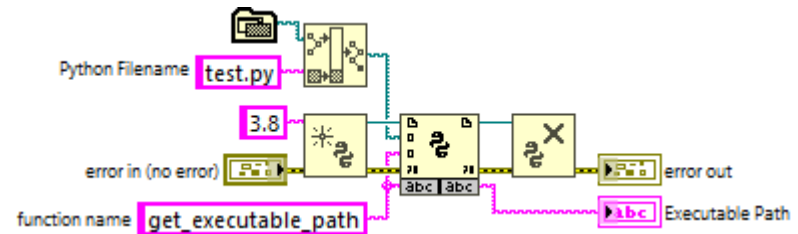


```
test.py x
test.py > ...
1 import sys
2
3 def get_executable_path() -> str:
4     return sys.executable
```



NI TestStand - Sequence Editor [Edit]

File Edit View Execute Debug Configure Source Control Tools Window Help

< > * [Icons] ?

Insertion Palette x python.seq x

Step Types ^

Python

Tests

- Pass/Fail Test
- Numeric Limit Test
- Multiple Numeric Limit Test
- String Value Test
- Action
- FTP Files
- Additional Results
- Sequence Call

Steps: MainSequence

STEP	DESCRIPTION	SETTINGS
+ Setup (0)		
- Main (1)		
+ Get Executable Path	Action, get_executable_path(...)	
<End Group>		
+ Cleanup (0)		

The Pythonic Requirements

FROM LABVIEW OR TESTSTAND

Jesper Kjær Sørensen

Systems Engineer | ✉: Jks@gpower.io

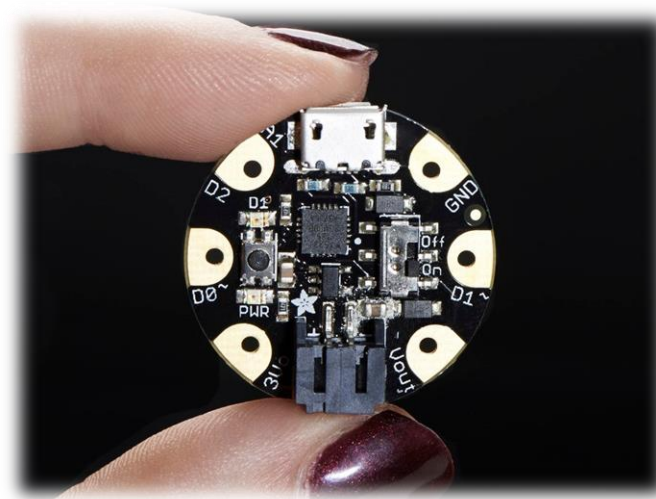
<https://www.linkedin.com/in/jesper-kjaer-soerensen/>



#OurGiantsAreFemale: Limor “LadyAda” Fried

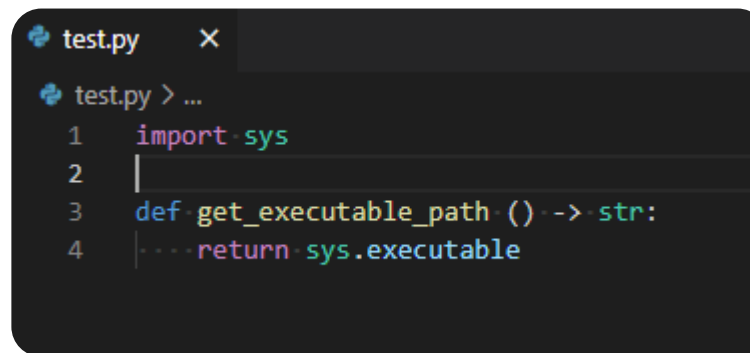


[Photo](#) by Max Morse / [CC BY](#)



Preconditions

- ❑ Separation of Concerns (SoC)
- ❑ Use Packaging Technology to orchestrate installations
- ❑ Windows Environment Variables and Command Line Applications
- ❑ Python Basics



```
test.py  ×
test.py > ...
1  import sys
2  |
3  def get_executable_path () -> str:
4  |     return sys.executable
```

What is the problem?

- ❑ Installing and configuring Python on Windows consistently
- ❑ Developing for Scalability, Distribution and Consistency
- ❑ Applying Python common practices, conventions and Expectations
- ❑ Connecting LabVIEW or TestStand to a specific Python environment
- ❑ The LabVIEW Python Node Examples does not work out of the box on a blank PC.

