py

Figure 80.66 TenthExampleUnit.pas Button1Click event handler

7. Add at the end of TenthExampleUnit.Pas Initialization
`MaskFPUEExceptions(True);`
`ReportMemoryLeaksOnShutdown := True;`
 just before final **End**.
8. Save project and unit in folder TenthExample.
9. Create the Python script `numpy_array.py` as shown in Figure 80.64.
10. Now build and run the project without debugging (**Shift+Ctrl+F9**).
11. Click **Load** and open `numpy_array.py`.
12. Click **Execute**.

Figure 80.67 shows the outcome. The array of integers submitted to the Python script are returned squared. The returned result is displayed in the **TMemo** window Memo2 and in the **TListBox** window ListBox1.

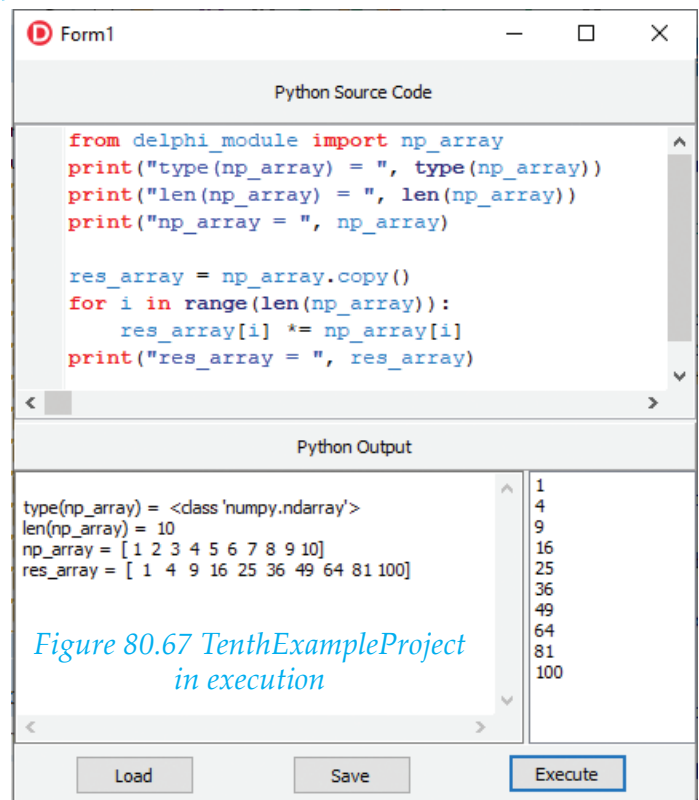


Figure 80.67 TenthExampleProject in execution

Information

FPU Exception mask

Delphi's default FPU Exception mask is different from most other Windows apps. It is incompatible with Python libraries written in C or C++. It is necessary to match the FPU mask that numpy, scipy, tensorflow, etc expect to operate with. PythonEngine.pas provides a function for doing that, `MaskFPUEExceptions`, which needs to be called - `MaskFPUEExceptions(True)` - before Python is loaded. The Initialization section of the main Delphi form can be used to do this.

Eleventh project www.educational-computing.com/DelphiBook/Code/Chapter80/EleventhExample.zip

Save project `TenthExampleProject.dproj` in a new folder, `EleventhExample`, rename the project `EleventhExampleProject.dproj` (**Save Project As**).

Save unit `TenthExampleUnit.pas` in new folder `EleventhExample`, rename the unit `EleventhExampleUnit.pas` (**Save As**).

1. Add two **TPanel** components to the form, `Panel4` and `Panel5`, clear the **Caption** property of each.
2. Set **Align** property of `Panel4` to **alLeft** and adjust its **Width** property so that it occupies less than half of a wide form - see [Figure 80.69](#).
3. In this order in the **Structure** pane, move `Panel3` to `Panel4`, move `Memo2` to `Panel4`, move `Splitter1` to `Panel4`, move `Panel2` to `Panel4`, move `SynEdit1` to `Panel4`, finally move `Panel1` to `Panel4`.
4. Delete `ListBox1`.
5. **Align** property of `Panel5` to **alClient**.
6. Set `Splitter2`'s **Align** property to **alLeft**.
7. Add a **TEdgeBrowser** component, `EdgeBrowser1`, to `Panel5` and set its **Align** property to **alClient**.

Support in VCL applications for working with web content through the **Chromium-based Edge WebView2** browser control is now present in Delphi 10.4 via the new **TEdgeBrowser** component.

TEdgeBrowser replaces **TWebBrowser**, which uses the **Internet Explorer WebBrowser** browser control.

TWebBrowser is still available in the VCL component set, with some notable changes.

TWebBrowser uses the Operating System-supplied **Internet Explorer WebBrowser browser control** so there is no preparation required; it will work wherever Windows has the Internet Explorer control available.

As **Microsoft Edge** is not an Operating System component at the time of writing this book, it is necessary to install **EdgeView2 SDK 0.9.488** by Microsoft® before the **TEdgeBrowser** component can work in Delphi applications.

The easiest way to do this is to use **GetIt Package Manager - Tools|GetIt Package Manager...**, type **Edge** in search box then return. [Figure 80.68](#) shows the currently available **EdgeView2 SDK**. Click this to select, then click the install button to install the package. This is shown as done in [Figure 80.68](#) by **Installed** being ticked.

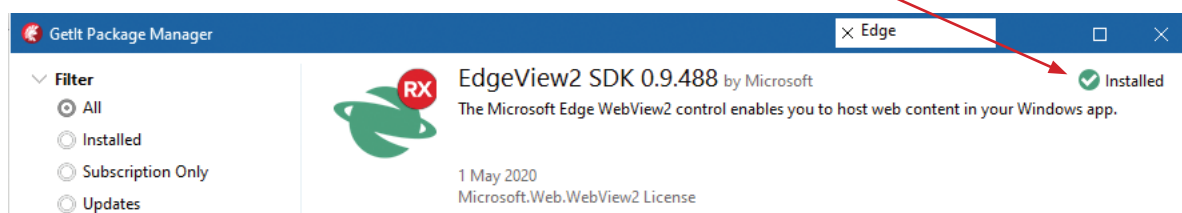


Figure 80.68 GetIt Package Manager EdgeView2 SDK

For the **TEdgeBrowser** component to work at runtime, the corresponding **WebView2Loader.dll** - x86 or x64 - must be placed in the project's output directory, i.e. `EleventhExample\Win32\Debug` or `EleventhExample\Win64\Debug`. Locate the folder containing the relevant **WebView2Loader.dll** (where Delphi 10.4 is installed, use **Redist** folder) and copy this dll.

