Integrating LabVIEW and Python

NATIONAL INSTRUMENTS™

Why LabVIEW?

- Seamless hardware integration
- Easy GUI development
- Rapid iteration & development
- Built-in parallelism
- Software engineering & productivity tools
- LabVIEW ecosystem
- Real-Time
- FPGA
- …

Why Python?

- Thousands of mature, open source packages

NumPy

matplotlib

pandas

SciPy.org

TensorFlow

plotly

OpenCV

Flask

SQLAlchemy

scikit learn

Requests
http for humans

NATIONAL INSTRUMENTS

Examples of using LabVIEW and Python together

Take advantage of thousands of Python packages

NumPy

matplotlib

pandas

TensorFlow

OpenCV

SciPy.org

plotly

scikit learn

Requests
http for humans

SQLAlchemy

Flask

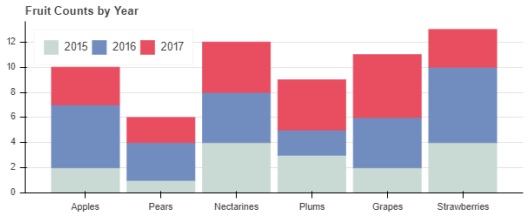
NATIONAL INSTRUMENTS

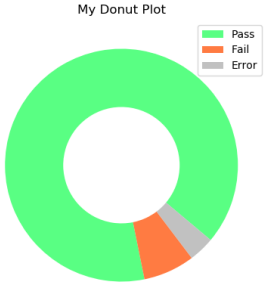
Take advantage of thousands of Python packages

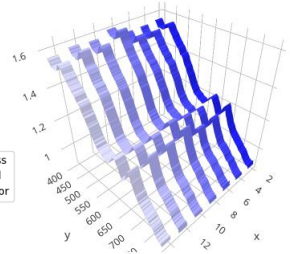
- **NumPy**: the fundamental package for scientific computing with Python.
- **SciPy**: provides many user-friendly and efficient numerical routines.
- **scikit-learn**: machine learning in Python.
- **scikit-image**: image processing in Python.
- **matplotlib**: a Python 2D plotting library.
- **iPython / Jupyter**: interactive computing with Python.
- **pandas**: high-performance, easy-to-use data structures & data analysis tools
- **requests**: Python HTTP for humans.
- **SQLAlchemy**: database abstraction library.
- **Tensorflow:** machine learning platform

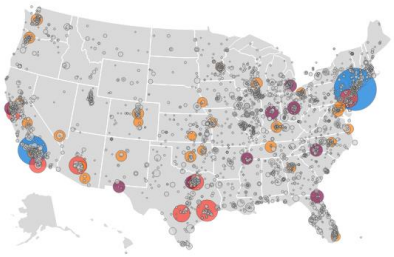
Integrate with anything that has a Python API

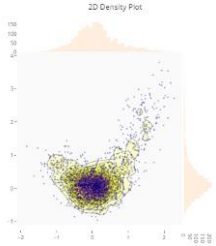
Leverage additional data visualizations from Python

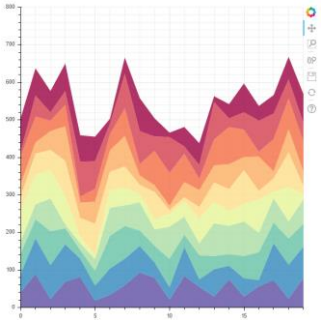


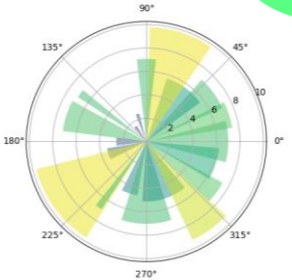


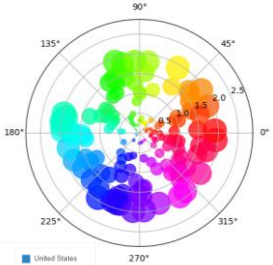


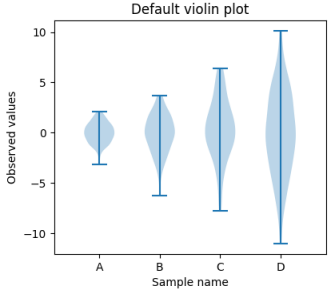


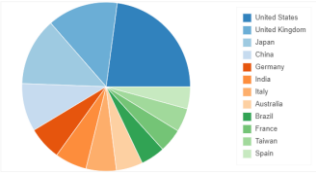


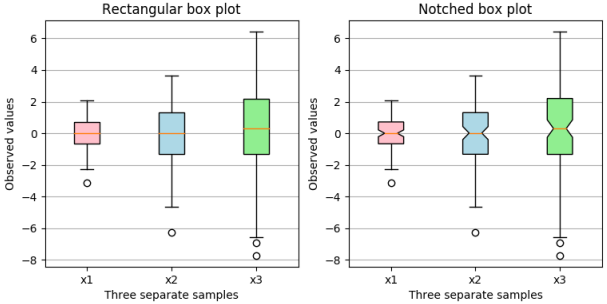


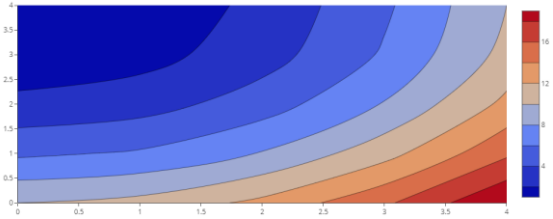


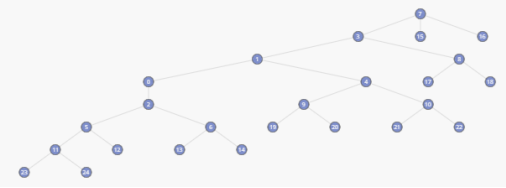












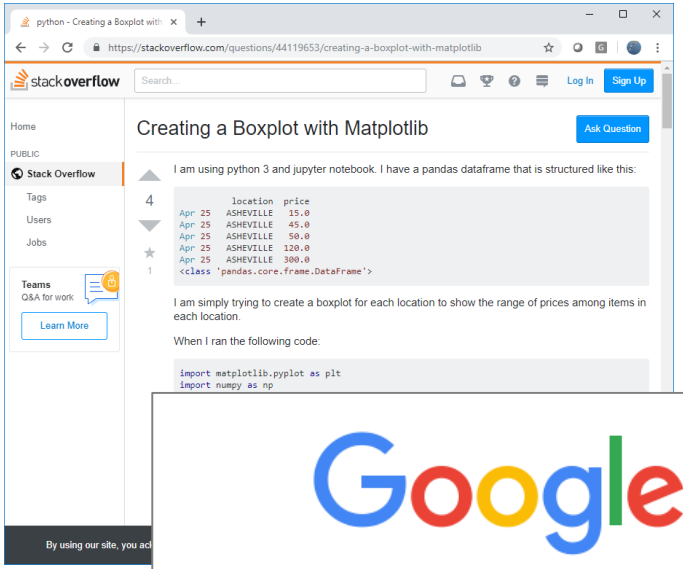
Reuse existing Python scripts

Co-workers

```python
def create_testresults_stackedbargraph (pass_data, fail_data, error_data, target_image_filepath, labels, title):
    ind = np.arange(len(pass_data))   # the x locations for the groups
    width = 0.35  # the width of the bars: can also be len(x) sequence

    plt.bar(ind, error_data, width, color='darkgrey')
    plt.bar(ind, fail_data, width, bottom=error_data, color='lightcoral')
    plt.bar(ind, pass_data, width, bottom=fail_data, color='mediumseagreen')
    # You can find a list of colors here: https://python-graph-gallery.com/python-colors/

    plt.ylabel('Number of Tests')
    plt.xticks(ind, labels)
    plt.title(title)
    plt.legend(['Error', 'Fail', 'Pass'])
    plt.savefig(target_image_filepath.replace('\\','/')) # replace all instances of '\' with '/'
```

```python
def create_donutplot (data, target_image_filepath, labels, title, colors):
    # Labels is a string list.  Data is a numeric list. Title is string. Colors is a string list of colors.
    fig, ax = plt.subplots(figsize=(5, 5), subplot_kw=dict(aspect="equal"))
    wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40, colors=colors)
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),
              bbox=bbox_props, zorder=0, va="center")

    for i, p in enumerate(wedges):
        ang = (p.theta2 - p.theta1)/2. + p.theta1
        connectionstyle = "angle,angleA=0,angleB={}".format(ang)
        kw["arrowprops"].update({"connectionstyle": connectionstyle})

    ax.set_title(title)
    ax.legend(labels)
    plt.savefig(target_image_filepath.replace('\\','/')) # replace all instances of '\' with '/'
    # plt.show()
    plt.close()
```