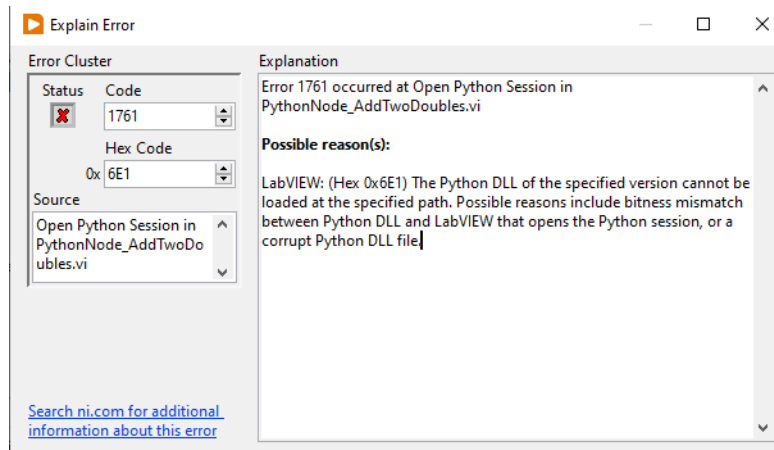
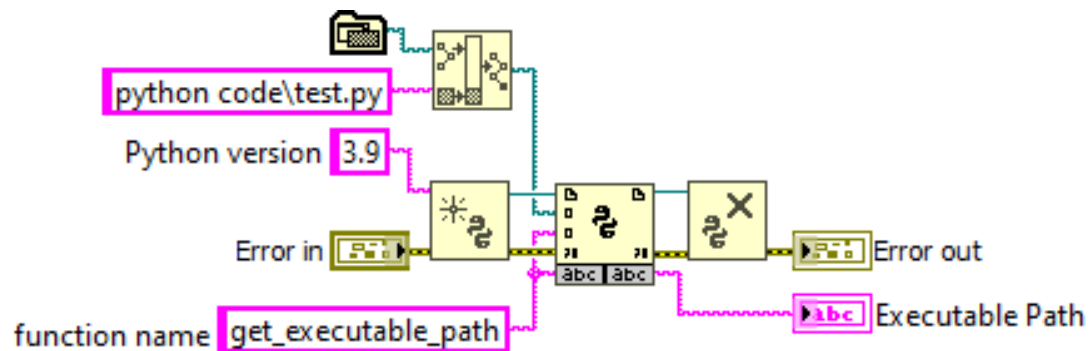


Photo by Rob Young / CC BY

Goals of this talk

- ❑ Install and manage Python through NI Package Manager consistently
- ❑ Expand Python with third party packages also in offline situations
- ❑ Have consistent Environments both in Development and Production
- ❑ Enable you to do one of these two scenarios and know the difference of implementation.



NI TestStand - Sequence Editor [Edit]

File Edit View Execute Debug Configure Source Control Tools Window Help

Insertion Palette

Step Types

Python

Tests

- Pass/Fail Test
- Numeric Limit Test
- Multiple Numeric Limit Test
- String Value Test

Action

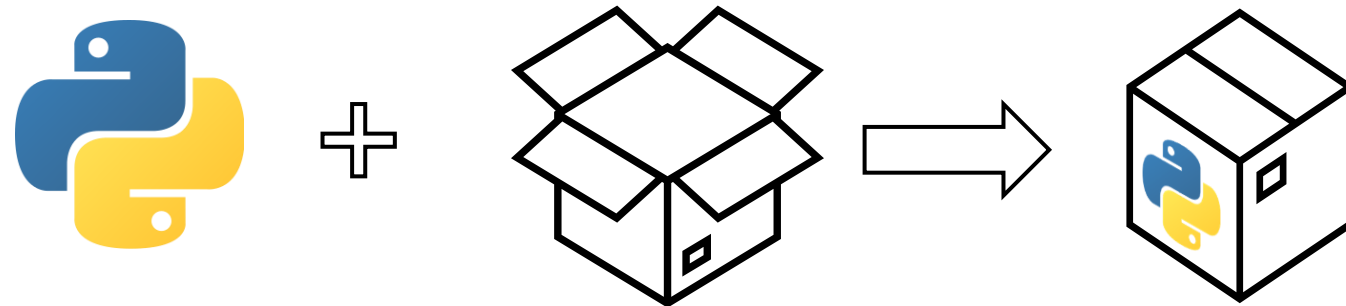
- Get Executable Path

Steps: MainSequence

STEP	DESCRIPTION	SETTINGS
+ Setup (0)		
- Main (1)		
Get Executable Path	Action, get_executable_path(...)	
<End Group>		
+ Cleanup (0)		

