LabVIEW-Python Integration

Hands-On Session, NIWeek 2019 with Danielle Jobe from VI Technologies

Table of Contents

Overview

Today we are going to familiarize ourselves with LabVIEW's Python Nodes. We will run through a few simple examples and then run a couple of machine learning/computer vision algorithms using OpenCV.

Software prerequisites

Your laptop in this session is already set up with:

- Python 3.6.8
  - OpenCV
  - NumPy

- LabVIEW 2018

For more information on setting up your laptop at home with the same requirements please see the "Getting Started on Your Own Computer" section at the bottom of this document.

Examples

We will go over the following examples:

- Hello World
- Adding two numbers
- Calculating the number of days since a date
- Calling a function that has multiple outputs
- Dictionary lookup with key-value pairs
- Facial recognition using OpenCV
- Corner detection using OpenCV

Before we Begin

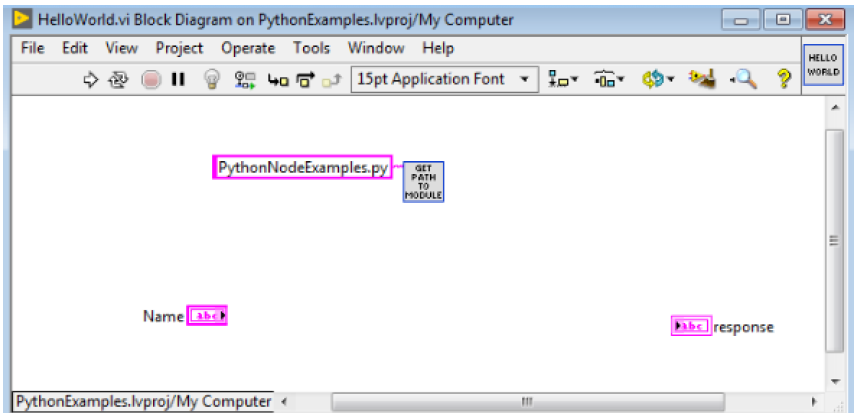
All of the examples for this session are found on the desktop under Desktop/LabVIEW2018_Python_Integration/lv_src/Workshop. As you might expect, the /Solutions folder contains all of these examples already completed, for reference. But looking in there would spoil all the fun!

All of these examples involve calling functions that are written in Python. This is not a session on how to write Python, so all of the functions that we will be calling are already written for you and will be located in a couple of python modules in the Desktop/LabVIEW2018_Python_Integration /py_src directory.

Example 1: Hello World – Strings

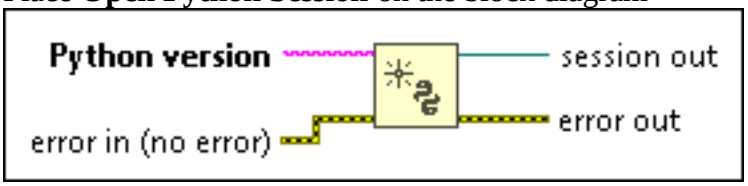
In this very simple example, we will learn how to use LabVIEW's Python Functions to call a HelloWorld function in python.

1. Open the PythonExamples.lvproj under Desktop/LabVIEW2018_Python_Integration/lv_src/Workshop
2. Open the HelloWorld.vi
3. The block diagram should look like this:

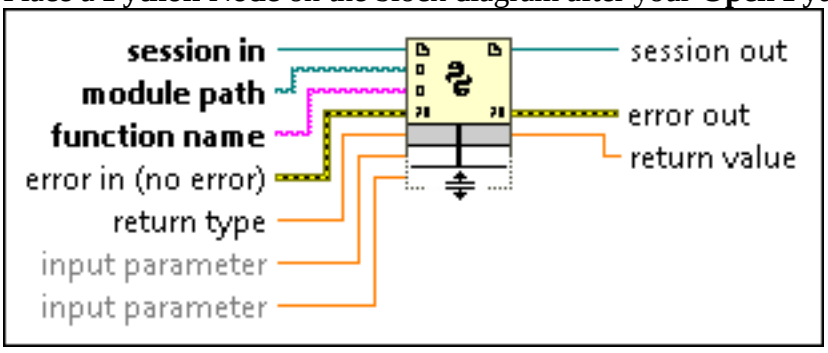


It contains a string control called "Name", a string indicator called "response" and a subVI called "GetPathToModule" with the module name input wired and set to PythonNodeExamples.py. This function will return the path to the python module located under py_src.

