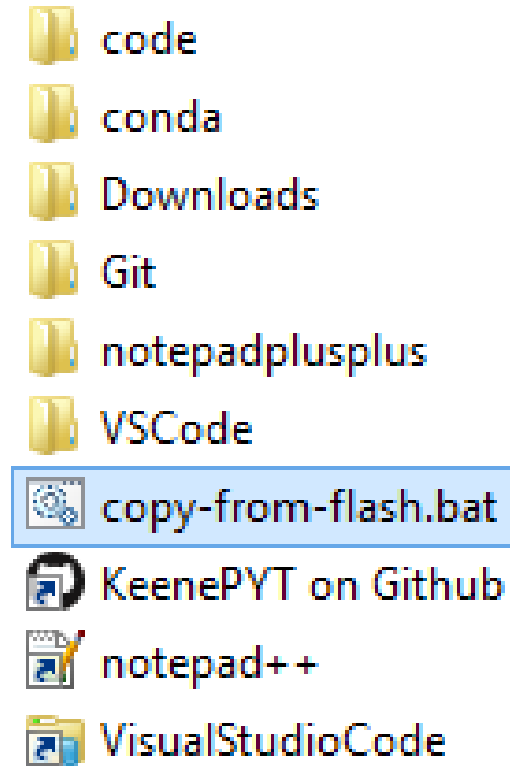


# Help us get started.

1. Get a flash drive.
2. Run **copy-from-flash.bat**.

This will create **C:\Projects\NEARC**.

The same code is available at  
<https://github.com/jswise/keenepyt>.



# Distribute Your Python Tools to ArcGIS Pro Users

Jason Wise, Earth Data Scientist, Terracon

# Distribute Your Python Tools to ArcGIS Pro Users

Jason Wise, Earth Data Scientist

**Terracon**

Northeast Arc User Group (NEARC)  
Spring 2019

# The Python Package Manager

It manages “environments” and “packages.” We’ll talk about what those are.

The screenshot shows the Python Package Manager window in ArcGIS Pro. The left sidebar contains a menu with options: New, Open, Save, Save As, Portals, Licensing, Options, Python (selected), Add-In Manager, Help, About, and Exit. The main area is titled 'Python Package Manager' and shows the 'Project Environment' as 'dev [C:\Users\jswise\AppData\Local\ESRI\conda\envs\dev]'. Below this are buttons for 'Managed Environments', 'Installed Packages', 'Update Packages', and 'Add Packages'. The 'Installed Packages' section lists 121 installed packages. A table shows the first few packages: arcgis (1.5.1), asn1crypto (0.24.0), astroid (2.2.5), atomicwrites (1.2.1), attrs (18.2.0), backcall (0.1.0), blas (1.0), bleach (2.1.4), ca-certificates (2018.03.07), and certifi (2018.8.24). The 'arcgis' package is highlighted, and a detailed view on the right shows its version (1.5.1), description (ArcGIS API for Python), homepage, license (Esri Master License Agreement (MLA)), and description (Script and automate ArcGIS Online and ArcGIS Enterprise).

ArcGIS Pro - KeenePYT - Map

## Python Package Manager

Project Environment

**dev** [C:\Users\jswise\AppData\Local\ESRI\conda\envs\dev]

Manage Environments

Installed Packages

Update Packages

Add Packages

### Installed Packages

The following list of Python packages are installed with ArcGIS Pro.

[Learn more about Conda packages](#)

Installed: 121

Name	Version
arcgis	1.5.1
asn1crypto	0.24.0
astroid	2.2.5
atomicwrites	1.2.1
attrs	18.2.0
backcall	0.1.0
blas	1.0
bleach	2.1.4
ca-certificates	2018.03.07
certifi	2018.8.24

**arcgis** Uninstall

Version: 1.5.1  
ArcGIS API for Python







[Homepage](#) License: Esri Master License Agreement (MLA)

**Description**  
Script and automate ArcGIS Online and ArcGIS Enterprise.

# Create dev & test environments.

In **C:\Projects\NEARC\code\keenepyt\batch**, run **create-dev-envs.bat**.

This will clone your default Python environment to two new environments, *dev* and *localtest*. It takes a while to run.