The Python Environment

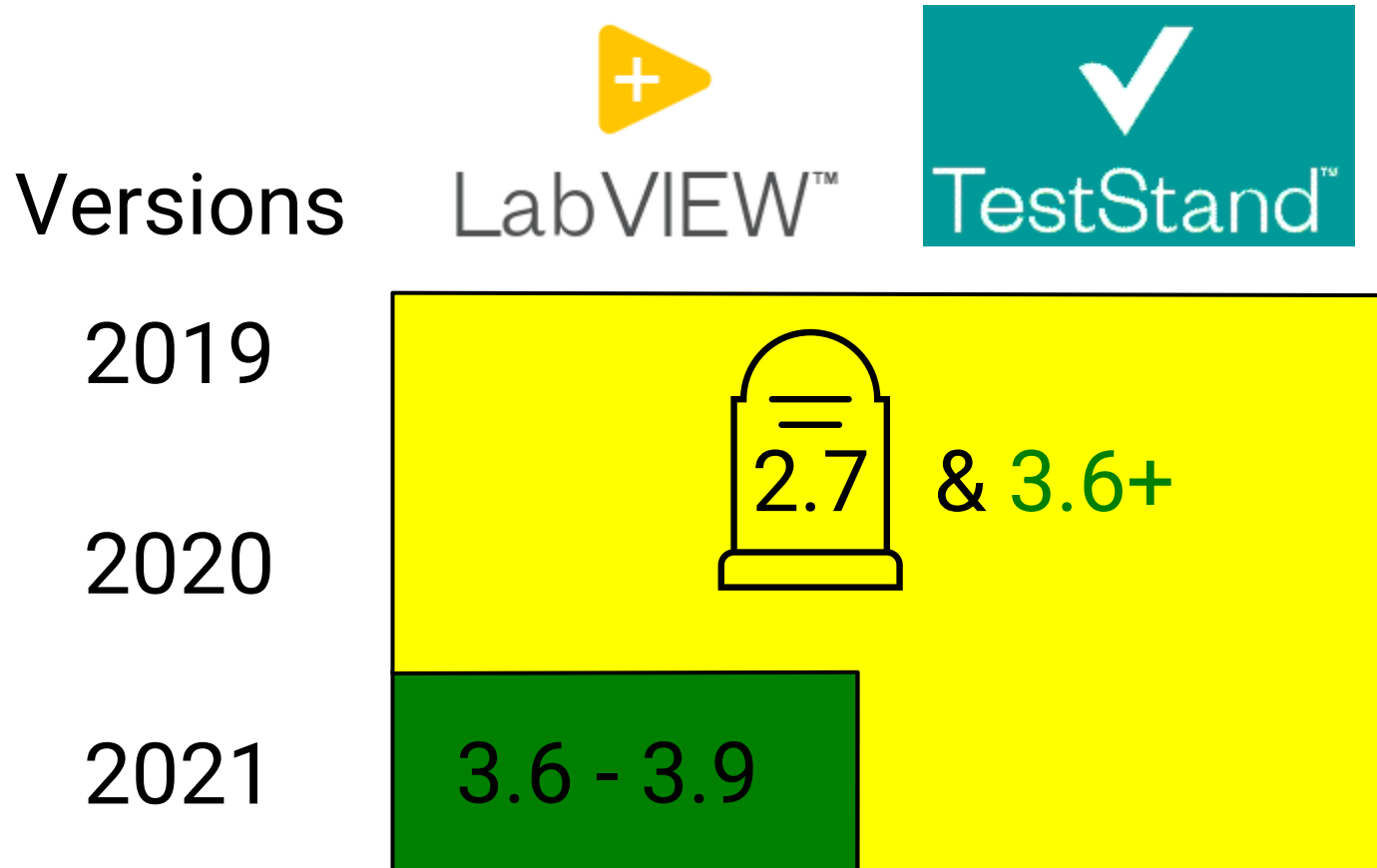
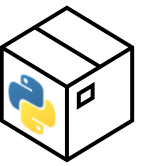




Creating a Python Installer Package

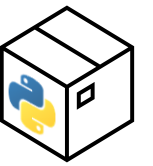
AND INSTALLING PYTHON ON WINDOWS

Python Compatibility Chart



IMPORTANT: Remember to match bitness. Use 32bit Python with 32Bit LabVIEW/Teststand

Installing the Interpreter consistently



```
CMD: C:\> python-x.y.z.exe /action [Optional configurations]
```

No interaction from user

```
C:\>python-3.9.8.exe /passive TargetDir="C:\Python39" PrependPath=1 CompileAll=1
```

No dialogue for user to see.

```
C:\>python-3.9.8.exe /quiet TargetDir="C:\Python39" PrependPath=1 CompileAll=1
```

Uninstallation:

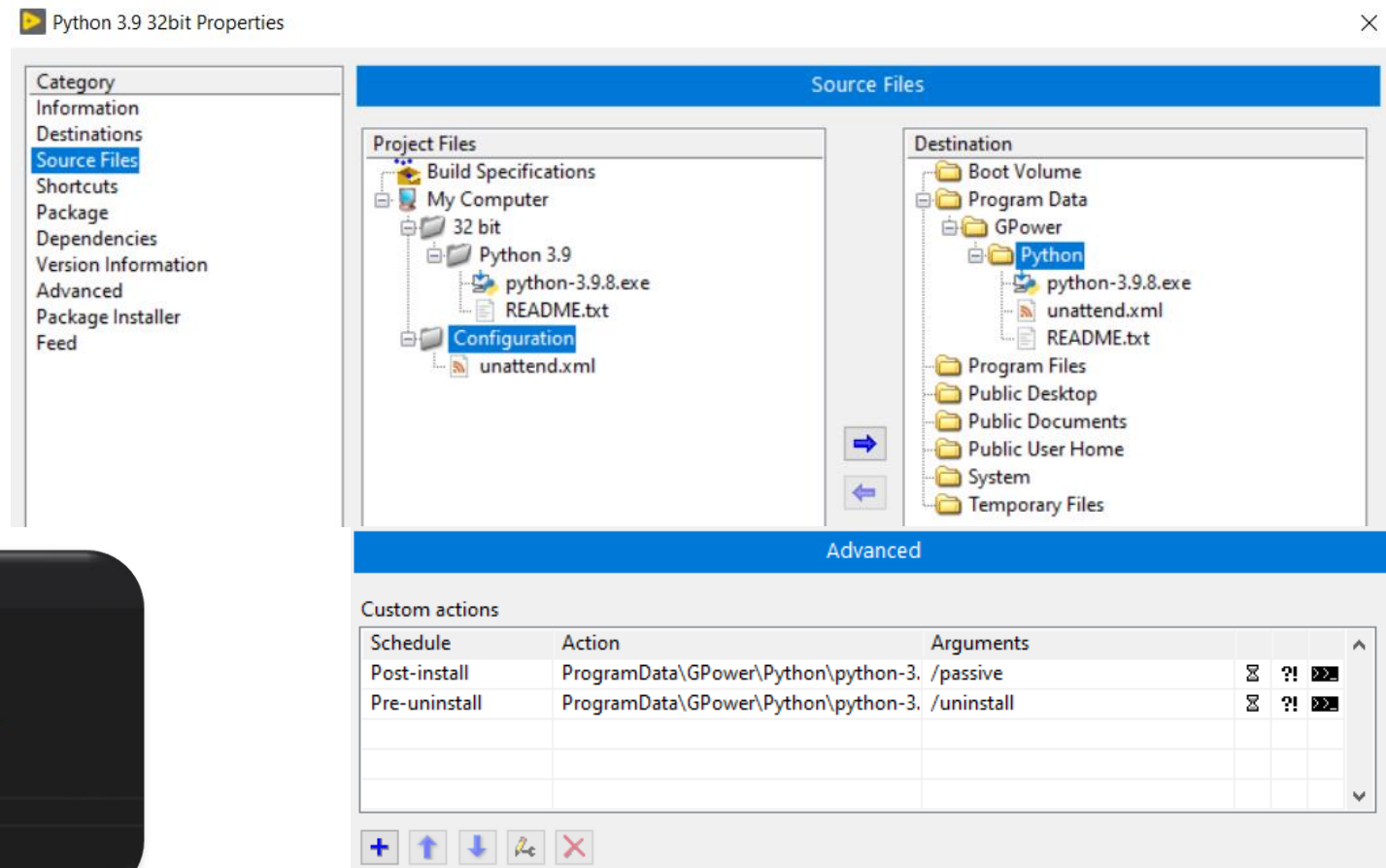
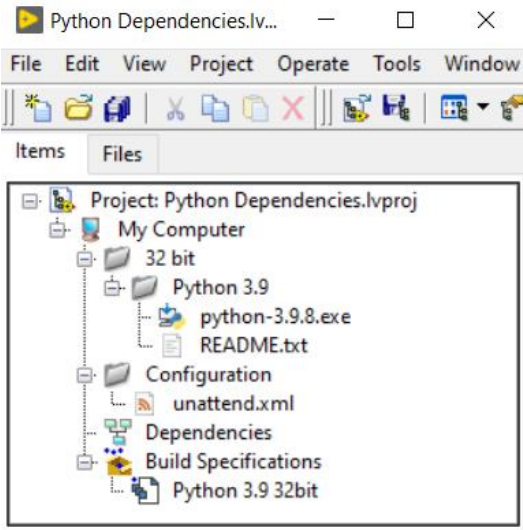
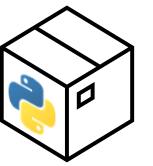
```
C:\>python-3.9.8.exe /uninstall
```

Solves the following:

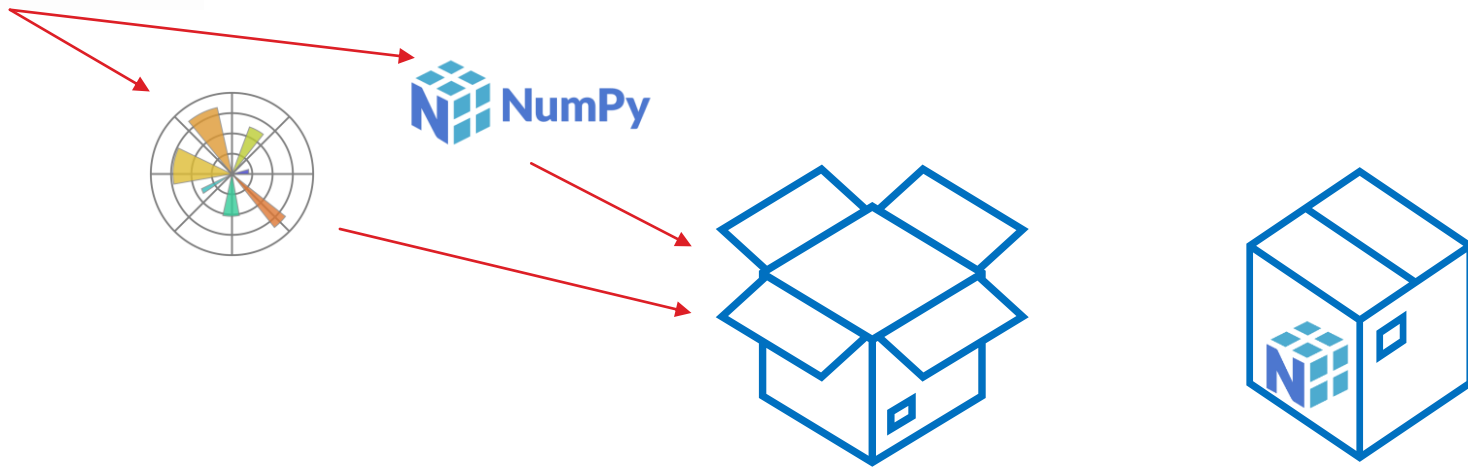
- Installs Python for Windows to C:\Python39
- Precompiles all Python interpreter files
- Adds Python to the System Path