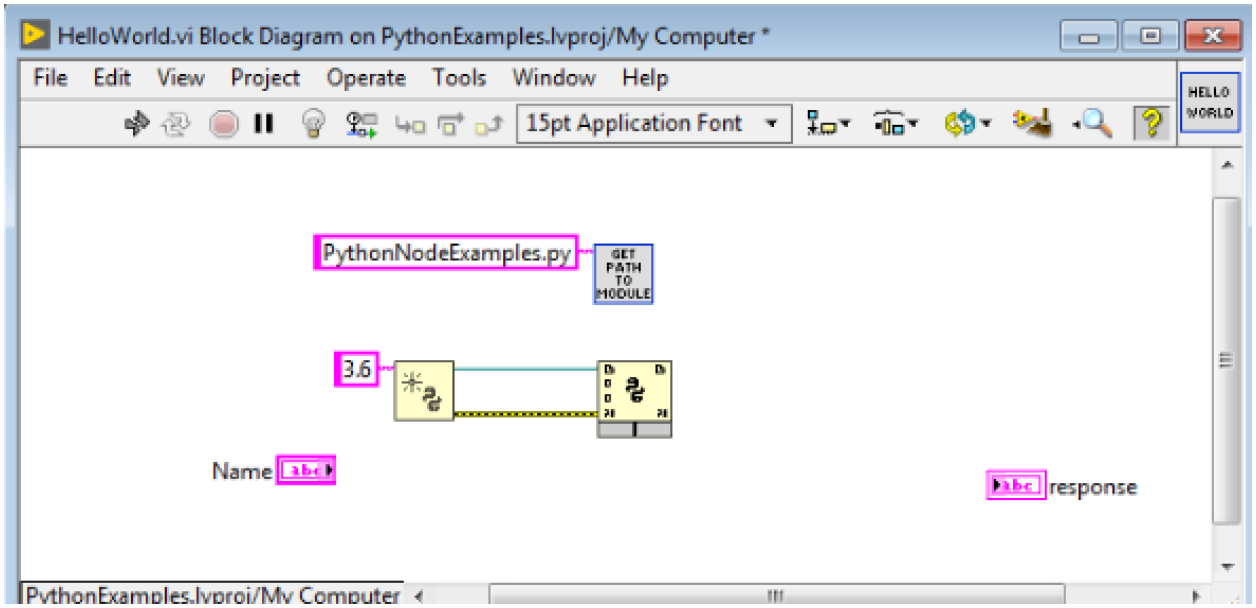
4. On the **Functions** palette go to **Connectivity > Python**. You can pin this menu for easy access.
5. Place **Open Python Session** on the block diagram



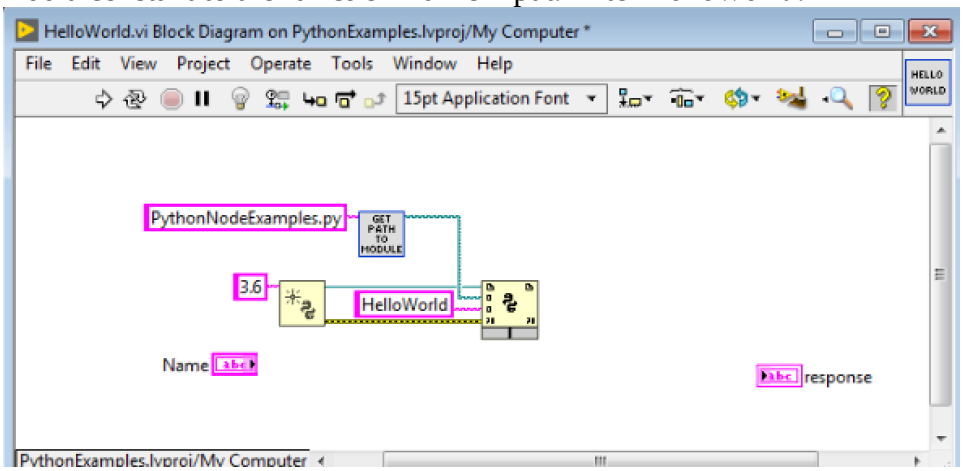
6. For the **Python version** input add a string constant and type 3.6 in it. This is the version of Python that is installed on this computer. You can also be more specific and type 3.6.8, it doesn't matter.
7. Place a **Python Node** on the block diagram after your **Open Python Session**.