Internet



Methods of calling Python from LabVIEW

Methods

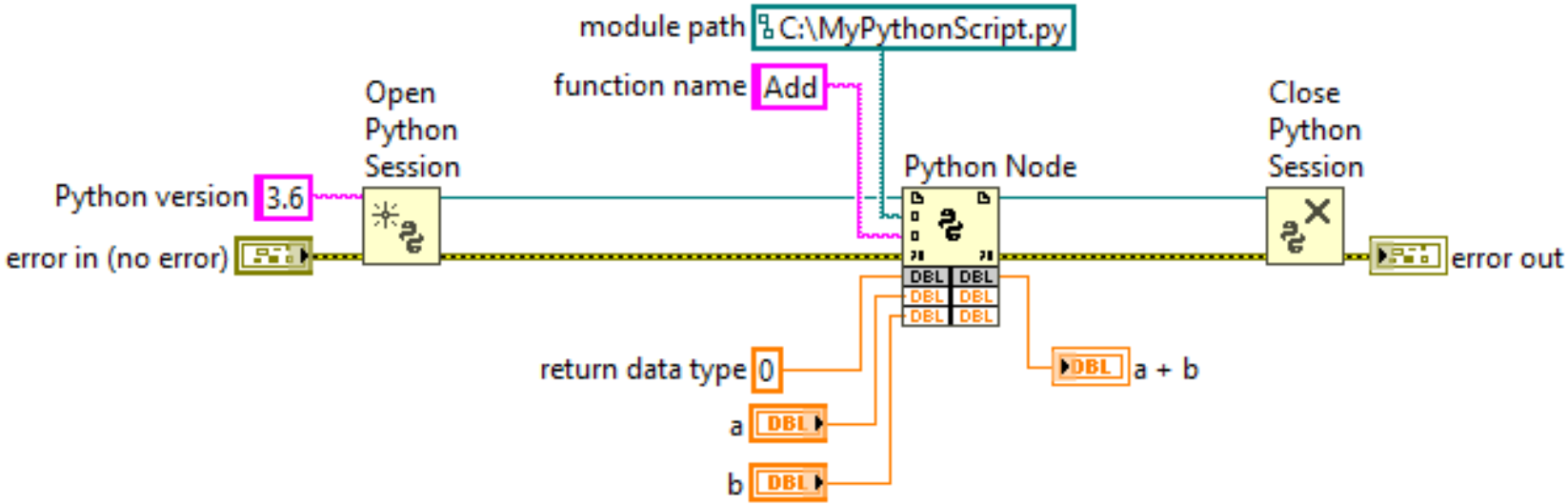
- Python Node
- Python Integration Toolkit for LabVIEW by Enthought

Methods of calling Python from LabVIEW

Python Node

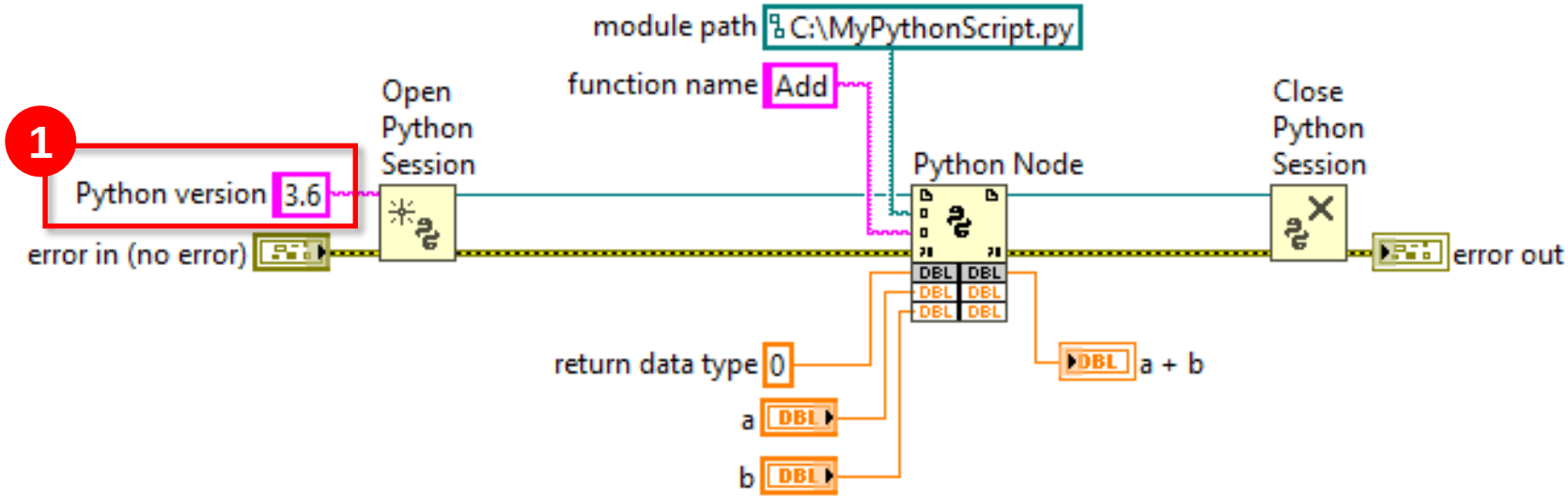
Python Node

- Supported in LabVIEW 2018 and later



Python Node

- Open session & specify Python version

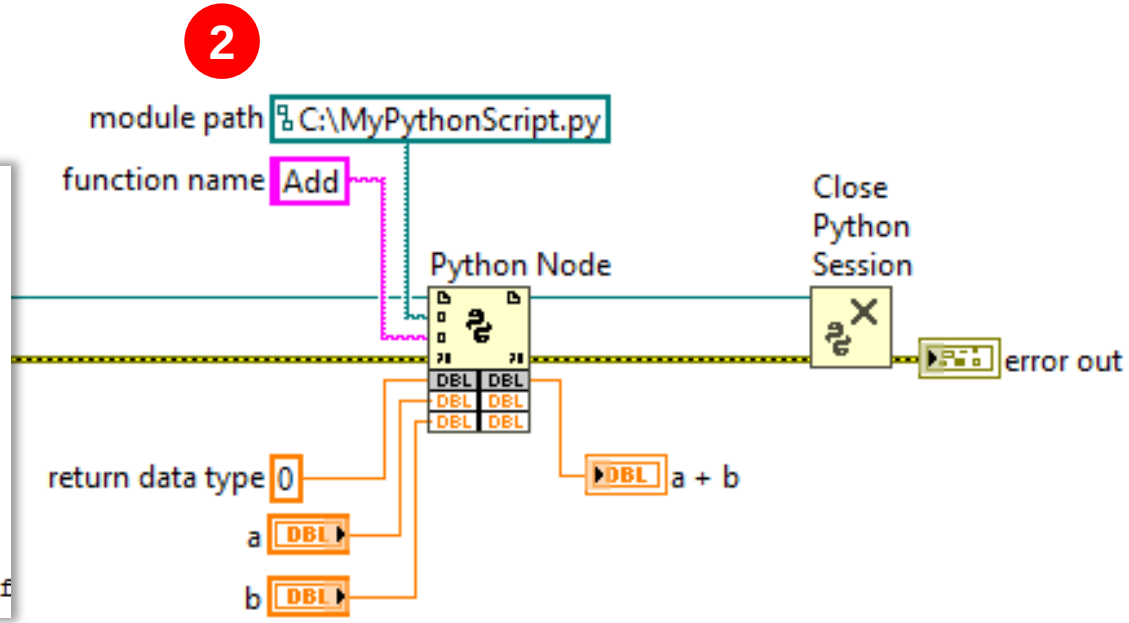


Python Node

- Call Python module (.py) and function

```
C:\MyPythonScript.py
```

```python
import math

def Add(a, b):
    return a+b;

def ConcatenateStrings(str1, str2):
    return str1 + str2;

def EuclideanDistance(point1, point2):
    xDiff = point1[0] - point2[0];
    yDiff = point1[1] - point2[1];
    return math.sqrt(xDiff*xDiff + yDiff
```



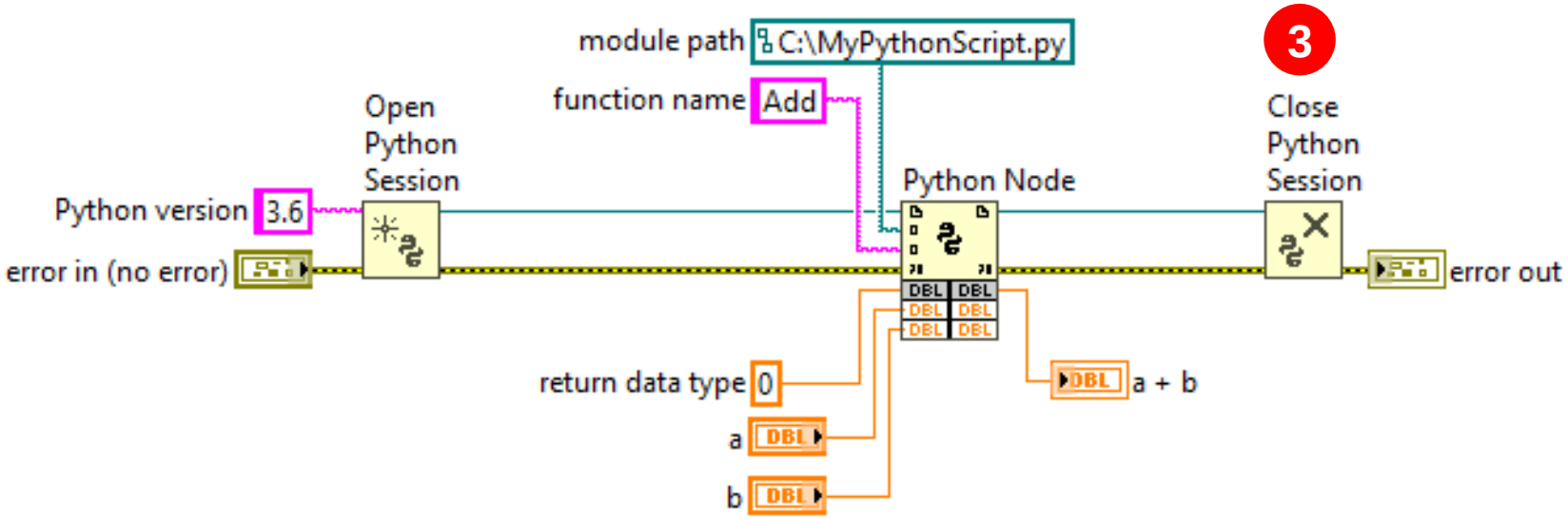
Python Node – Supported Data Types

LabVIEW Data Type	Python Data Type
Integers	`int`
SGL, DBL	`float`
String	`str`
Boolean	`bool`
Array	`list or NumPy array`
Cluster	`tuple`

Note LabVIEW 2019 adds support for NumPy arrays.