For a complete list of install options: <https://docs.python.org/3/using/windows.html#installing-without-ui>

Wrapping Python installer in an NI Package



```
unattend.xml
unattend.xml
1 <Options>
2   <!--<Option Name="InstallAllUsers" Value="1"/>
3   <!--<Option Name="TargetDir">c:\Python39</Option>
4   <!--<Option Name="CompileAll" Value="1"/>
5   <!--<Option Name="PrependPath" Value="1"/>
6 </Options>
```



Third Party Python Packages

THE ART OF USING PIP

The Requirements.txt File



```
requirements.txt
requirements.txt
1 ##### Requirements without Version Specifiers #####
2 pytest
3 pytest-cov
4 beautifulsoup4
5
6 ##### Requirements with Version Specifiers #####
7 # ... See https://www.python.org/dev/peps/pep-0440/#version-specifiers
8 docopt == 0.6.1 ..... # Version Matching. Must be version 0.6.1
9 keyring >= 4.1.1 ..... # Minimum version 4.1.1
10 coverage != 3.5 ..... # Version Exclusion. Anything except version 3.5
11 Mopidy-Dirble ~ = 1.1 ..... # Compatible release. Same as >= 1.1, != 1.*
12
13 ##### Refer to other requirements files #####
14 -r other-requirements.txt
15
16 ##### A particular file #####
17 ./downloads/numpy-1.9.2-cp34-none-win32.whl
18 http://wxpython.org/Phoenix/snapshot-builds/wxPython\_Phoenix-3.0.3.dev1820+49a8884-cp34-none-win\_amd64.whl
19
20 ##### Additional Requirements without Version Specifiers #####
21 # ... Same as 1st section, just here to show that you can put things in any order.
22 rejected
23 green
```

<https://pip.pypa.io/en/stable/reference/requirements-file-format/>

Managing Packages for Python



Running modules as programs in Python

```
CMD: C:\> python -m module [module options]
```

Install a single package using the Internet

```
C:\>python -m pip install numpy
```

Install a requirements file from the internet

```
C:\>python -m pip install -r requirements.txt
```

Download packages specified by a requirements file to a local repository

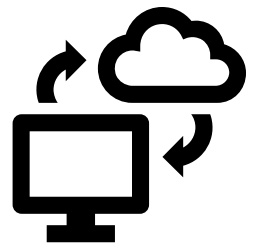
```
C:\>python -m pip download -r requirements.txt -d local_folder_path
```

Offline support

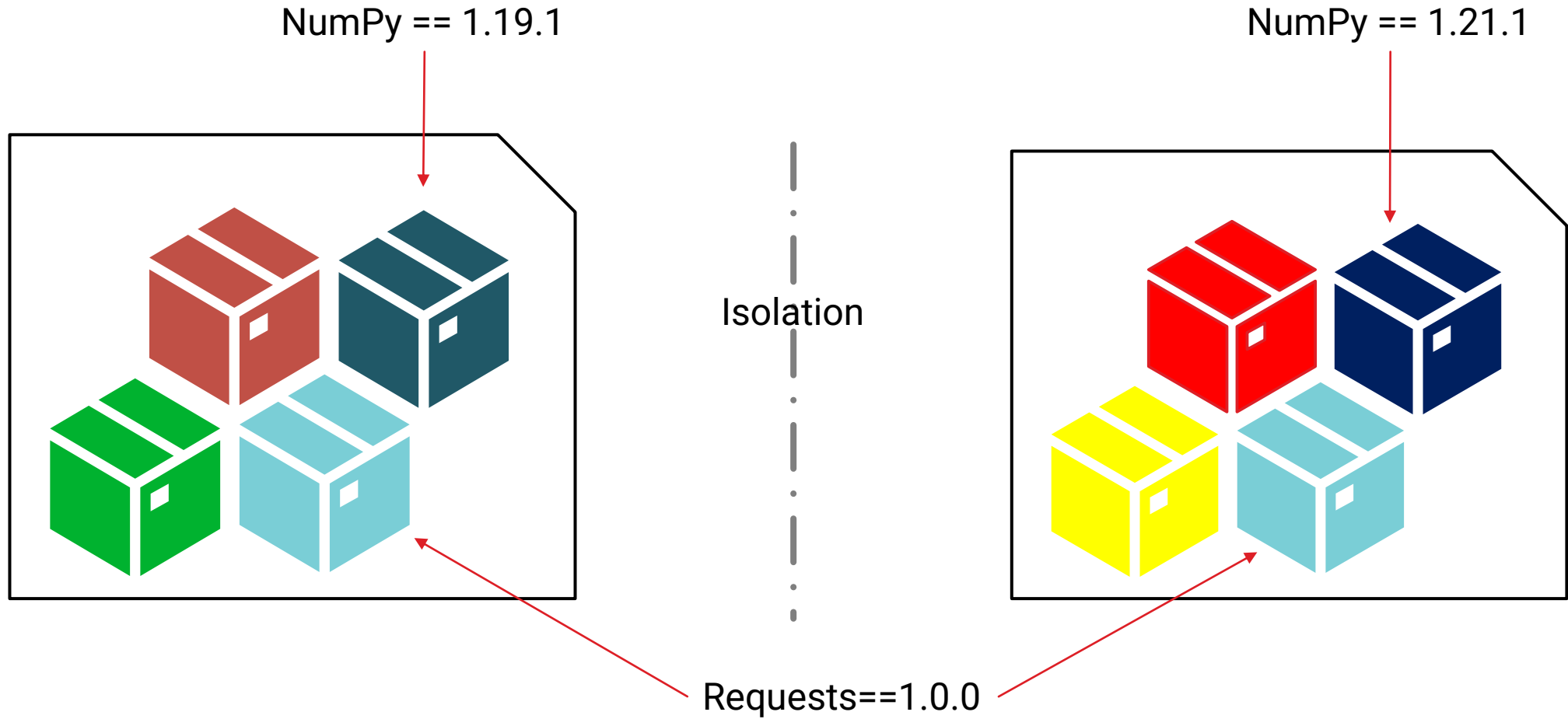
Install a requirements file from a local folder

```
C:\>python -m pip install -r requirements.txt --no-index --find-links local_folder_path
```

```
requirements.txt X
C: > git > piptest > requirements.txt
1 # install Virtualenv and Numpy
2 virtualenv; sys_platform == 'win32'
3 numpy; sys_platform == 'win32'
```