8. Add a **TPyDelphiWrapper** component, `PyDelphiWrapper1`, to the form.
9. The **Engine** property of `PyDelphiWrapper1` should be set to `PythonEngine1` and its **Module** property set to `PythonModule1`.
10. Edit the `FormCreate` event handler so that it is as shown in [Figure 80.70](#).

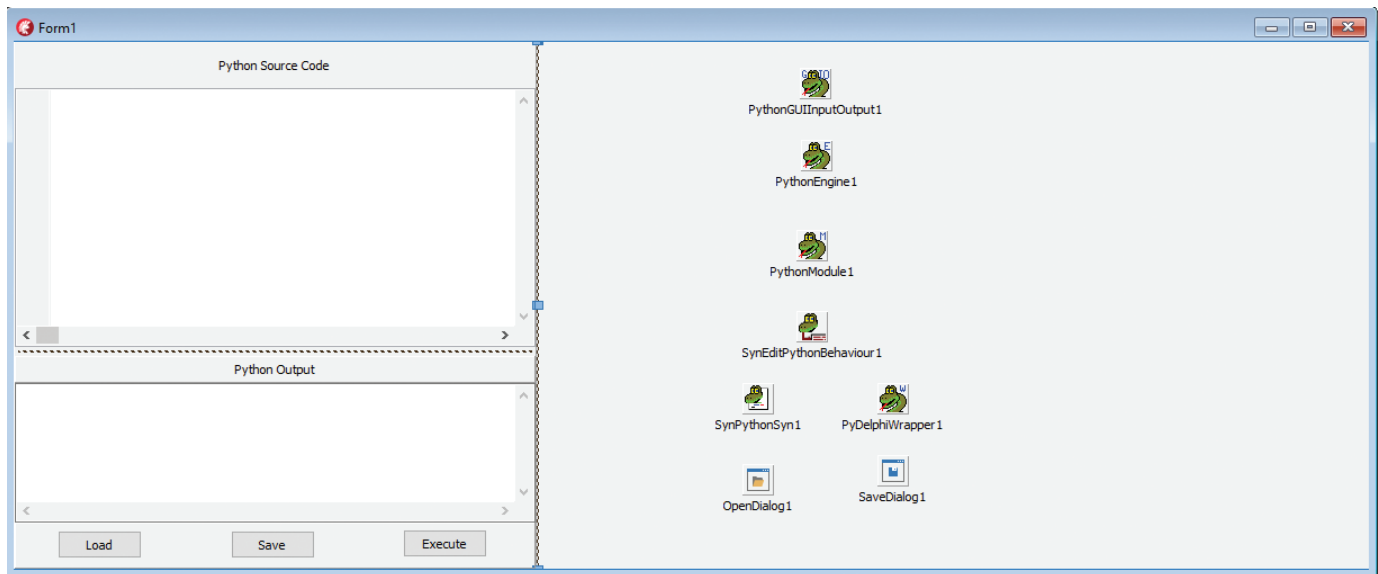


Figure 80.69 EleventhExampleProject GUI design

11. Edit the `Button1Click` event handler so that it is as shown in Figure 80.71.

12. Delete the **Uses** clause in the **Implementation** section.

Object reference variable `Py` assigned reference to wrapped `EdgeBrowser1`

```

Procedure TForm1.FormCreate(Sender : TObject);
Begin
    PythonEngine1.SetPythonHome('c:\Users\drbond\Anaconda3');
    PythonEngine1.LoadDll;
    Var Py := PyDelphiWrapper1.Wrap(EdgeBrowser1, soReference);
    PythonModule1.SetVar('edge_browser', Py);
    GetPythonEngine.Py_DECREF(Py);
End;
    
```

Enumerated type defined in `WrapDelphi.pas`

Setting name that Python script will use to refer to the object referenced by `Py`

Figure 80.70 EleventhExampleUnit.pas FormCreate event handler

```

Procedure TForm1.Button1Click(Sender : TObject);
Begin
    PythonEngine1.ExecStrings(SynEdit1.Lines);
End;
    
```

Figure 80.71 EleventhExampleUnit.pas Button1Click event handler

13. Change **ReportMemoryLeaksOnShutdown** in the Initialization section to **False**. There is a very small memory leak.

14. This example project and others require the following python modules:

- numpy** - supports large, multi-dimensional arrays and matrices, and high-level mathematical functions to operate on these arrays.
- pandas** - supports data manipulation and analysis, includes data structures and operations for manipulating numerical tables and time series.
- matplotlib** - plotting library.
- seaborn** - data visualization library based on matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- mpld3** - brings together matplotlib and the JavaScript library for creating interactive data visualizations for the web, **D3js**. The result is a simple API for exporting your matplotlib graphics to HTML code which can be used within the browser.
- bokeh** - an interactive visualization library for modern web browsers.

altair - a declarative statistical visualization library.

scikit-learn - machine learning library.

15. If Anaconda is used then the **conda package manager** is available to install any required packages.

Switch to the command line and enter the command as shown in [Figure 80.72](#), replacing package-name with the name of the required package. Anaconda will already have installed some of the required packages.

```
conda install package-name
```

Figure 80.72 Command line use of conda

16. Save project and unit in folder `EleventhExample`.

17. Three Python example scripts have already been prepared:

- `EleventhExampleMatplotlib.py`
- `EleventhExampleBokeh.py`
- `EleventhExampleAltair.py`

18. Now build (**Shift+F9**) the project.

19. Paste **WebView2Loader.dll** in `Win32\Debug` or `Win64\Debug` folder depending on target.

20. Click Run Without Debugging (**Shift+Ctrl+F9**).

21. Click **Load** and open `EleventhExampleBokeh.py`.

22. Click **Execute** - [Figure 80.73](#) shows the outcome.

23. Use the sliders in the righthand window to interact with the Python script and alter the sine wave drawn by Delphi in the righthand window.