Active	Environments	Clone	Remove
<input type="radio"/>	arcgispro-py3 C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3		
<input checked="" type="radio"/>	dev C:\Users\jswise\AppData\Local\ESRI\conda\envs\dev		
<input type="radio"/>	localtest C:\Users\jswise\AppData\Local\ESRI\conda\envs\localtest		

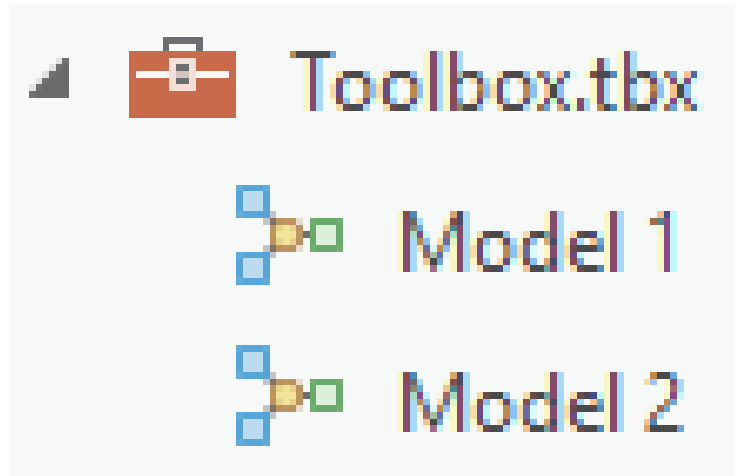
# The goals:

1. Give a geoprocessing toolbox to other users.
2. Easily update the toolbox.

With traditional methods (e.g. passing out .TBX files and Python scripts), updating is the hard part.

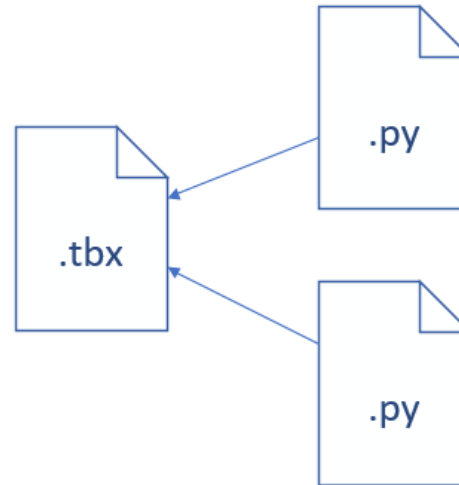
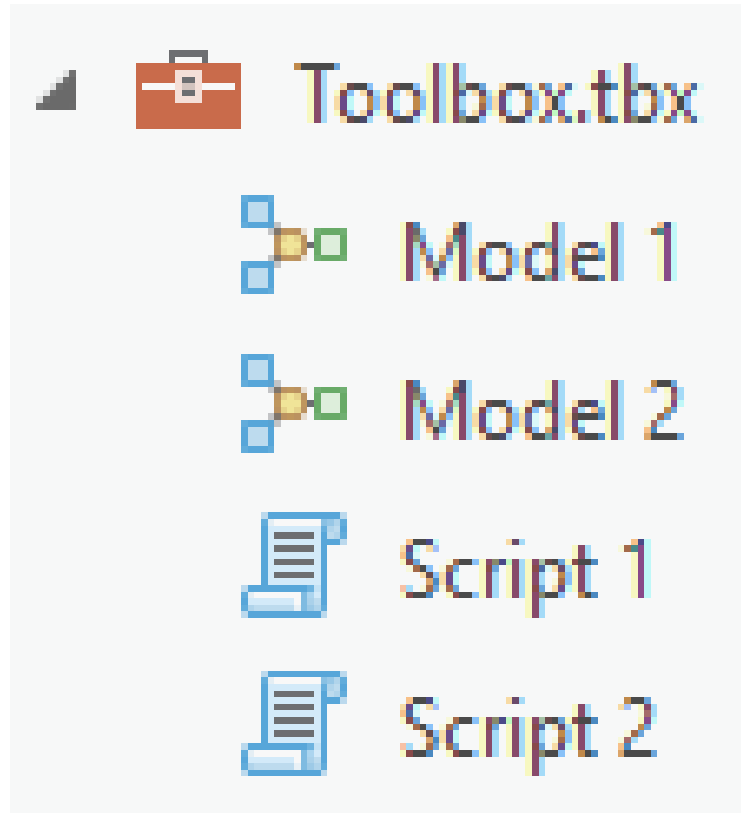
# The Toolbox Menagerie

A traditional toolbox is a binary file.