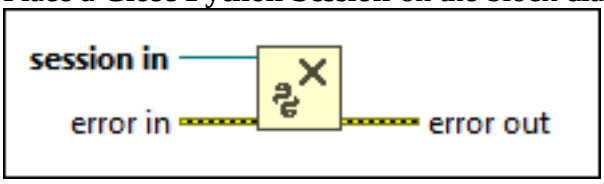
8. Wire the **session out** output from the **Open Python Session** function to the **session in** input on the **Python Node.**



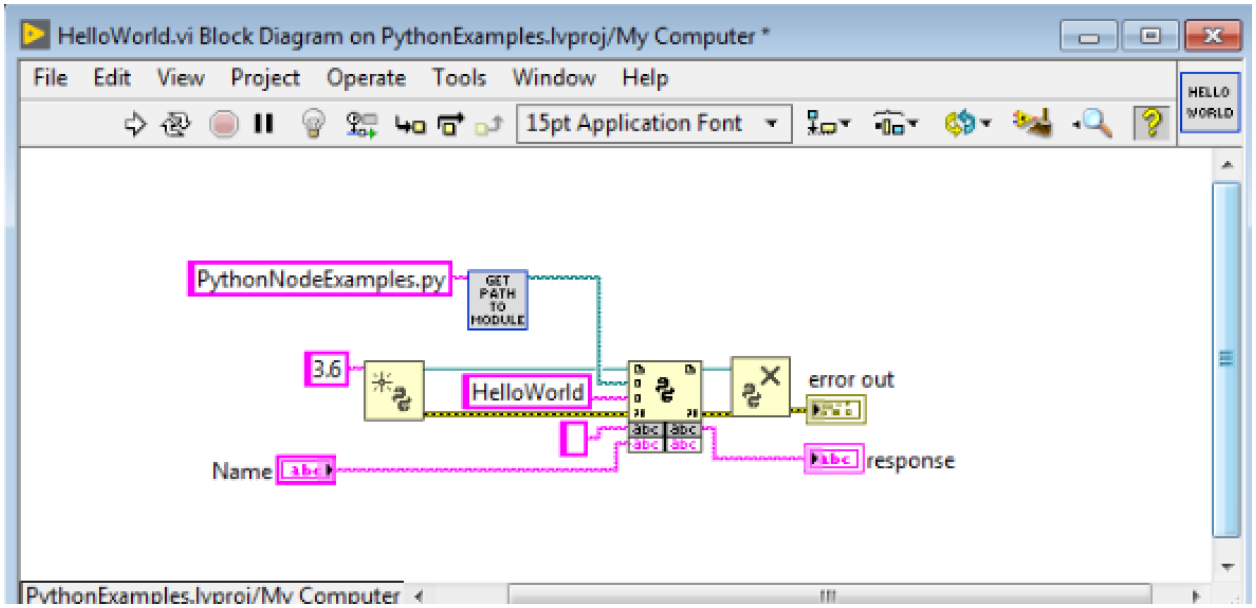
9. Wire the output of the "GetPathToModule" vi to the **module path** input of the python node.
10. Add a constant to the **function name** input. Enter *HelloWorld*.



11. Add a string constant to the **return type** input. This function returns a string.
12. Connect the "response" indicator to the **return value** output.
13. Drag the bottom border of **python node** down to expose one **input parameter** input. Connect the "Name" String control to this input.
14. Place a **Close Python Session** on the block diagram after your **Python Node**



15. Wire the **session out** output from the **Python Node** to the **session in** input of **Close Python Session**.



16. Go to the Front Panel and write your name in the "Name" string control.
17. Run the VI – it should return "Hello [Your Name]!"

Example 2: Adding Two Numbers

Now that you know how to put the Python Functions on the block diagram and how to connect them and their inputs and outputs, we will expand this knowledge with a few different data types. The first data type we will see is a numeric.

Numeric Type 1: I32

1. Open the Add.vi
2. Using what you learned from the previous example, **open a Python Session**, add a **Python Node**, and then **Close the Python Session**.
3. We will still be calling the same module, but this time instead of "HelloWorld" we will be calling the "Add" function.
4. Add an I32 constant to the **return type** input.
5. Expand the **Python Node** to expose two **input parameters**. Wire the two I32 controls "A" and "B" to those inputs.
6. Wire the "C" indicator to the **return value** output of the python node.
7. Go to the Front Panel and try adding some different integers.

```
Data type conversion:
LabVIEW: I32 → Python: Int
```

Numeric Type 2: Dbl

1. Change all your data types in and out on the previous example to doubles.
2. On the front panel change one of the inputs to a fractional number like 3.14159
3. Run it and see that without changing the function python adapts the data type and will return a fractional number.