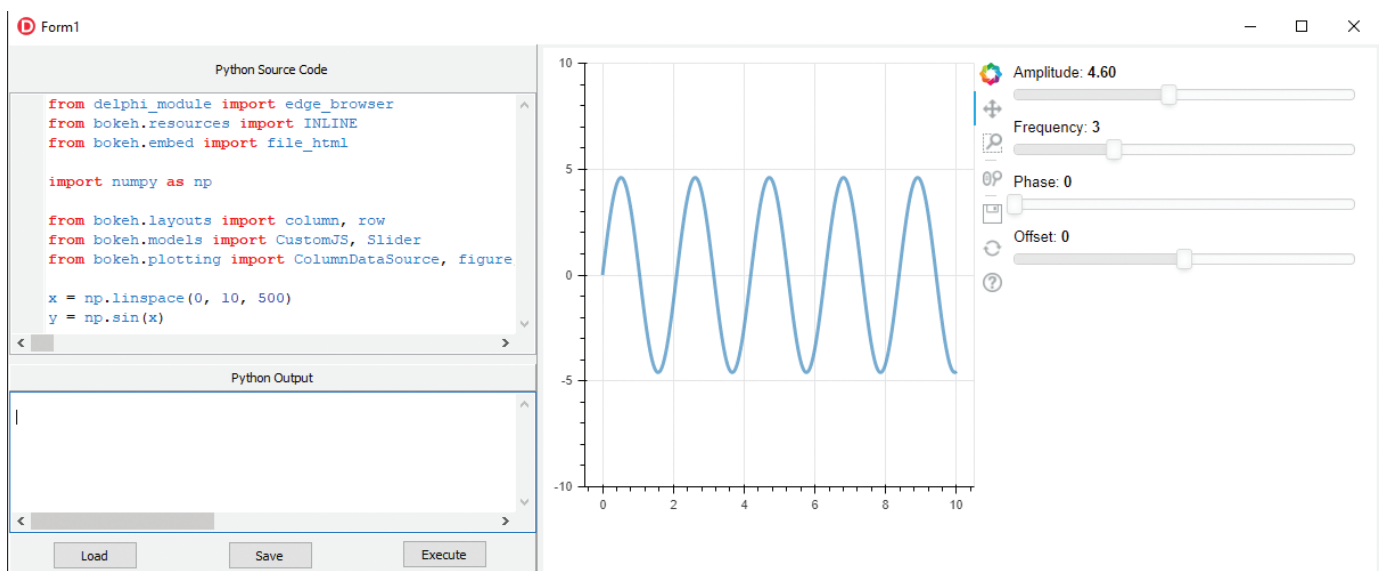


Figure 80.73 `EleventhExampleProject` in execution with script `EleventhExampleBokeh.py` loaded

Save project `EleventhExampleProject.dproj` in a new folder, `TwelfthExample`, rename the project `TwelfthExampleProject.dproj` (**Save Project As**).

Save unit `EleventhExampleUnit.pas` in new folder `TwelfthExample`, rename the unit `TwelfthExampleUnit.pas` (**Save As**).

HOW TO PROGRAM EFFECTIVELY IN DELPHI

We need to install the package **SVGIconImageList VCL & FMX** (*Figure 80.74*) using GetIt Package Manager - **Tools|GetIt Package Manager**. Use of the components in this package simplifies manipulating SVG images. Once installed the components can be found in the component palette under **Ethea**.

1. Delete `EdgeBrowser1` component from `Panel5`.
2. Add a **TSVGIconImage** component, `SVGIconImage1` to `Panel5`.
3. Edit `FormCreate` so that it is as shown in *Figure 80.75*.

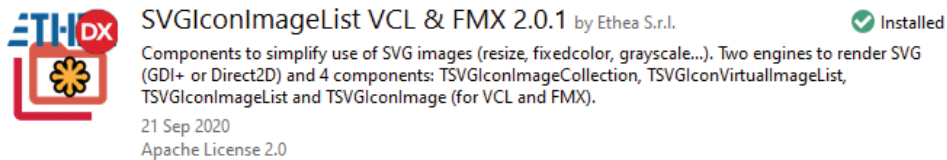


Figure 80.74 View in GetIt Package Manager of SVGIconImageList VCL and FMX

www.educational-computing.com/DelphiBook/Code/Chapter80/TwelfthExample.zip

4. Save project and unit in folder `TwelfthExample`.
5. A Python example script has already been prepared `TwelfthExamplePenguins.py` - *Figure 80.76*.
6. Now run (**Shift+Ctrl+F9**) the project.
7. Click **Load** and open `TwelfthExamplePenguins.py`.

```
Procedure TForm1.FormCreate(Sender : TObject);
Begin
    PythonEngine1.SetPythonHome('c:\Users\drbond\Anaconda3');
    PythonEngine1.LoadDll;
    Var Py := PyDelphiWrapper1.Wrap(SVGIconImage1, soReference);
    PythonModule1.SetVar('svg_image', Py);
    GetPythonEngine.Py_DECREF(Py);
End;
```

Setting name that Python script will use to refer to the object referenced by Py

Figure 80.75 TwelfthExampleUnit.pas FormCreate event handler

8. Click **Execute** - *Figure 80.77* shows the outcome.

```
from delphi_module import svg_image
from io import StringIO
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("penguins")
sns.pairplot(df, hue="species")
figfile = StringIO()
plt.savefig(figfile, format='svg')
figdata_svg = figfile.getvalue()
svg_image.SvgText = figdata_svg
```

Figure 80.76 TwelfthExamplePenguins.py

Thirteenth project

Load project `NinthExampleProject.dproj` and then save in a new folder, `ThirteenthExample`, rename the project `ThirteenthExampleProject.dproj` (**Save Project As**).

Save unit `NinthExampleUnit.pas` in new folder `ThirteenthExample`, rename the unit `ThirteenthExampleUnit.pas` (**Save As**).