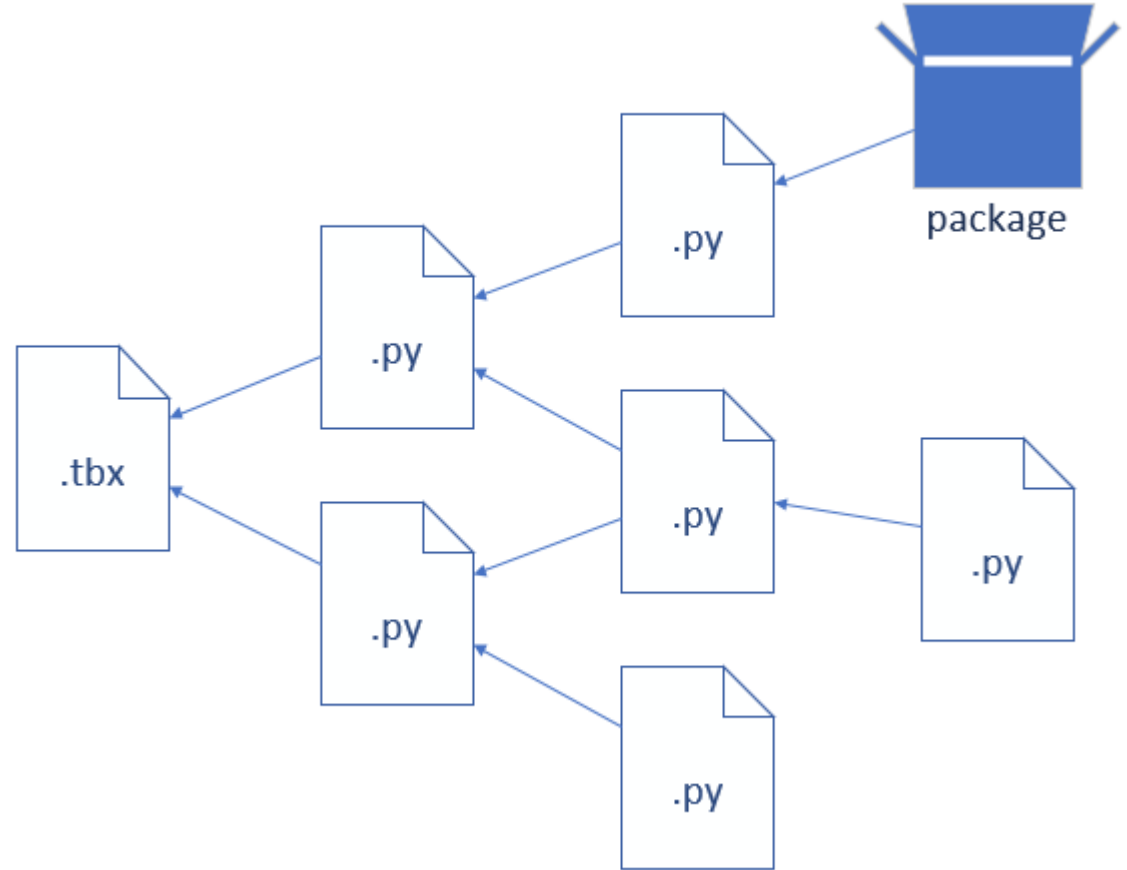
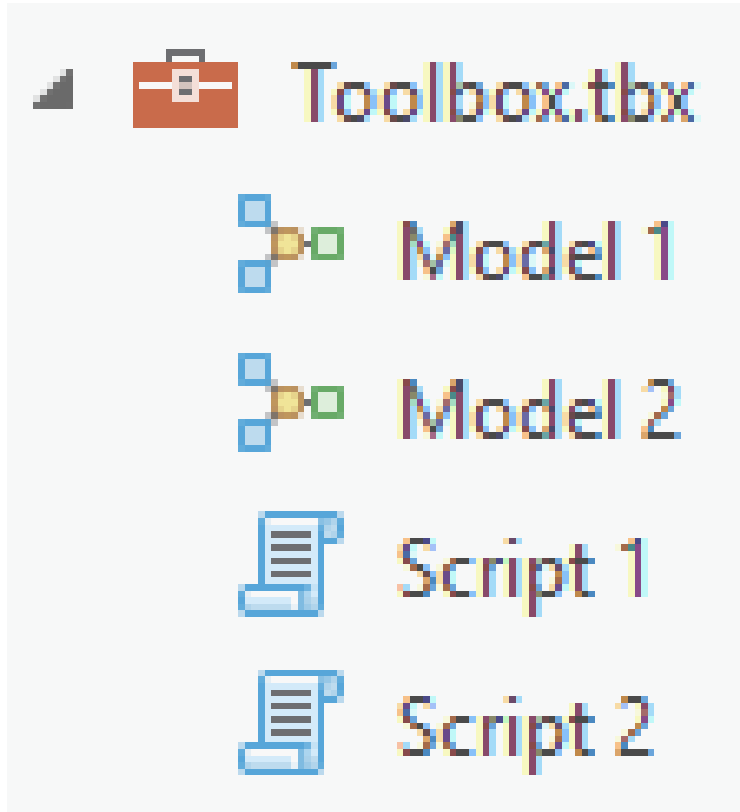
# The Toolbox Menagerie