Python Node

- Close the session



Python Node – Getting Started

1. Install LabVIEW 2018 or later.

2. Install Python.
   - Install supported version (2.7, 3.6)
   - Install same bitness (32-bit, 64-bit) as LabVIEW

3. Locate your Python script and function
   - Install any Python packages required by the Python script

4. Write/Run LabVIEW VI that calls Python script/function using Python Node

Python Node – Getting Started

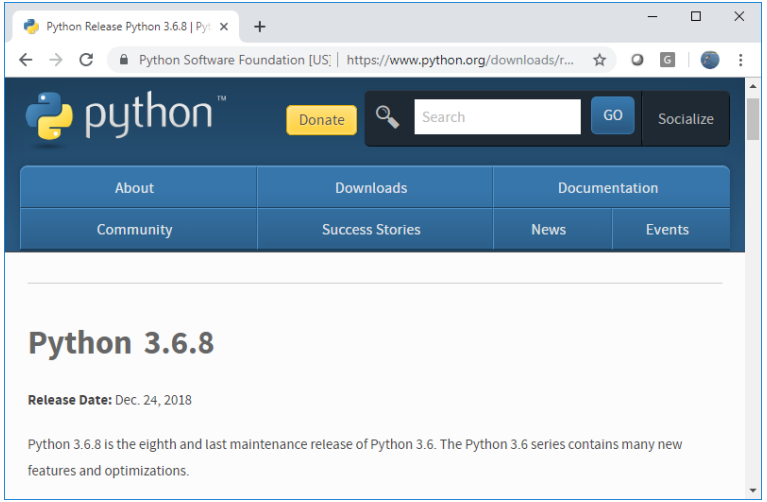
1. Install LabVIEW 2018 or later.

2. Install Python.
   - Install supported version (2.7, 3.6)
   - Install same bitness (32-bit, 64-bit) as LabVIEW

3. Locate your Python script and function
   - Install any Python packages required by the Python script

4. Write/Run LabVIEW VI that calls Python script/function using Python Node

https://www.python.org/



Python 3.6 https://www.python.org/downloads/release/python-368/
Python 2.7 https://www.python.org/downloads/release/python-2716/

NATIONAL INSTRUMENTS

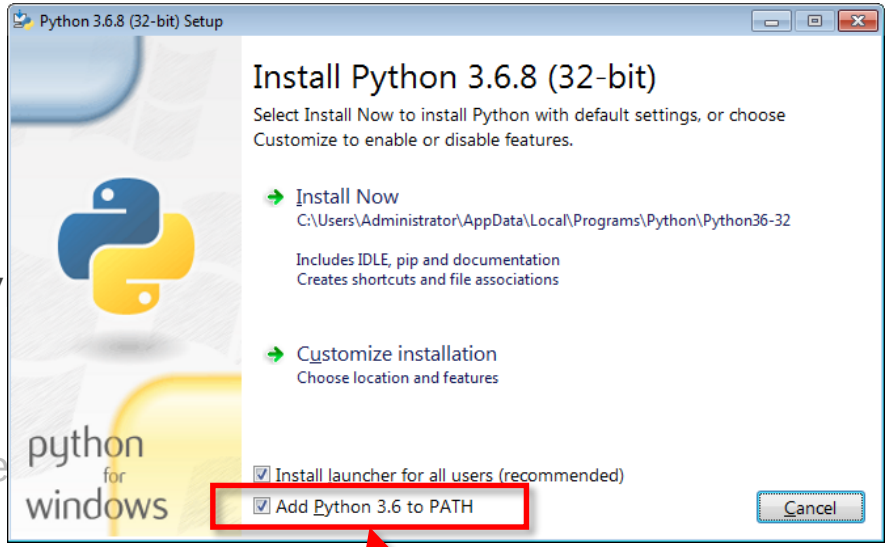
Python Node – Getting Started

1. Install LabVIEW 2018 or later.

2. Install Python.
   - Install supported version (2.7, 3.6)
   - Install same bitness (32-bit, 64-bit) as LabV

3. Locate your Python script and function
   - Install any Python packages required by the Python script

4. Write/Run LabVIEW VI that calls Python script/function using Python Node



If using Python 3.6, enable this checkbox in the installer

NATIONAL INSTRUMENTS

Python Node – Getting Started

1. Install LabVIEW 2018 or later.

2. Install Python.
   - Install supported version (2.7, 3.6)
   - Install same bitness (32-bit, 64-bit) as LabVIEW

3. Locate your Python script and function
   - Install any Python packages required by the Python script

4. Write/Run LabVIEW VI that calls Python script/function using Python Node

MyPythonScript.py

```
import sys
import cv2
import numpy as np
import os.path as op
```

Command Prompt

```
C:\Python36\Scripts>pip install numpy
```

NATIONAL INSTRUMENTS

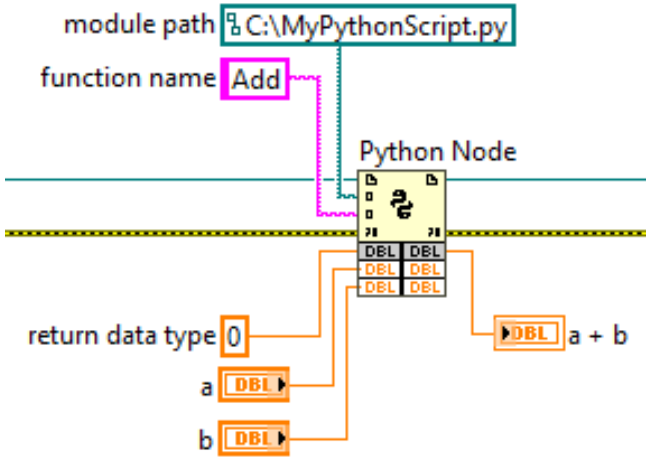
Python Node – Getting Started

1. Install LabVIEW 2018 or later.

2. Install Python.
   - Install supported version (2.7, 3.6)
   - Install same bitness (32-bit, 64-bit) as LabVIEW

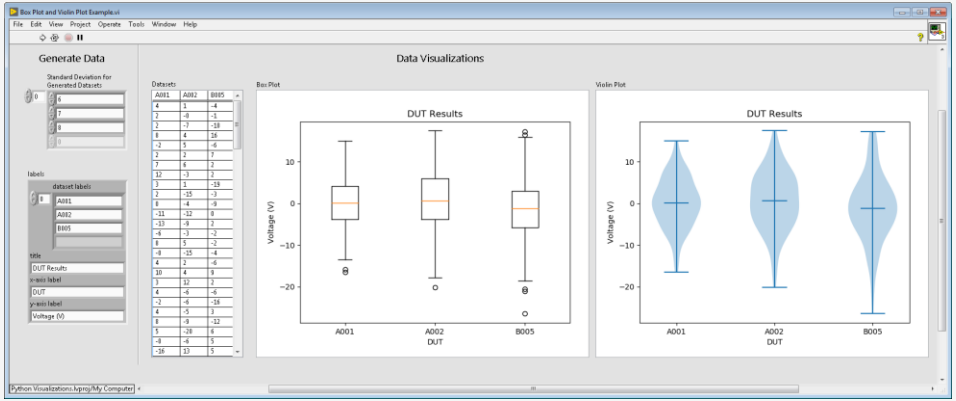
3. Locate your Python script and function
   - Install any Python packages required by the Python script