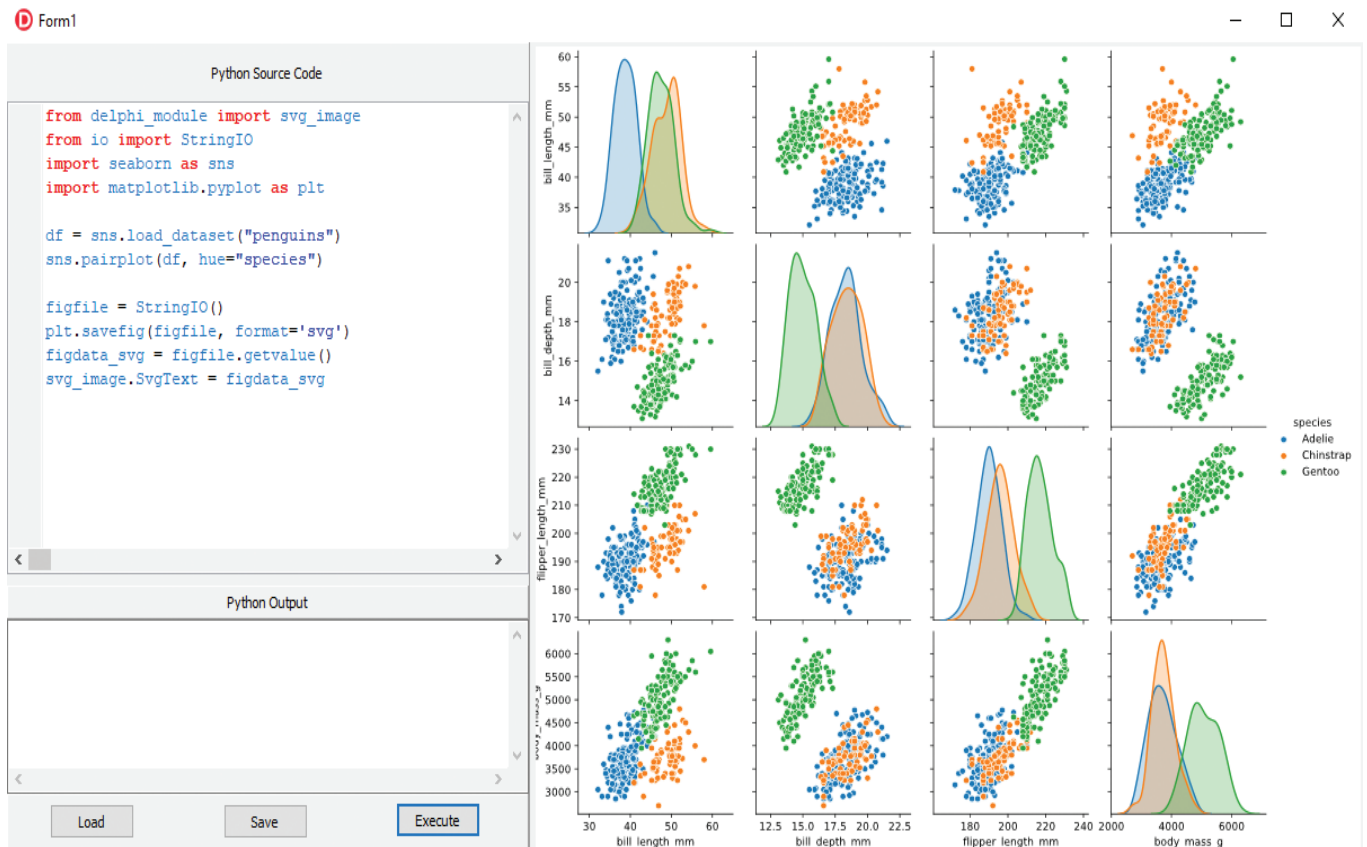


Figure 80.77 TwelfthExampleProject in execution

1. Delete `PythonWrapper` www.educational-computing.com/DelphiBook/Code/Chapter80/ThirteenthExample.zip

2. Replace **Uses** clause in the **Implementation** section with

`Uses`

`TypeInfo, VarPyth, WrapDelphiVCL`

3. Delete **Type** and **Var** statements in the **Implementation** section.

4. Delete **Function** `IsPrime` in the **Implementation** section.

5. Delete **Class Function** `TDelphiFunctions.count_primes` in the **Implementation** section.

6. Insert just before **End** keyword of `TForm1` Class definition

`Public`

`PyDelphiWrapper1 : TPyDelphiWrapper;`

7. Delete body of event handler `FormCreate` and replace with

8. Replace event handler `Button1Click` with

9. Rename **ModuleName** property of `PythonModule1`, `delphi_controls`.

10. Set `Form1`'s **Width** property to 980.

11. Set `Form1`'s **Height** property to 690.

```

PythonEngine1.SetPythonHome('c:\Users\drbond\Anaconda3');
PythonEngine1.LoadDll;
PyDelphiWrapper1 := TPyDelphiWrapper.Create(Self);
PyDelphiWrapper1.Engine := PythonEngine1;
PyDelphiWrapper1.Module := PythonModule1;
PyDelphiWrapper1.Initialize; //Use only if PyDelphiWrapper1 created at run time

```

- Set `SynEdit`'s **RightEdge** property to 120.

```
Procedure TForm1.Button1Click(Sender : TObject);
Var
    PyObjectRef : PPyObject;
Begin
    PyObjectRef := PyDelphiWrapper1.Wrap(Form1);
    PythonModule1.SetVar('MainForm', PyObjectRef);
    PythonEngine1.Py_DECREF(PyObjectRef);
    PythonEngine1.CheckError;
    PythonEngine1.ExecStrings(SynEdit1.Lines);
End;
```

- Add to the end of the unit

Initialization

```
ReportMemoryLeaksOnShutdown := DebugHook <> 0;
```

- Save project and unit in folder `ThirteenthExample`.
- A Python example script has already been prepared, `ThirteenthExample.py` - [Figure 80.79](#).
- Now run (**Shift+Ctrl+F9**) the project.
- Click `Load` and open `ThirteenthExample.py`.
- Click `Execute` - [Figure 80.78](#) shows the outcome.

The design of the Python-generated form may be prototyped first in Delphi. Once the form design is complete in Delphi, the property values may be read from the corresponding `.dfm` file and used in the Python script.

Fourteenth project

- Create a new **Windows VCL Application for Delphi**.