Virtual Environments for TestStand



Virtual Environments



Install Virtualenv to create and manage your virtual environments once.

```
C:\>python -m pip install virtualenv
```

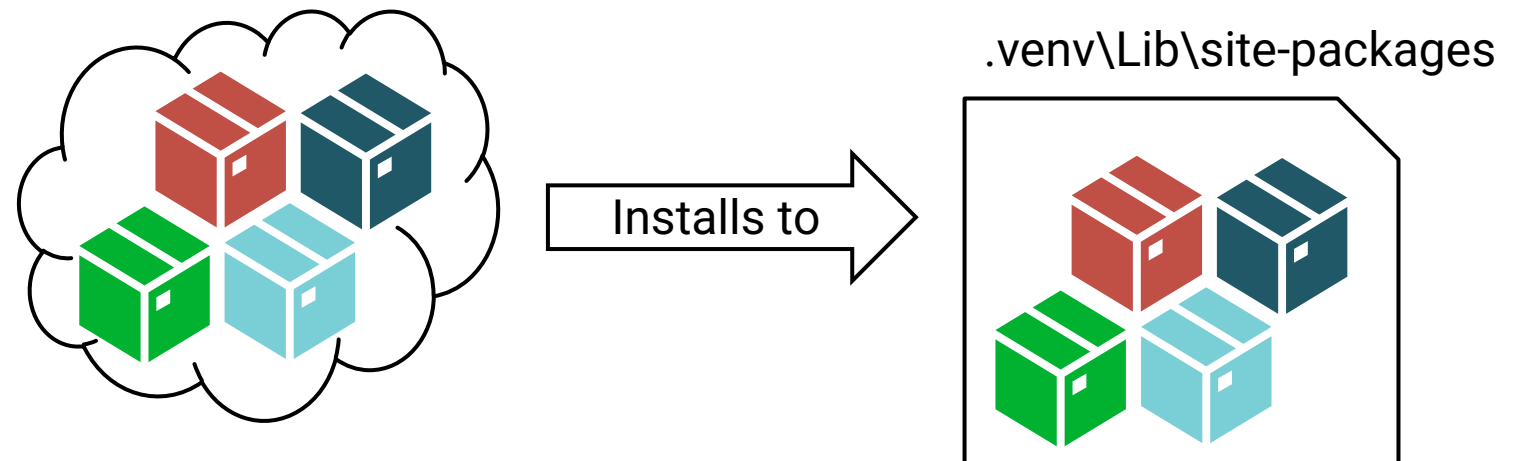
Create the virtual environment for TestStand in your folder of choice and activate it

```
C:\FolderOfChoice>python -m virtualenv .venv
```

```
C:\FolderOfChoice >.venv\Scripts\activate
```

Install the packages in the active environment isolated from the global environment

```
(.venv) C:\folder_of_choice > python -m pip install -r requirements.txt --no-index --find-links LocalRepoFolder
```



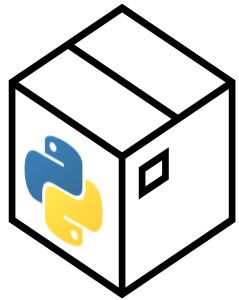
Recipe: Creating the NI Package Python Package distribution