4. Write/Run LabVIEW VI that calls Python script/function using Python Node

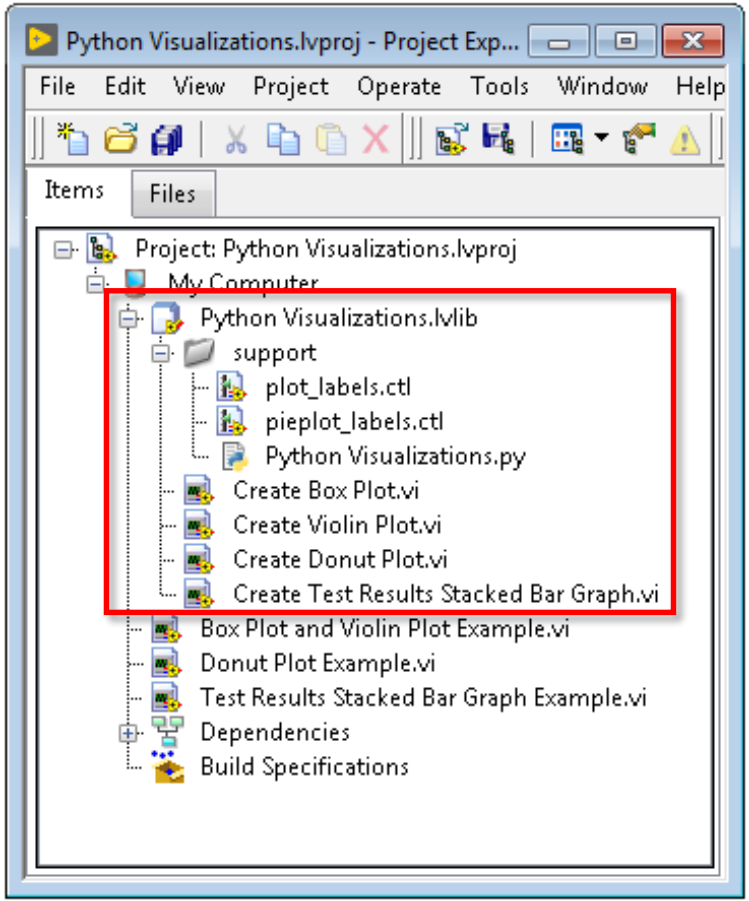


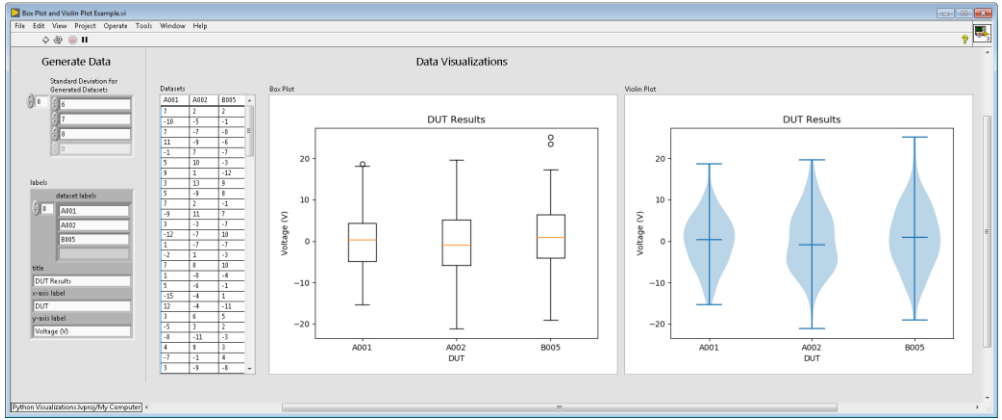
NATIONAL INSTRUMENTS

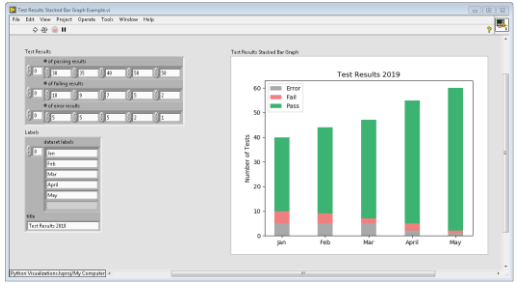
Python Node Demos

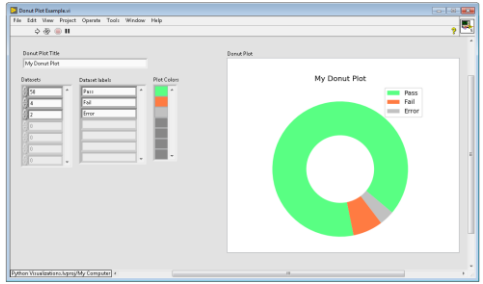


Reuse Python Visualizations (matplotlib) Demo

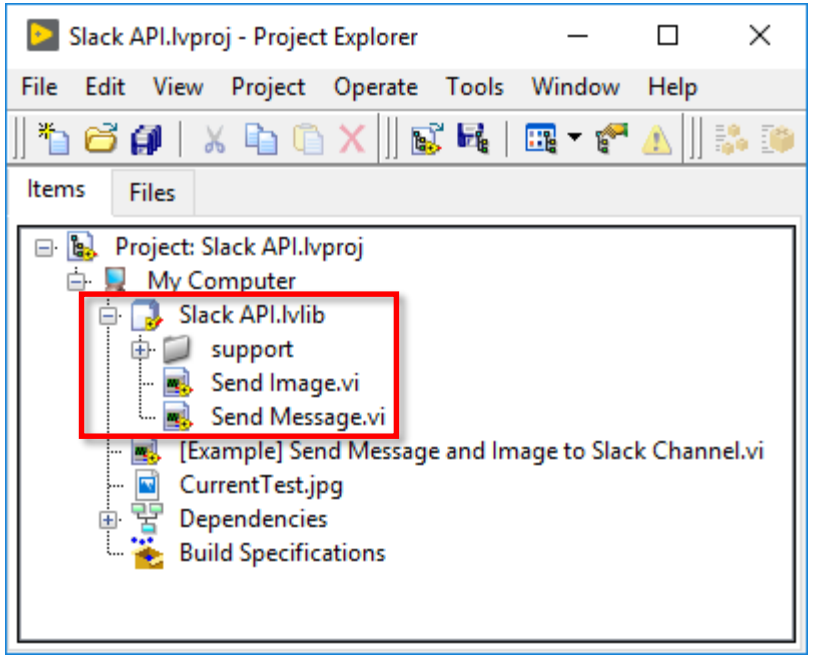


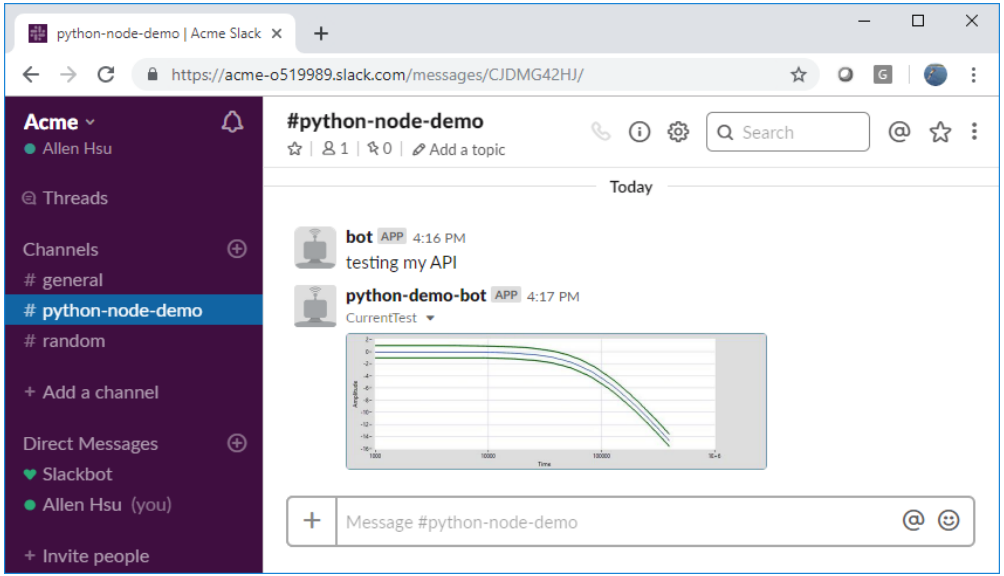






Reuse Python API for Slack (Demo)

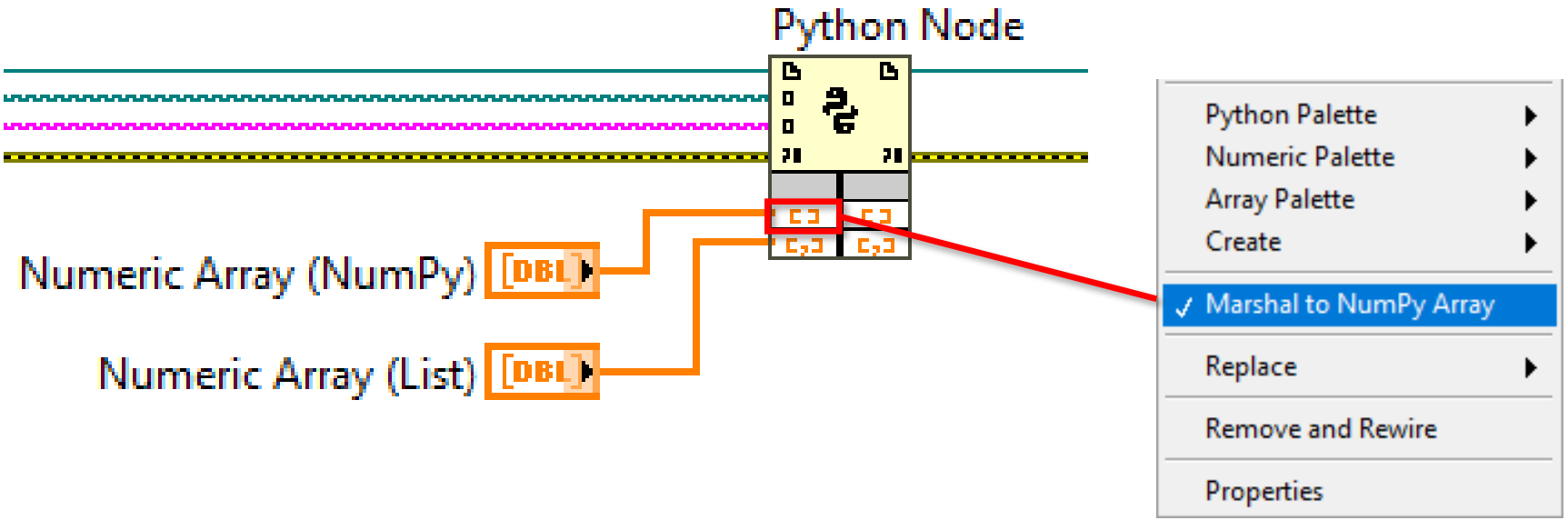




NATIONAL INSTRUMENTS

Python Node – Tips

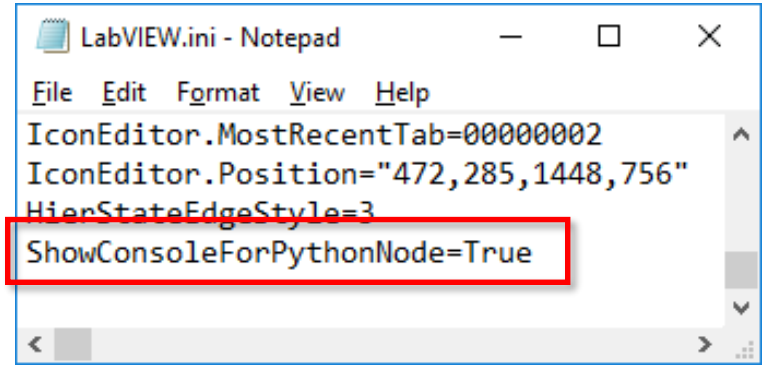
- If Python function has a NumPy array parameter, then marshal your LabVIEW numeric array as a NumPy array (instead of a list)