Output from Python script `ThirteenthExample.py`

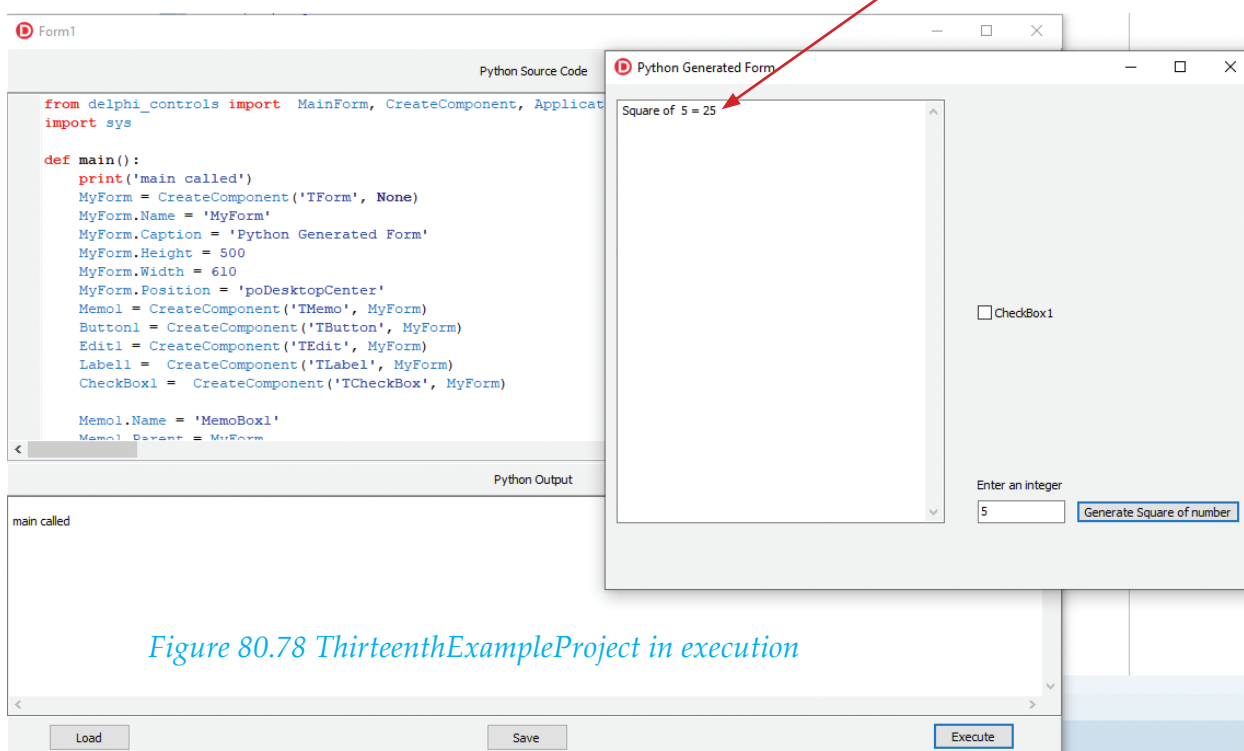


Figure 80.78 ThirteenthExampleProject in execution

```

from delphi_controls import MainForm, CreateComponent, Application, Screen, Form, Button, CheckBox, Label, Memo
import sys

def main():
    print('main called')
    MyForm = CreateComponent('TForm', None)
    MyForm.Name = 'MyForm'
    MyForm.Caption = 'Python Generated Form'
    MyForm.Height = 500
    MyForm.Width = 610
    MyForm.Position = 'poDesktopCenter'
    Memo1 = CreateComponent('TMemo', MyForm)
    Button1 = CreateComponent('TButton', MyForm)
    Edit1 = CreateComponent('TEdit', MyForm)
    Label1 = CreateComponent('TLabel', MyForm)
    CheckBox1 = CreateComponent('TCheckBox', MyForm)

    Memo1.Name = 'MemoBox1'
    Memo1.Parent = MyForm
    Memo1.Top = 14
    Memo1.Width = 300
    Memo1.Height = 387
    Memo1.Left = 10
    Memo1.Text = ''
    Memo1.TabOrder = 0
    Memo1.ScrollBars = 'ssVertical'
    Button1.Name = 'Button1'
    Button1.Parent = MyForm
    Button1.Left = 430
    Button1.Top = 380
    Button1.Width = 150
    Button1.Height = 21
    Button1.Caption = 'Generate Square of number'
    Button1.TabOrder = 1
    def ClickHandler(Sender):
        if CheckBox1.Checked:
            print(Sender.Name, ' was clicked')
            Memo1.Lines.Add( 'Square of ' + Edit1.Text + ' = ' + str(int(Edit1.Text) * int(Edit1.Text)))

    Button1.OnClick = ClickHandler
    Edit1.Name = 'Edit1'
    Edit1.Parent = MyForm
    Edit1.Left = 340
    Edit1.Top = 380
    Edit1.Width = 80
    Edit1.Height = 21
    Edit1.NumbersOnly = True
    Edit1.TabOrder = 2
    Edit1.Text = ''

    def KeyPressHandler(Sender, Key):
        print('non-numeric value')

    def KeyUpHandler(Sender, Key, Shift):
        if not (Key.Value in range(48, 59)) :
            a = Edit1.Text
            Edit1.Text = a[:len(a)-1]
            Edit1.SelStart = len(a)
            print('UP:non-numeric value')

    Edit1.OnKeyUp = KeyUpHandler
    Label1.Name = 'Label1'
    Label1.Parent = MyForm
    Label1.Left = 340
    Label1.Top = 359
    Label1.Width = 121
    Label1.Height = 21
    Label1.Caption = 'Enter an integer'
    CheckBox1.Name = 'CheckBox1'
    CheckBox1.Parent = MyForm
    CheckBox1.Left = 340
    CheckBox1.Top = 200
    CheckBox1.Width = 97
    CheckBox1.Height = 17

    MyForm.ShowModal()

if __name__ == '__main__':
    main()

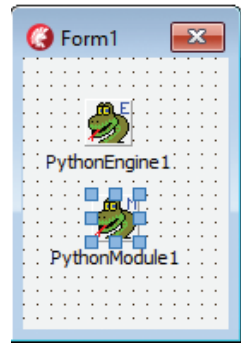
```

Figure 80.79 ThirteenExample.py

HOW TO PROGRAM EFFECTIVELY IN DELPHI

www.educational-computing.com/DelphiBook/Code/Chapter80/FourteenthExample.zip

- Drop a **TPythonEngine** component on the form, `PythonEngine1`.
- Drop a **TPythonModule** component on the form, `PythonModule1` - *Figure 80.80*.
- Set `Form1`'s **Width** property to 136, its **Height** property to 198 and **BorderIcons** property to `[biSystemMenu]`.
- Set properties of `PythonEngine1` and `PythonModule1` in Object Inspector as for *ThirteenthExample*.
- Create event handler `FormCreate` with body as shown in *Figure 80.81*.
- Add `PyDelphiWrapper1` to `TForm1`, a **Uses** clause to the **Implementation** section and variable `PythonProgram` as shown in *Figure 80.81*.
- Rename the project `FourteenthExampleProject.dproj` (**Save Project As**) and save in new folder `FourteenthExample`.
- Save unit as `FourteenthExampleUnit.pas` in new folder `FourteenthExample` (**Save As**).
- Now build (**Shift+F9**) the project.
- Place a copy of `FourteenthExample.py` (*Figure 80.82*) in the `Windows32\Debug` folder of `FourteenthExample`.
- Now run (**Shift+Ctrl+F9**) the project. *Figure 80.83* shows the outcome.