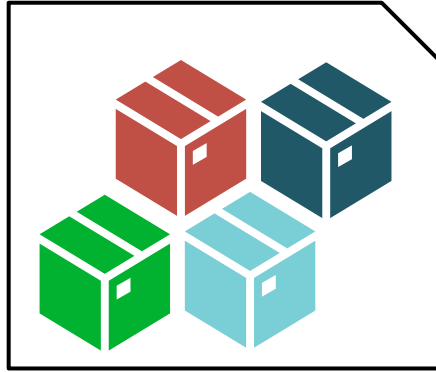
- Install the version of Python you need to use.
- Download required Packages using Pip.

```
python -m pip download -r requirements.txt -d local_folder_path
```

- Create an NI Package
- *(Optional CI) Initiate pip download when Requirements.txt changes*
- Include all the .whl files from the **local_folder_path** in the NI package
- Choose a Package Destination ie. C:\ProgramData\LocalPythonRepo\
C:\ProgramData\LocalPythonRepo\
- Set a version of the Package, use Semantic versioning www.semver.org
Major.Minor.Patch.Build
- Recreate your virtual environment from the distributed and versioned packages. (DevPC)
- Distribute it to the target PC (Production)
- Install on target PC using NI Package Manager



.venv\Lib\site-packages



LabVIEW™

Launching the Environment

FROM LABVIEW AND TESTSTAND

TestStand Calling the Python Environment

P Python Adapter Configuration [X]

Python Interpreter Options

Python Interpreter to use: Global [v]

Python Version: 3.9 [v]

Python Virtual Environment: [v] [Folder Icon]

... [Folder Icon]

Display Console for Interpreter Sessions

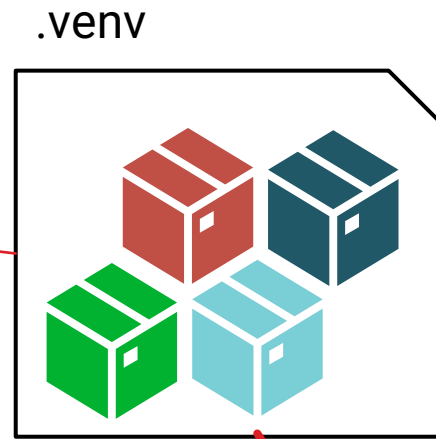
Python Module Viewer

Application Path: notepad.exe [v] [Folder Icon]

C:\WINDOWS\system32\notepad.exe

Arguments: %ModulePath%

Help OK Cancel



Step Settings for Action

P Module Properties

Module: [v] [Folder Icon] [File Icon] [Key Icon] [Refresh Icon]

(No file specified)

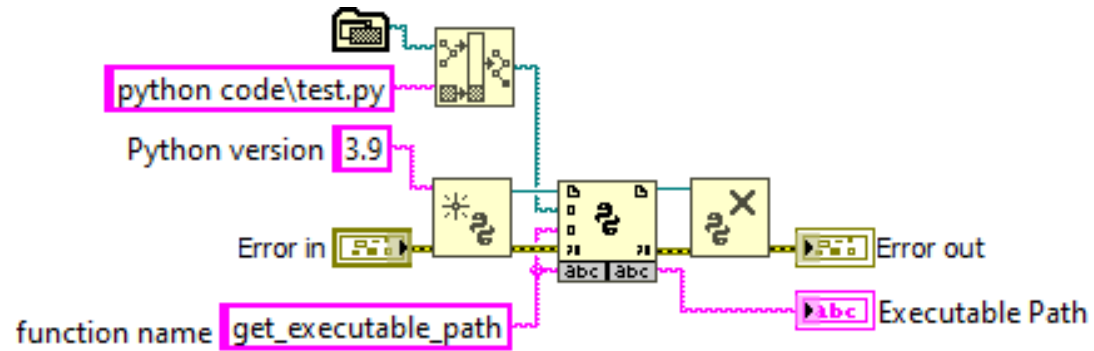
Operation Scope: Module [v] Operation Type: Call Method [v]

Class Name: [v] Function Name: [v]

Class Instance: [v] *fx* ✓

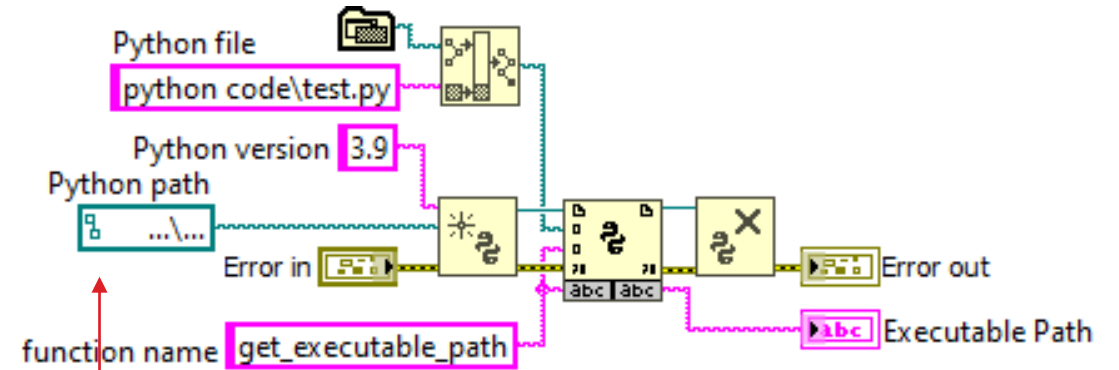
NAME	TYPE	LOG	VALUE	
Return Value	Dynamic	[v] [checkbox]	<i>fx</i> ✓	[+] [-]