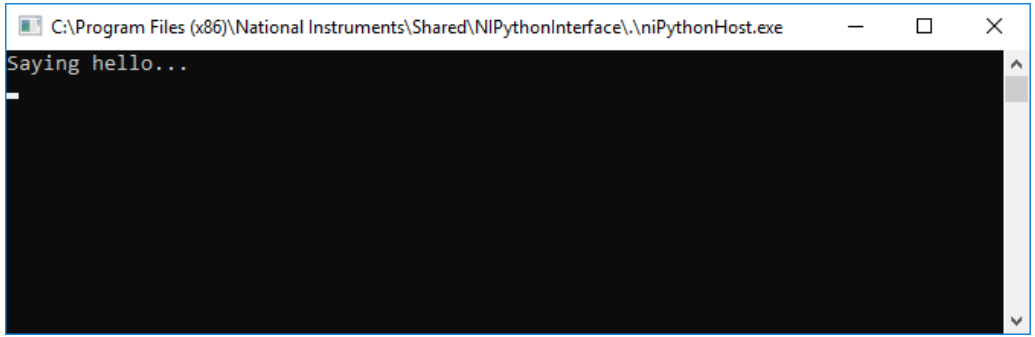


LabVIEW 2019*

Python Node – Tips

- Debugging
  - Add token to `<labview>\LabVIEW.ini` to show the Python console while Python Node is running

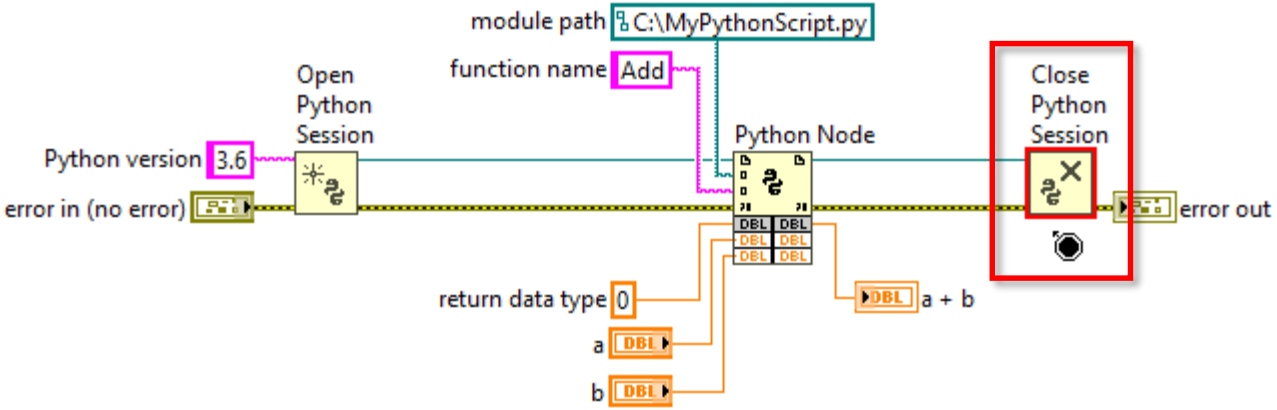




Python Node – Tips

- Debugging
  - Add token to `<labview>\LabVIEW.ini` to show the Python console while Python Node is running

Tip Place a breakpoint on the Close Python Session to keep the console window open



Python Node – Tips

- Read/write to global variable by using a wrapper function

Global variable

Wrapper function

Wrapper function

```
my_global_variable = "abc"

def read_my_global_variable():
    global my_global_variable
    return my_global_variable

def write_my_global_variable(value):
    global my_global_variable
    my_global_variable = value
```

Python Node – Tips

- Installing a set of packages
  - Use `pip freeze > requirements.txt` to generate file that lists installed packages.

```
C:\Users\ahsu\AppData\Local\Programs\Python\Python36-32\Scripts>pip freeze > C:\Users\Pub
lic\Documents\requirements.txt
```

requirements.txt ...

File Edit Format View Help

```
aiohttp==3.5.4
async-timeout==3.0.1
attrs==19.1.0
certifi==2019.3.9
cffi==1.12.3
chardet==3.0.4
cycler==0.10.0
idna==2.8
```

  - Use `pip install -r requirements.txt` to install the list of installed packages

```
C:\Users\ahsu\AppData\Local\Programs\Python\Python36-32\Scripts>pip install -r C:\Users\P
ublic\Documents\requirements.txt
```

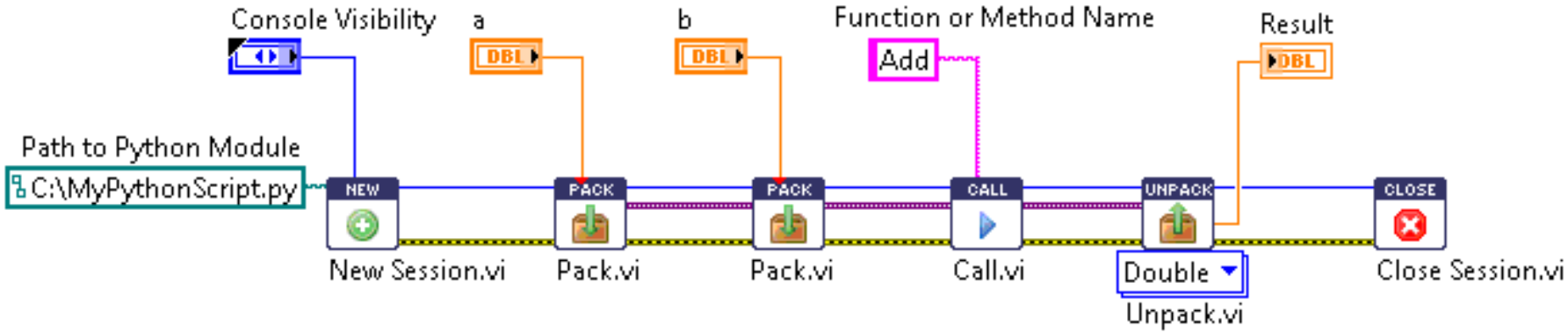
NATIONAL INSTRUMENTS

Methods of calling Python from LabVIEW

Python Integration Toolkit
(By Enthought)

Python Integration Toolkit

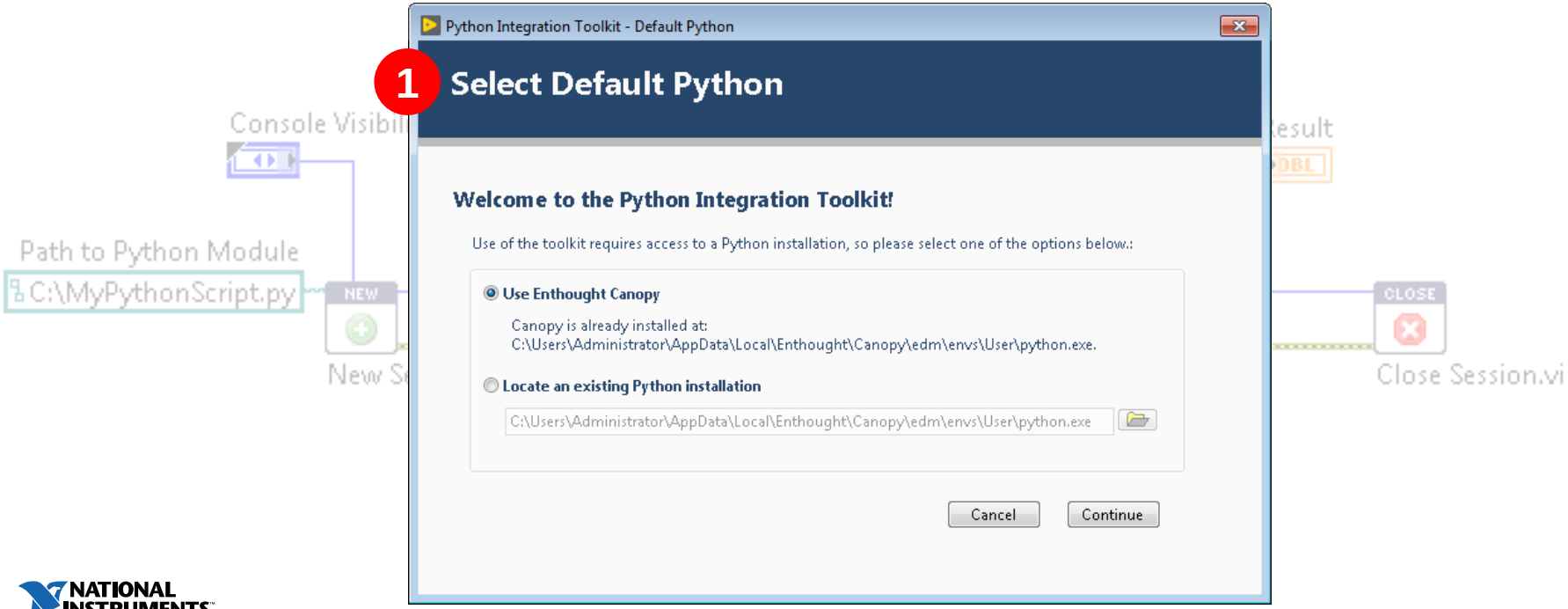
- Supported in LabVIEW 2015 and later
- Price: $749