*Figure 80.80
FourteenthExampleProject
form design*

```
Unit FourteenthExampleUnit;
Interface
Uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
  System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,
  PythonEngine, WrapDelphi;
Type
  TForm1 = Class(TForm)
    PythonEngine1 : TPythonEngine;
    PythonModule1 : TPythonModule;
    Procedure FormCreate(Sender : TObject);
    Public
      PyDelphiWrapper1 : TPyDelphiWrapper;
    End;
Var
  Form1 : TForm1;
Implementation
{$R *.dfm}
Uses
  TypInfo, VarPyth, WrapDelphiVCL;
Var
  PythonProgram : TStringList;
Procedure TForm1.FormCreate(Sender : TObject);
Var
  PyObjectRef : PPyObject;
Begin
  Form1.Position := poDesktopCenter;
  Form1.Width := 0;
  Form1.Height := 0;
  Form1.BorderIcons := [biSystemMenu];
  PythonEngine1.SetPythonHome('c:\Users\drbond\Anaconda3');
  PythonEngine1.LoadDll;
  PyDelphiWrapper1 := TPyDelphiWrapper.Create(Self);
  PyDelphiWrapper1.Engine := PythonEngine1;
  PyDelphiWrapper1.Module := PythonModule1;
  PyDelphiWrapper1.Initialize;
  PyObjectRef := PyDelphiWrapper1.Wrap(Form1);
  PythonModule1.SetVar('MainForm', PyObjectRef);
  PythonEngine1.Py_DECREF(PyObjectRef);
  PythonEngine1.CheckError;
  PythonProgram := TStringList.Create;
  PythonProgram.LoadFromFile('ThirteenthExample.py', TEncoding.UTF8);
  PythonEngine1.ExecString(PythonProgram.Text);
End;
End.
```

Figure 80.81 FourteenthExampleUnit.pas

```

from delphi_controls import MainForm, CreateComponent, Application, Screen, Form, Button, Label,
Memo
import sys
def main():
    print('main called')
    MyForm = CreateComponent('TForm', None)
    MyForm.Name = 'MyForm'
    MyForm.Caption = 'Python Generated Form'
    MyForm.Height = 500
    MyForm.Width = 610
    MyForm.Position = 'poDesktopCenter'
    Memo1 = CreateComponent('TMemo', MyForm)
    Button1 = CreateComponent('TButton', MyForm)
    Edit1 = CreateComponent('TEdit', MyForm)
    Label1 = CreateComponent('TLabel', MyForm)

    Memo1.Name = 'MemoBox1'
    Memo1.Parent = MyForm
    Memo1.Top = 14
    Memo1.Width = 300
    Memo1.Height = 387
    Memo1.Left = 10
    Memo1.Text = ''
    Memo1.TabOrder = 0
    Memo1.ScrollBars = 'ssVertical'
    Button1.Name = 'Button1'
    Button1.Parent = MyForm
    Button1.Left = 430
    Button1.Top = 380
    Button1.Width = 150
    Button1.Height = 21
    Button1.Caption = 'Generate Square of number'
    Button1.TabOrder = 1
    def ClickHandler(Sender):
        Memo1.Lines.Add('Square of ' + Edit1.Text + ' = ' + str(int(Edit1.Text) * int(Edit1.Text)))
    Button1.OnClick = ClickHandler
    Edit1.Name = 'Edit1'
    Edit1.Parent = MyForm
    Edit1.Left = 340
    Edit1.Top = 380
    Edit1.Width = 80
    Edit1.Height = 21
    Edit1.NumbersOnly = True
    Edit1.TabOrder = 2
    Edit1.Text = ''
    def KeyUpHandler(Sender, Key, Shift):
        if not (Key.Value in range(48, 59)):
            a = Edit1.Text
            Edit1.Text = a[:len(a)-1]
            Edit1.SelStart = len(a)
    Edit1.OnKeyUp = KeyUpHandler
    Label1.Name = 'Label1'
    Label1.Parent = MyForm
    Label1.Left = 340
    Label1.Top = 359
    Label1.Width = 121
    Label1.Height = 21
    Label1.Caption = 'Enter an integer'
    MyForm.ShowModal()
if __name__ == '__main__':
    main()

```

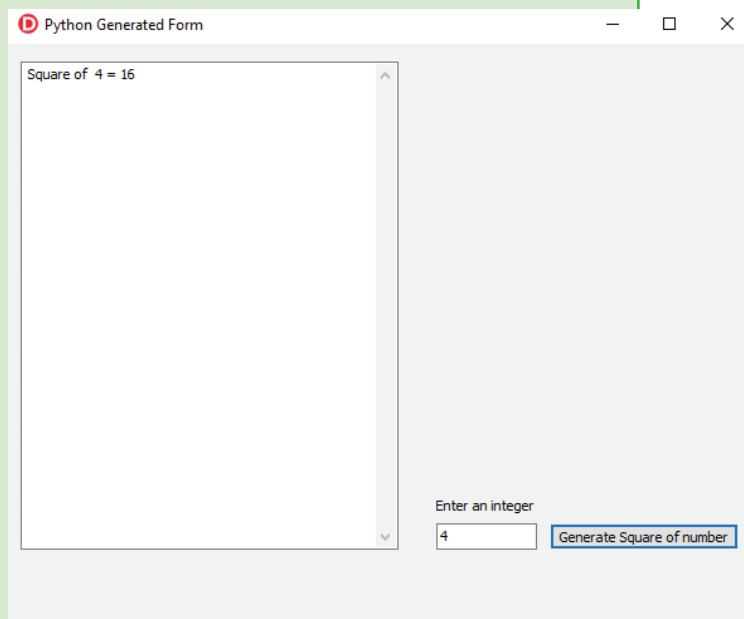


Figure 80.83 FourteenthExampleProject in execution

Information

Python for Delphi

With Python for Delphi it is possible to leverage the best of both Python and Delphi.