LabVIEW Calling the Python Environment



LabVIEW™
2019 LabVIEW™
2020

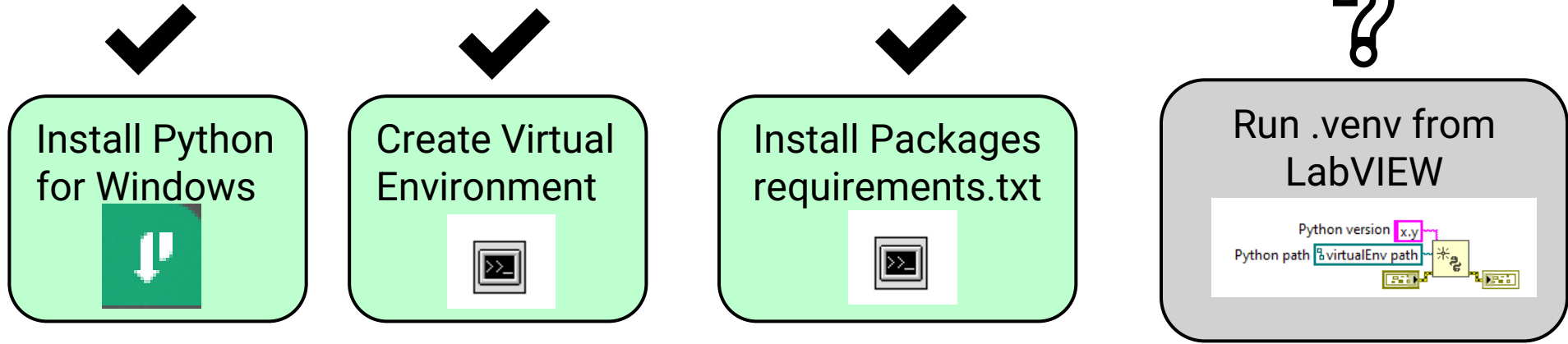
Uses the Global Python
So no virtual environments



LabVIEW™
2021

Path to the pythonXY.dll

My expected workflow



Ways around this, but there may be consequences to this.

Download the Python **embeddable** zip package
(Has no Pip support out of the box)

- Unpack this to .venv in the projects folder
- Modify **pythonxy._pth** by uncommenting **import site**
- Download <https://bootstrap.pypa.io/get-pip.py>
- Open a terminal in .venv\
• Run: `c:\...\venv\python get-pip.py`
- Prepend your pip commands with `\Scripts\pip`
- Install Packages.

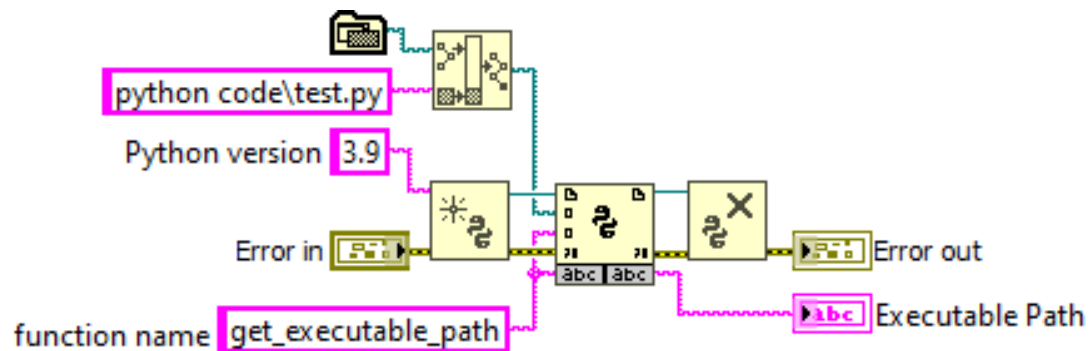
```
*python39._pth - Notepad
File Edit Format View Help
python39.zip
.

# Uncomment to run site.main() automatically
# import site

get-pip.py
libcrypto-1_1.dll
libffi-7.dll
libssl-1_1.dll
LICENSE.txt
pyexpat.pyd
python.cat
python.exe
python3.dll
python39._pth
```

Goals of this talk

- ✓ Install and manage Python through NI Package Manager consistently
- ✓ Expand Python with third party packages also in offline situations
- ✓ Have consistent Environments both in Development and Production
- ✓ Enable you to do one of these two scenarios and know the difference of implementation.



NI TestStand - Sequence Editor [Edit]

File Edit View Execute Debug Configure Source Control Tools Window Help

Insertion Palette

Step Types

Python

Tests

- Pass/Fail Test
- Numeric Limit Test
- Multiple Numeric Limit Test
- String Value Test

Action

- Get Executable Path

Steps: MainSequence

STEP	DESCRIPTION	SETTINGS
+ Setup (0)		
- Main (1)		
Get Executable Path	Action, get_executable_path(...)	
<End Group>		
+ Cleanup (0)		

Questions?