NATIONAL INSTRUMENTS

Python Integration Toolkit

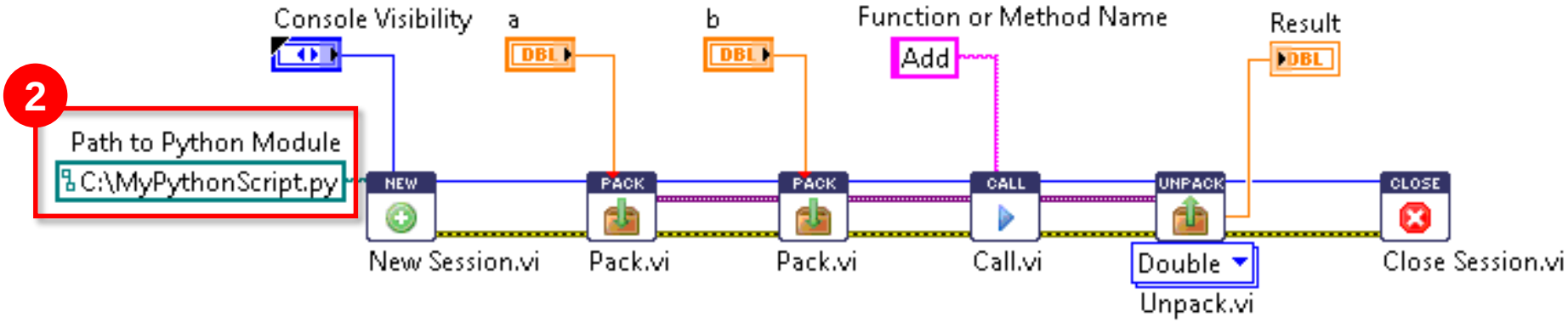
- Specify which Python installation to use



NATIONAL INSTRUMENTS

Python Integration Toolkit

- Open session & specify Python module (.py)

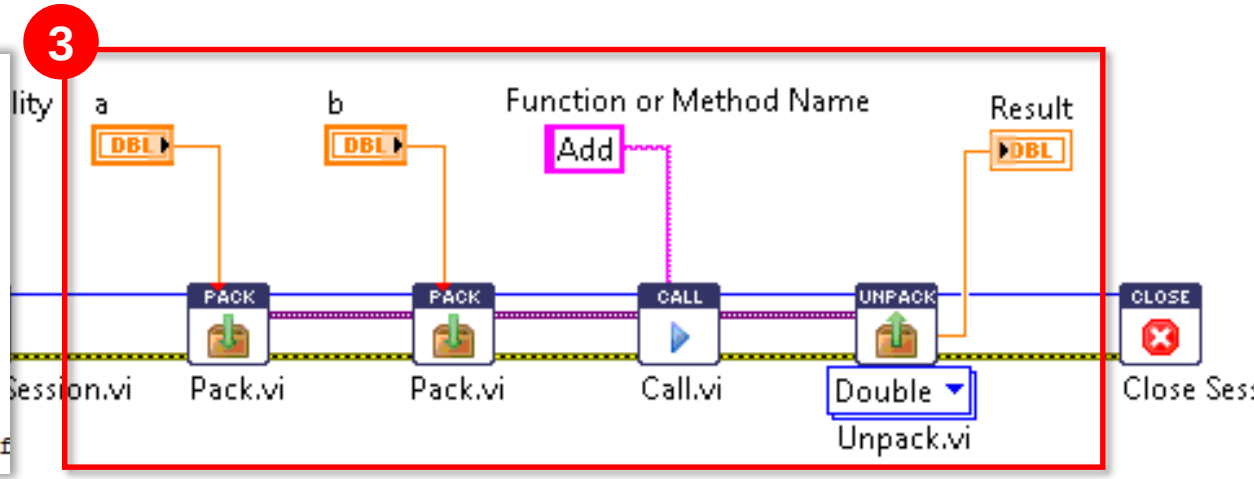


Python Integration Toolkit

- Call Python function

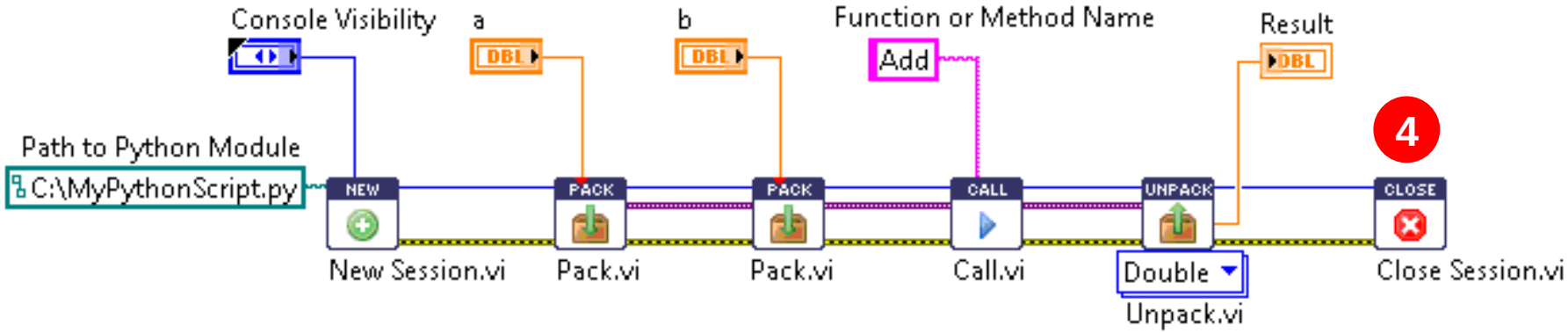
C:\MyPythonScript.py

```python
import math

def Add(a, b):
    return a+b;

def ConcatenateStrings(str1, str2):
    return str1 + str2;

def EuclideanDistance(point1, point2):
    xDiff = point1[0] - point2[0];
    yDiff = point1[1] - point2[1];
    return math.sqrt(xDiff*xDiff + yDiff
```



Python Integration Toolkit

- Close the session



Python Integration Toolkit – Getting Started

1. Install LabVIEW 2015 or later.

2. Install Python Integration Toolkit
   - Automatically installs Python and Canopy IDE

3. Locate your Python script and function
   - Install Python packages required by Python functions via Canopy IDE

4. Run LabVIEW VI that calls Python script/function using Python Integration Toolkit API

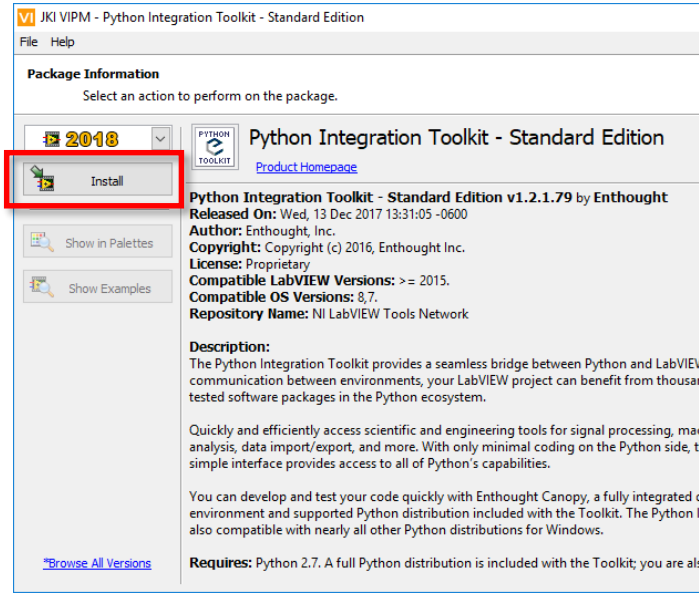
Python Integration Toolkit – Getting Started

1. Install LabVIEW 2015 or later.

2. Install Python Integration Toolkit
   - Automatically installs Python and Canopy IDE

3. Locate your Python script and function
   - Install Python packages required by Python functions via Canopy IDE

4. Run LabVIEW VI that calls Python script/function using Python Integration Toolkit API



NATIONAL INSTRUMENTS

Python Integration Toolkit – Getting Started

1. Install LabVIEW 2015 or later.

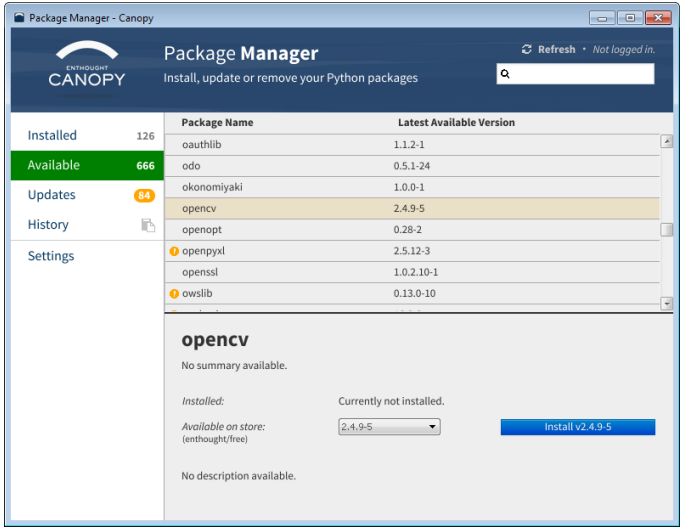
2. Install Python Integration Toolkit
   - Automatically installs Python and Canopy IDE

3. Locate your Python script and function
   - Install Python packages required by Python functions via Canopy IDE

4. Run LabVIEW VI that calls Python script/function using Python Integration Toolkit API

MyPythonScript.py

```
1  import sys
2  import cv2
3  import numpy as np
4  import os.path as op
5
```



NATIONAL INSTRUMENTS

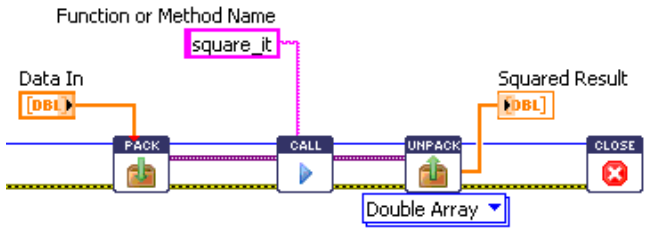
Python Integration Toolkit – Getting Started