With Python for Delphi:

1. It is very easy to integrate Python into Delphi applications in a RAD way, thus providing access to a vast range of Python libraries.
2. Expose Delphi functions/procedures, objects, records and types to Python using low-or high-level interfaces.
3. Create/access/use Python objects/modules in Delphi code using a high-level interface (VarPyth).
4. Run Python code in threads.
5. Create Python extension modules.
6. Wrap VCL as a Python extension module to create GUIs with Python.

Figure 80.82 FourteenthExample.py

Fifteenth project

This project was inspired by Kiriakos Vlahos's Webinar

<https://github.com/pyscripter/python4delphi/tree/master/Tutorials/Webinar%20I>

Python's excellent library support for data science and machine learning has led to it dominating these two areas.

Delphi is renowned for its support for Rapid Application Development(RAD) - the first version of Skype to be created used a Delphi-programmed front end and a C-programmed back end.

It therefore makes sense to leverage the strengths of each to create a usable machine learning application.

This project is dependent on the Python library scikit-learn (**sklearn**), a free software machine learning library for the Python Programming Language. This machine learning library forms the core of the Python back end whilst the front end, written in Delphi, reports on and displays an output graph that shows the closeness of the predicted world-wide figures for the total number of Covid-19 cases over a ten day period at the end of data range. The front and back ends are connected by Python for Delphi.

Use the **conda package manager** from Anaconda or **pip** install the Python machine learning library **sklearn**.

Also make sure that **numpy**, **pandas**, **matplotlib** and **io** are also installed.

This project uses COVID-19 time-series data for every country in the world reporting the total number of Covid-19 cases to-date. This data is available in comma separated value(**csv**) format from the Center(sic) for Systems Science and Engineering (CSSE) at John Hopkins University:

https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv

The format is Province/State, Country/Region, Latitude, Longitude, total recorded number of cases to date (dates range from 22/1/2020 to 25/10/2020, inclusive for the day that this project was run).

For example,

, United Kingdom, 55.3781, -3.4336, 0, 0,, 873800

The first field is empty because the figures are not broken down in the UK's case by area.

The Python back end program is trained and confirmed on a subset of this data. The evolved statistical model is then tested on a ten day subset of this data and the result plotted and saved as an SVG image. This image is then exported for display by the Delphi program. Python for Delphi connects the front and back ends.

The machine learning model uses Bayesian Ridge Regression hybridized with an n-degree Polynomial. Probabilistic distribution is used to estimate the value of the dependent variable instead of relying traditional methods.

The initial values of the parameters to be fine-tuned are

```
tol = [1e-4, 1e-3, 1e-2]
alpha_1 = [1e-7, 1e-6, 1e-5, 1e-4]
alpha_2 = [1e-7, 1e-6, 1e-5, 1e-4]
lambda_1 = [1e-7, 1e-6, 1e-5, 1e-4]
lambda_2 = [1e-7, 1e-6, 1e-5, 1e-4]
```

The **pandas**("panel data") is ideal for importing and extracting data from csv files. **Pandas** enables data to be represented as a virtual spreadsheet and as such shares many features with Microsoft® Excel:

```
confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
```

The Bayesian ridge polynomial regression model expressed in Python 3.8 is as follows

```
bayesian_grid = {'tol': tol, 'alpha_1': alpha_1, 'alpha_2': alpha_2, 'lambda_1': lambda_1, 'lambda_2': lambda_2}
```

```
bayesian = BayesianRidge(fit_intercept = False, normalize = True)
```

```
bayesian_search = RandomizedSearchCV(bayesian, bayesian_grid, scoring = 'neg_mean_squared_error', cv = 3,
return_train_score = True, n_jobs = -1, n_iter = 40, verbose = 1)
```

```
bayesian_search.fit(poly_X_train_confirmed, y_train_confirmed)
```

1. Load `TwelfthExampleProject.dproj` into Delphi.
1. Save project `TwelfthExampleProject.dproj` in a new folder, `FifteenthExample`, rename the project `FifteenthExampleProject.dproj` (**Save Project As**).
2. Save unit `TwelfthExampleUnit.pas` in new folder `FifteenthExample`, rename the unit `FifteenthExampleUnit.pas` (**Save As**).
3. If the package **SVGIconImageList VCL & FMX** is not already installed then install it as per `TwelfthExampleProject`.
4. Compile, link and run (**Shift+Ctrl+F9**) the project.
5. [Figure 80.85](#) shows `FifteenthExample.py` which is the Python program which performs Bayesian ridge polynomial regression. It takes about 15 seconds to produce output so be patient.
6. Click **Load** button and open `FifteenthExample.py`.
7. Click **Execute** button. The output produced is shown in [Figure 80.84](#).

On 26th October 2020 the number of worldwide confirmed Covid-19 cases was 43, 446,557.