4. What is the maximum number of digits past the decimal that the python function will return?

```
Data type conversion:
LabVIEW: DBL → Python: Float
```

Example 3: Calculating the Number of Days Since a Date – Clusters as Inputs

We have now tried strings and numerics, now let's try calling a python function that takes a cluster as an input.

1. Open CalculateDaysSince.vi from the project
2. Using what you learned from the first example, **open a Python Session**, add a **Python Node**, and then **Close the Python Session**.
3. We will still be calling the same module, but this time we will be calling the "CalculateDaysSince" function.
4. Add an I32 constant to the **return type** input.
5. Expand the **Python Node** to expose one **input parameter**. Wire the "Date" cluster to that input.
6. Wire the "days since date" indicator to the **return value** output.
7. Go to the Front Panel and enter a date. Run the VI to find out how many days have elapsed since that date. Try a few dates like your birthday, your anniversary, your dog's birthday, etc.

```
Data type conversion:
LabVIEW: Enum (U16) → Python: Int
LabVIEW: Cluster → Python: Tuple
```

Example 4: Returning Multiple Values – Clusters as Outputs, Arrays as Inputs

We have seen using clusters as inputs and how Python handles them, now we will see that when you call a function that returns multiple outputs the Python node will return a cluster with the outputs. This example also uses an array as an input, which is handled in Python as a list.

1. Open MaxAndMin.vi from the project
2. Using what you learned from the first example, **open a Python Session**, add a **Python Node**, and then **Close the Python Session**.
3. We will still be calling the same module, but this time we will be calling the "maxAndMin" function.
4. Add a cluster constant to the **return type** input. Add two doubles to that cluster. You can also just create a constant from the "Max and Min" indicator.

5. Expand the **Python Node** to expose one **input parameter**. Wire the "Numbers" control to that input.
6. Wire the "Max and Min" indicator to the **return value** output
7. Go to the Front panel and expand the array input to show a few elements. Populate the array with various numbers.
8. Run the VI to see the max and min of the array.

```
Data type conversion:
LabVIEW: Array → Python: list
```

Example 5: Dictionary Lookup

Many applications use key value pairs to store data in a dictionary or hash table. In this example we will take an array of clusters containing key-value pairs and look up a value by its key. You might have done this in LabVIEW either using a for loop iterating over elements in an array, a variant dictionary, or with associative arrays. Here we will call a python function that will perform the lookup on the array for us.

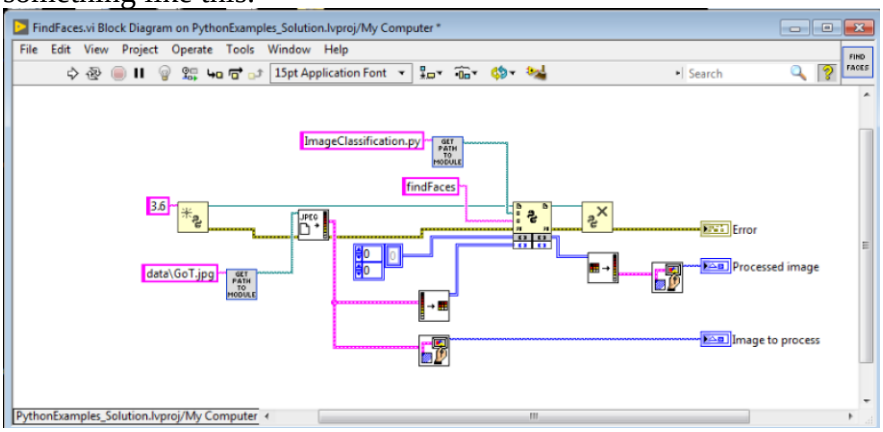
1. Open the "GetValueForKey.vi" from the project.
2. Using what you learned from the first example, **open a Python Session**, add a **Python Node**, and then **Close the Python Session**.
3. We will still be calling the same module, but this time we will be calling the "getValueForKey" function.
4. Add a double numeric constant to the **return type** input.
5. Expand the **Python Node** to expose two **input parameters**. Wire the "Dictionary" array control to the first **input parameter**. Wire the "Key" input to the second **input parameter**. This will be the key that we are looking up in the dictionary.
6. The "value" indicator to the **return value** output.
7. Go to the Front panel and populate the array with various key names and values.
8. Enter a key name and run the VI to see the value associated with that key.