1. Install LabVIEW 2015 or later.

2. Install Python Integration Toolkit
   - Automatically installs Python and Canopy IDE

3. Locate your Python script and function
   - Install Python packages required by Python functions via Canopy IDE

4. Run LabVIEW VI that calls Python script/function using Python Integration Toolkit API

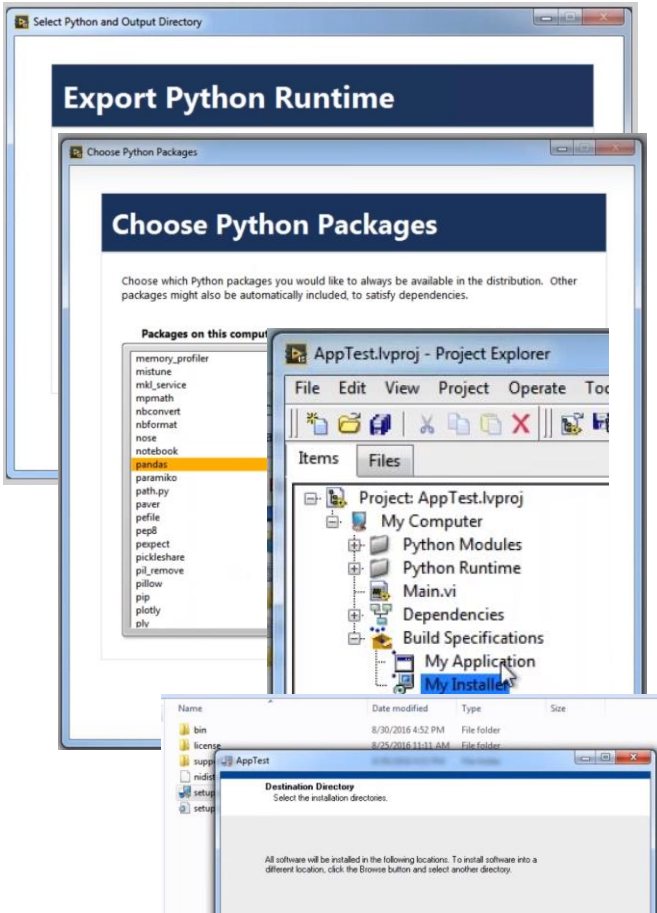


Python Integration Toolkit – Benefits

- Easy deployment of LabVIEW EXE and Python environment
- Includes Canopy Python IDE
- Native functions for the following:
  - Get/set Python global variables
  - Display Python console
  - Specify which Python installation to use
- Supports Python classes
- Variety of shipping examples

Easy deployment of LabVIEW EXE and Python environment

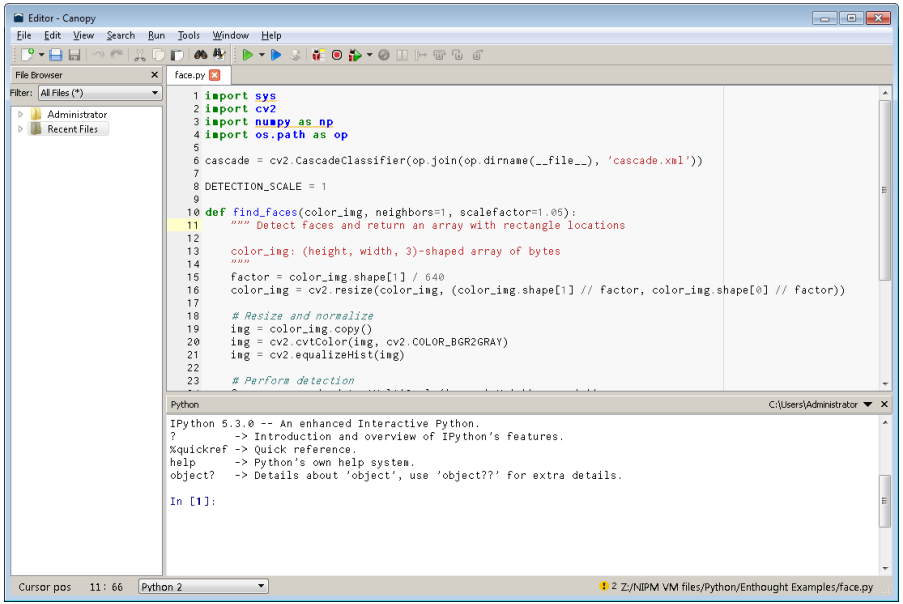
1. **[Toolkit]** Export Python Runtime
   - Tools»Python Integration Toolkit»Export Python Runtime
     - Choose Python packages to export
2. **[LabVIEW EXE Buildspec]** Create EXE
3. **[LabVIEW Installer Buildspec]** Create installer
   - Installs LabVIEW-built EXE
   - Installs Python Runtime to subfolder of installed application. Python Runtime is scoped just to the application.
   - Calls Python Runtime as post-install step

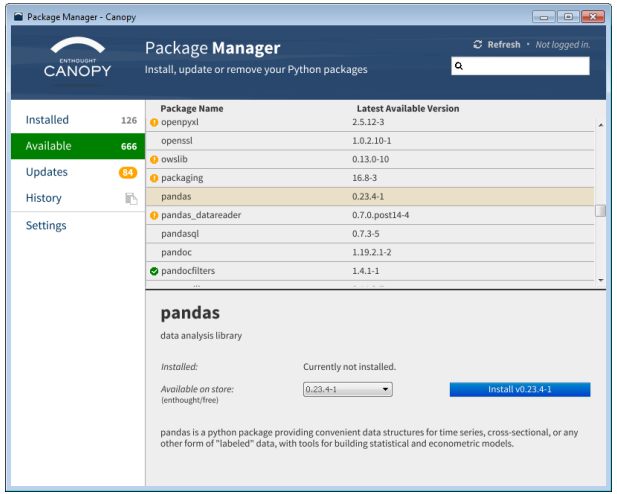


http://docs.enthought.com/python-for-LabVIEW/guide/deployment.html

Includes Canopy Python IDE

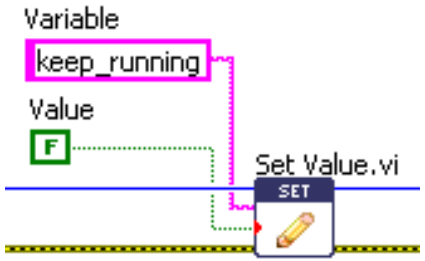
- Analysis and development environment for scientists and engineers using Python
- Provides curated set of packages managed through a package manager GUI
- Tailored to needs and workflows of scientists, analysts, and engineers

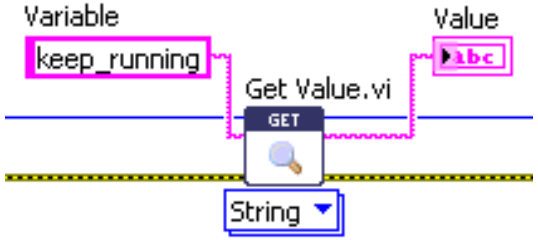




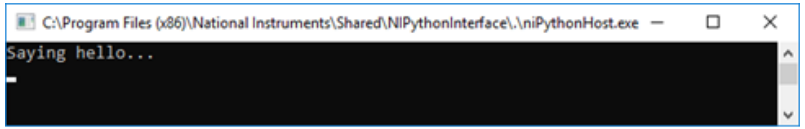
Additional Functions

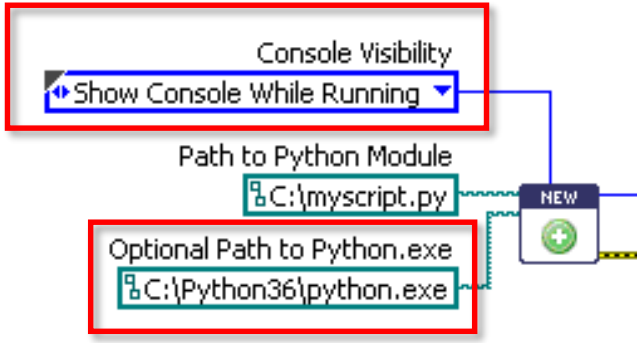
Get and set Python global variables using native functions





Display Python console for debugging



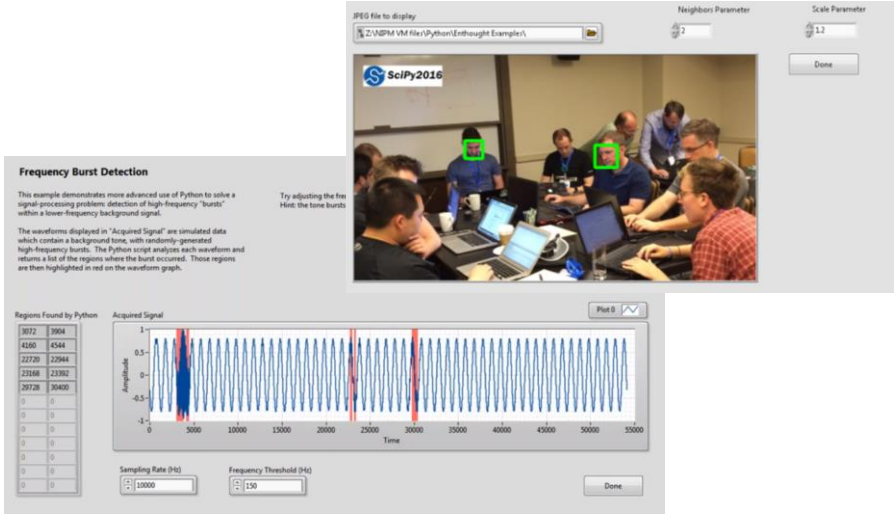


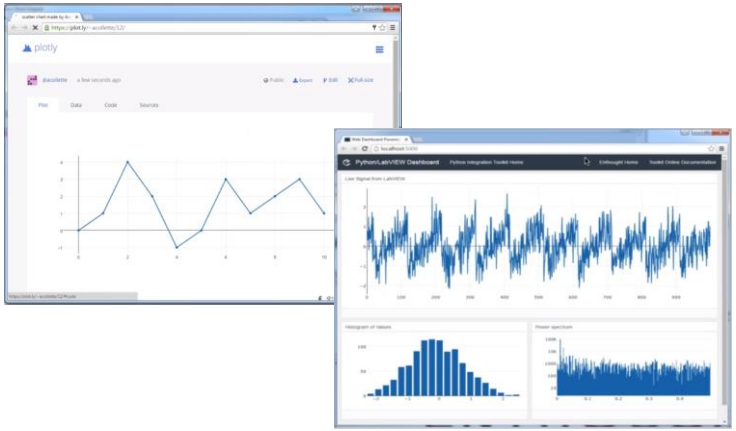
Specify path to Python.exe

NATIONAL INSTRUMENTS

Variety of Shipping Examples

- Face detection using OpenCV package
- Frequency burst detection
- Call a cloud service
- Publish data to the web with plot.ly
- Python-powered web dashboard
- Get and set Python global variables
- Capture Printed Output
- Communication between concurrent Python script and LabVIEW VI
- …