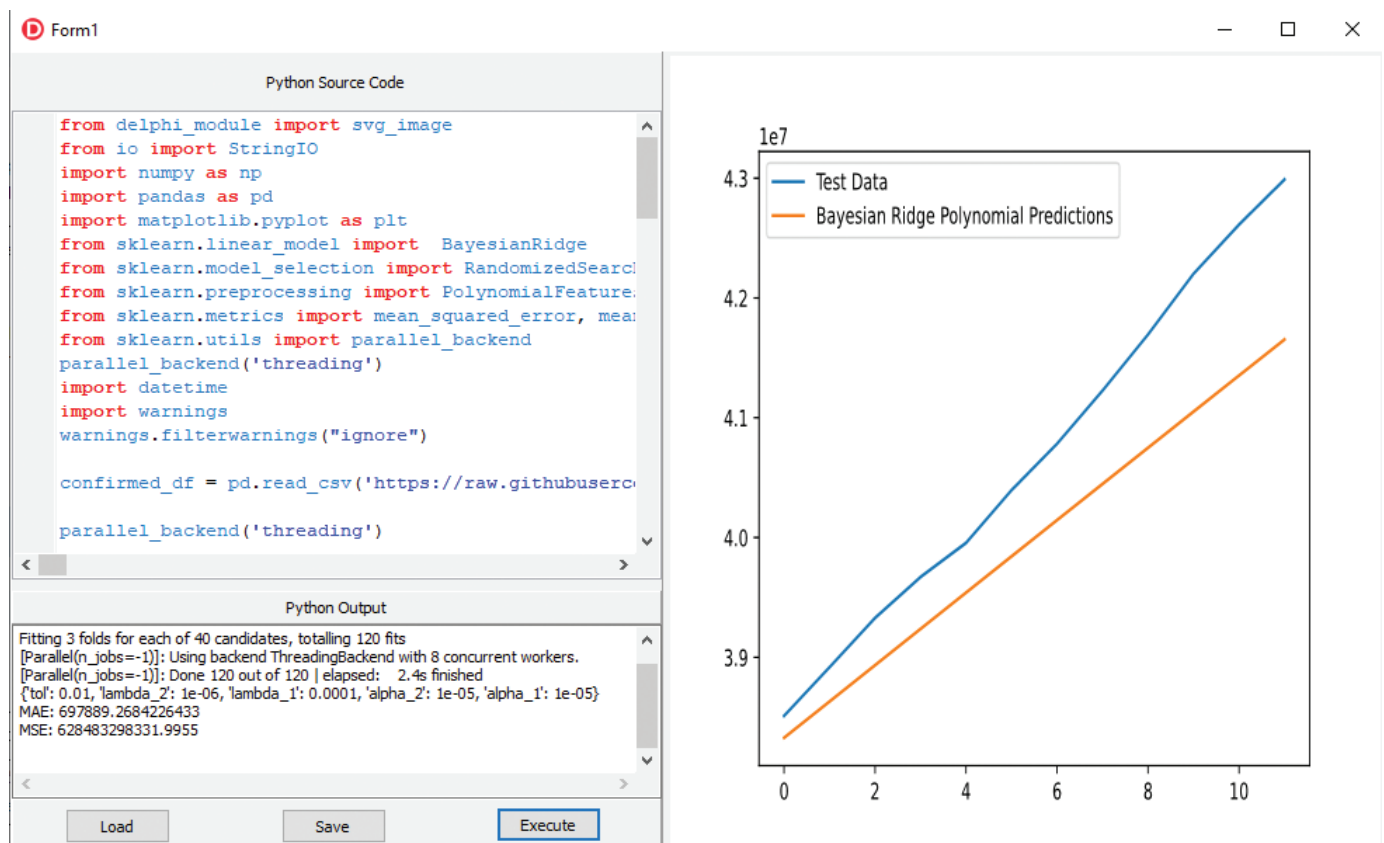


Figure 80.84 FifteenthExampleProject in execution

```

from delphi_module import svg_image
from io import StringIO
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import BayesianRidge
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.utils import parallel_backend
parallel_backend('threading')
import datetime
import warnings
warnings.filterwarnings("ignore")

confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')

parallel_backend('threading')
cols = confirmed_df.keys()
confirmed = confirmed_df.loc[:, cols[4]:cols[-1]]

dates = confirmed.keys()
world_cases = []

for i in dates:
    confirmed_sum = confirmed[i].sum()
    world_cases.append(confirmed_sum)

# window size
window = 7

days_since_1_22 = np.array([i for i in range(len(dates))]).reshape(-1, 1)
world_cases = np.array(world_cases).reshape(-1, 1)

days_in_future = 10
future_forecast = np.array([i for i in range(len(dates)+days_in_future)]).reshape(-1, 1)
start = '1/22/2020'
start_date = datetime.datetime.strptime(start, '%m/%d/%Y')
future_forecast_dates = []
for i in range(len(future_forecast)):
    future_forecast_dates.append((start_date + datetime.timedelta(days=i)).strftime('%m/%d/%Y'))

X_train_confirmed, X_test_confirmed, y_train_confirmed, y_test_confirmed = train_test_split(days_since_1_22[50:], world_cases[50:], test_size=0.05, shuffle=False)

# transform the data for polynomial regression
bayesian_poly = PolynomialFeatures(degree=5)
bayesian_poly_X_train_confirmed = bayesian_poly.fit_transform(X_train_confirmed)
bayesian_poly_X_test_confirmed = bayesian_poly.fit_transform(X_test_confirmed)
bayesian_poly_future_forecast = bayesian_poly.fit_transform(future_forecast)

# bayesian ridge polynomial regression
tol=[1e-4, 1e-3, 1e-2]
alpha_1=[1e-7, 1e-6, 1e-5, 1e-4]
alpha_2=[1e-7, 1e-6, 1e-5, 1e-4]
lambda_1=[1e-7, 1e-6, 1e-5, 1e-4]
lambda_2=[1e-7, 1e-6, 1e-5, 1e-4]
normalize = [True, False]
bayesian_grid = {'tol': tol, 'alpha_1': alpha_1, 'alpha_2': alpha_2, 'lambda_1': lambda_1, 'lambda_2': lambda_2}
bayesian = BayesianRidge(fit_intercept=False, normalize=True)
bayesian_search = RandomizedSearchCV(bayesian, bayesian_grid, scoring='neg_mean_squared_error', cv=3, return_train_score=True, n_jobs=-1, n_iter=40, verbose=1)
bayesian_search.fit(bayesian_poly_X_train_confirmed, y_train_confirmed)
print(bayesian_search.best_params_)
bayesian_confirmed = bayesian_search.best_estimator_
test_bayesian_pred = bayesian_confirmed.predict(bayesian_poly_X_test_confirmed)
bayesian_pred = bayesian_confirmed.predict(bayesian_poly_future_forecast)
print('MAE:', mean_absolute_error(test_bayesian_pred, y_test_confirmed))
print('MSE:', mean_squared_error(test_bayesian_pred, y_test_confirmed))
plt.plot(y_test_confirmed)
plt.plot(test_bayesian_pred)
plt.legend(['Test Data', 'Bayesian Ridge Polynomial Predictions'])
figfile = StringIO()
plt.savefig(figfile, format='svg')
figdata_svg = figfile.getvalue()
svg_image.SvgText = figdata_svg

```

Figure 80.85 FifteenthExample.py