Example 6: OpenCV Facial Detection

A great use case for calling python functions within LabVIEW is performing computer vision algorithms on images using OpenCV. For this example, we will follow a standard OpenCV example and call a function that uses a Haar Cascades machine learning algorithm to do facial recognition on an image. This example will show you how to pass an image to Python so that you can perform OpenCV functions on it and then read it back from Python. There are several ways you can accomplish this, like passing file paths or even sending memory locations. For this example, we will directly pass the unflattened 24-bit pixmap.

1. Open the "FindFaces.vi" from the project.
2. Using what you learned from the first example, **open a Python Session**, add a **Python Node**, and then **Close the Python Session**.
3. This time we will be calling a new module, as you'll see "ImageClassification.py" wired to the "GetPathToModue" subVI. We will be calling a function called "findFaces".

4. Add an array constant to the **return type** input. Add a U32 numeric constant to that array. Add one more dimension to the array so it is a 2D array. This will represent our unflattened 24-bit pixmap that we will receive back from Python.
5. To find the faces on an image we will need an image. Add the "Read JPEG File.vi" (Functions > Graphics & Sound > Graphics Formats) to the block diagram and add a path control or constant to the input. Point that path to the jpg file at Desktop/LabVIEW2018_Python_Integration /py_src/data/GoT.jpg. You can make this easier by adding another copy of the GetPathToModule vi and add "data\GoT.jpg" to the string input instead of the module name and wiring the path output to the **path to JPEG file** input.
6. Add an "Unflatten Pixmap.vi" (Functions > Graphics & Sound > Graphics Formats) to the block diagram and wire the **image data** output of the "Read JPEG File.vi" to the **image data** input of the "Unflatten Pixmap.vi."
7. Expand the **Python Node** to show one input. Wire the **24-bit pixmap** output of "Unflatten Pixmap.vi" to that input.
8. To display the image that will be processed use a "Draw Flattened Pixmap" (Functions > Graphics & Sound > Picture Functions) and connect it to the "Image to process" indicator
9. All that we will need to do to view the output of the python node is call the "Flatten Pixmap.vi" and then "Draw Flattened Pixmap." Connect the output of the "Draw Flattened Pixmap" to the "Processed image" indicator. You should end up with something like this:



10. Go to the Front panel and run the VI to see how the faces are detected by the algorithm.

Example 7: OpenCV Corner Detection

For the last example we are going to do a slight modification of the previous example and use OpenCV to perform Harris Corner Detection.

1. Open the "FindCorners.vi" from the project. Leave the "FindFaces.vi" open.
2. Copy the code from the "FindFaces.vi" – this will make things simpler since passing the image is the same.
3. Instead of referencing the image at data\GoT.jpg change it to get the image at data\cubes.jpg. This will find edges on a greyscale image of cubes.
4. This function will return multiple outputs – it will return the image as an unflattened 24-bit pixmap, and the "Harris corners" and "Refined corners" indicators.

5. Using what you have learned in the previous examples, add a return type to the python node that will contain those data types, and then show them using the indicators already provided on the "FindCorners.vi"
6. Once you have it working properly you should see a processed image with the pixels representing the corners colored in green and red. The arrays will give you the pixel coordinates of the corners highlighted in the images. The Harris corners are shown in red, and the refined corners are shown in green.

Getting Started on Your Own Computer

I hope you found this demo useful. If you would like to repeat it at home, show it to a colleague, or play with it some more, all of the files for this demonstration can be found on GitHub at https://github.com/DanielleJobe/LabVIEW2018-Python-Integration-Example

Prerequisites

LabVIEW 2018 or newer
Python 3.6 (3.6.8 is recommended)
pip 19 or newer

Installation

Clone or download the repo
Install the python packages using pip:

```
cd [repo clone location]/py_src
pip install -r requirements.txt
```

Note: You will need to have configured pip and python in your computer's system environmental variables

Resources:

https://www.python.org/ftp/python/3.6.8/python-3.6.8-amd64.exe - Python 3.6.8 32-bit installer for windows (download link)

http://www.ni.com/product-documentation/54295/en/ - Installing python for calling python code

https://www.computerhope.com/issues/ch000549.htm - Setting system path