NATIONAL INSTRUMENTS

Enthought Python Integration Toolkit Demos

- Face detection using OpenCV package
- Frequency burst detection
- Call a cloud service
- Publish data to the web with plot.ly
- Python-powered web dashboard
- Get and set Python global variables
- Capture Printed Output
- …

Additional Use Cases

Concurrent communication between LabVIEW and long-running Python operation

How

- LabVIEW calls Python function that launches a long-running operation in a new thread… which can communicate data via global variable or Python queue.

- LabVIEW and the long-running Python function can communicate via Python queues and/or global variables

Demo

```python
def launch_complicated_operation():
    """ Called from LabVIEW to start the operation in a parallel thread """
    global worker_thread

    worker_thread = Thread(target=do_something_complicated)

    # This line is important; the "daemon" flag instructs Python not to wait
    # for the thread to finish before exiting.  We set it to True, so that
    # when we use Close Session from LabVIEW, Python will be sure to exit.
    worker_thread.daemon = True

    worker_thread.start()
def do_something_complicated():
    """ Simulates a long-running Python operation that periodically puts
    information into the queue.
    """

    status_queue.put("Starting up...")

    for idx in xrange(10):

        # See if LabVIEW has requested that we stop.
        if not keep_running:
            status_queue.put("Exiting early...")
            break

        # Do work.  In this case, we just sleep for 1 second.
        status_queue.put("Working... {}".format(idx+1))
        time.sleep(1)

    status_queue.put("Done.")
```

```python
def is_running():
    """ True if the worker thread is running, False otherwise """
    if worker_thread is not None and worker_thread.is_alive():
        return True
    return False

def check_queue():
    """ Get the latest item on the queue, or return an empty string """
    try:
        result = status_queue.get_nowait()
        if result is None:
            result = ""
    except Empty:
        result = ""
    return result
```

NATIONAL INSTRUMENTS

Calling LabVIEW from Python

- https://github.com/ni/python_labview_automation

Launch LabVIEW

Set VI control values

Run VI

Get VI indicator values

Get error message

Close LabVIEW

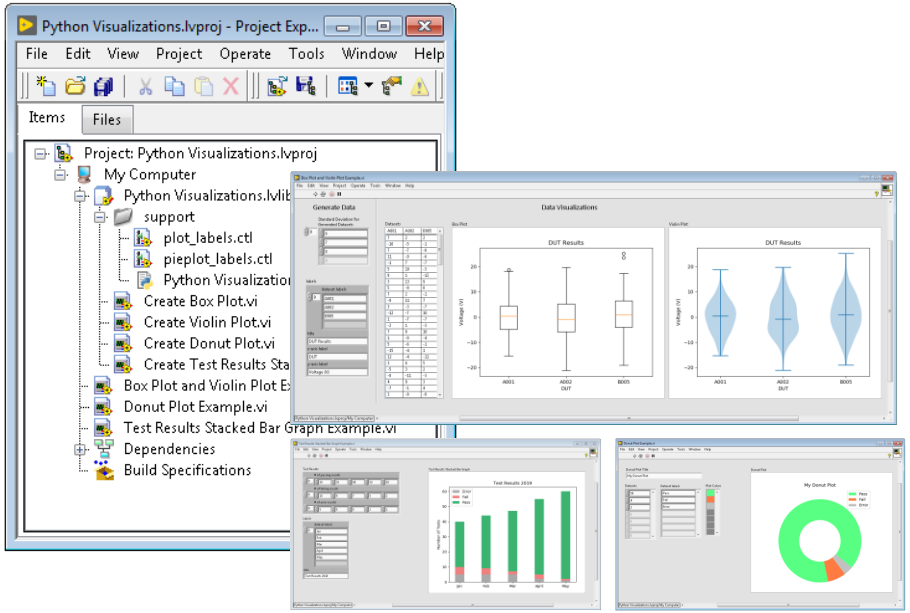
```
from labview_automation import LabVIEW
lv = LabVIEW()
lv.start() # Launches the active LabVIEW with the listener VI
with lv.client() as c:
    control_values = {
        "DBL Control": 5.0,
        "String Control": "Hello World!",
        "Error In": {
            "status": False,
            "code": 0,
            "source": ""
        }
    }
    indicators = c.run_vi_synchronous(
        vi_path, control_values)
    print(indicators['Result'])
    error_message = c.describe_error(indicators['Error Out'])
lv.kill() # Stop LabVIEW
```

Additional Resources

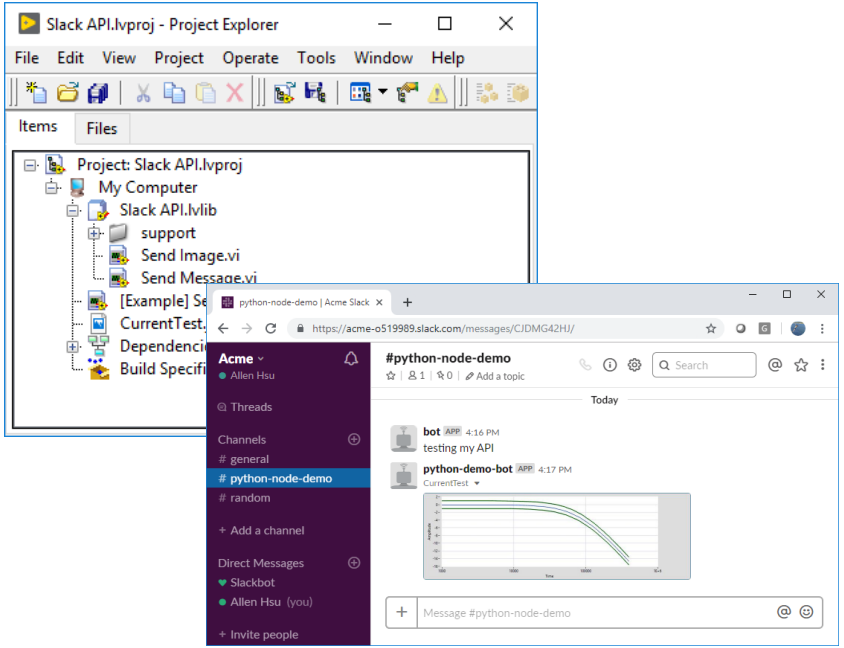
Download Python demo code

- https://github.com/allenh-ni/labview-python-demo

Python data visualizations



Slack API



NATIONAL INSTRUMENTS

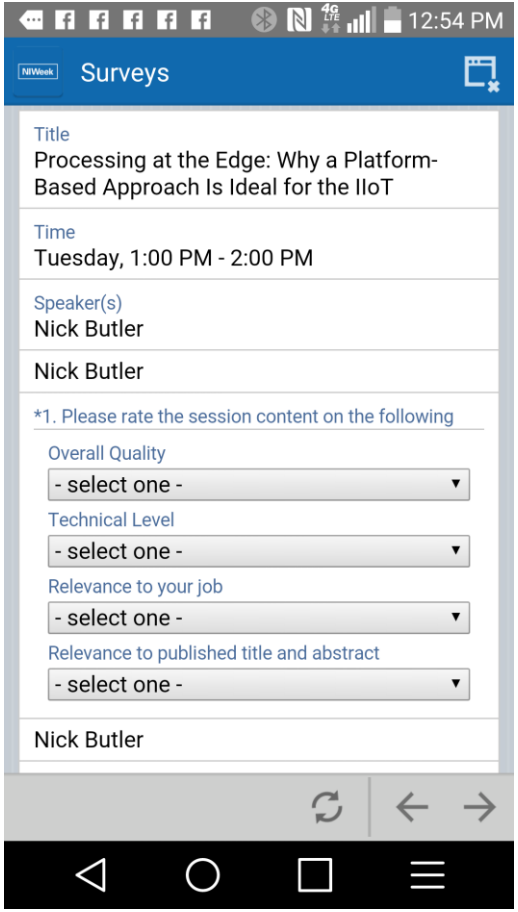
Additional Content

- Calling Python Class Methods Using LabVIEW Python Node
  https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019UFmSAM&l=en-US
- Using Optional Arguments with Python Node in LabVIEW
  https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019WoASAU&l=en-US

NIWeek Hands-On: Integrate LabVIEW with Python

- **When:** Wed, May 22nd, 10:30 – 11:30am
- **Where:** Meeting Room 18C
- **Speaker:** Danielle Jobe, VI Technologies

NATIONAL INSTRUMENTS™



Before you go, take the survey.

NATIONAL INSTRUMENTS

Stay Connected During and After NIWeek

ni.com/niweekcommunity

facebook.com/NationalInstruments

twitter.com/niglobal

youtube.com/nationalinstruments