Each script is a Python module (file).



# The Toolbox Menagerie

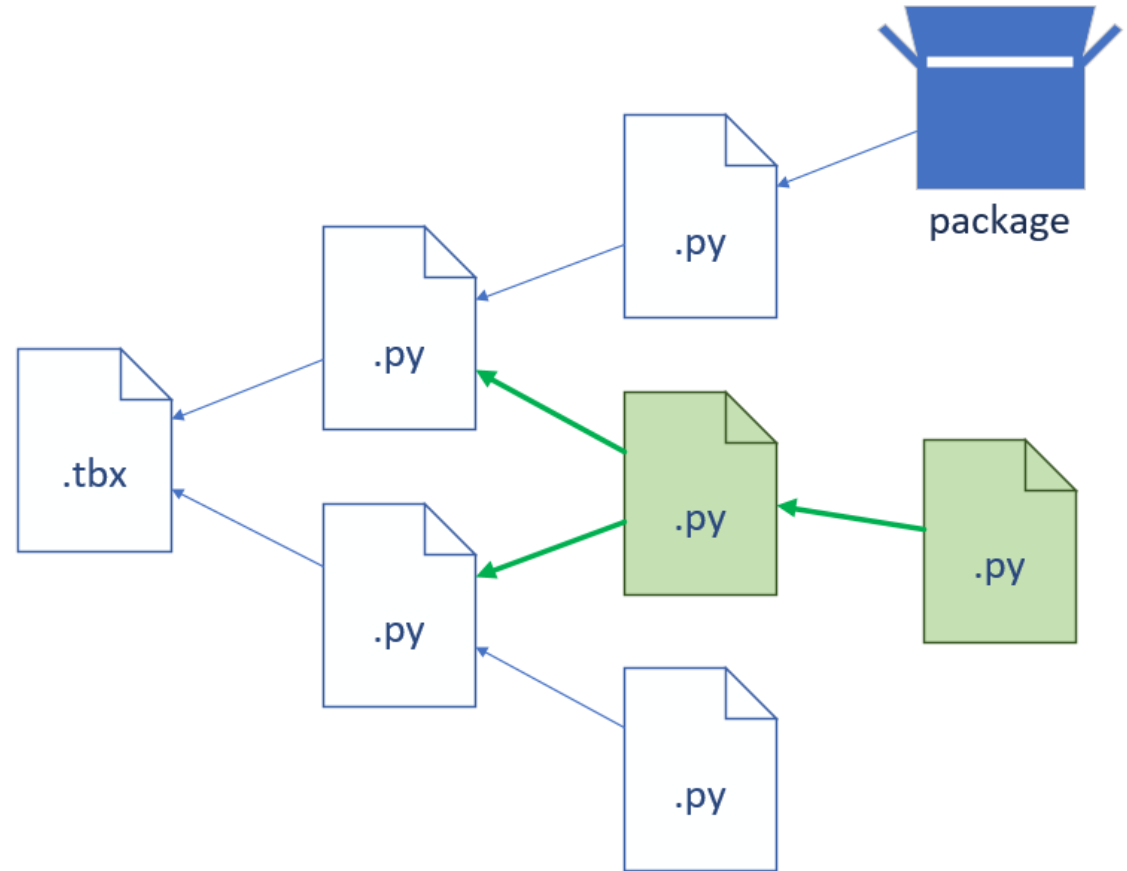
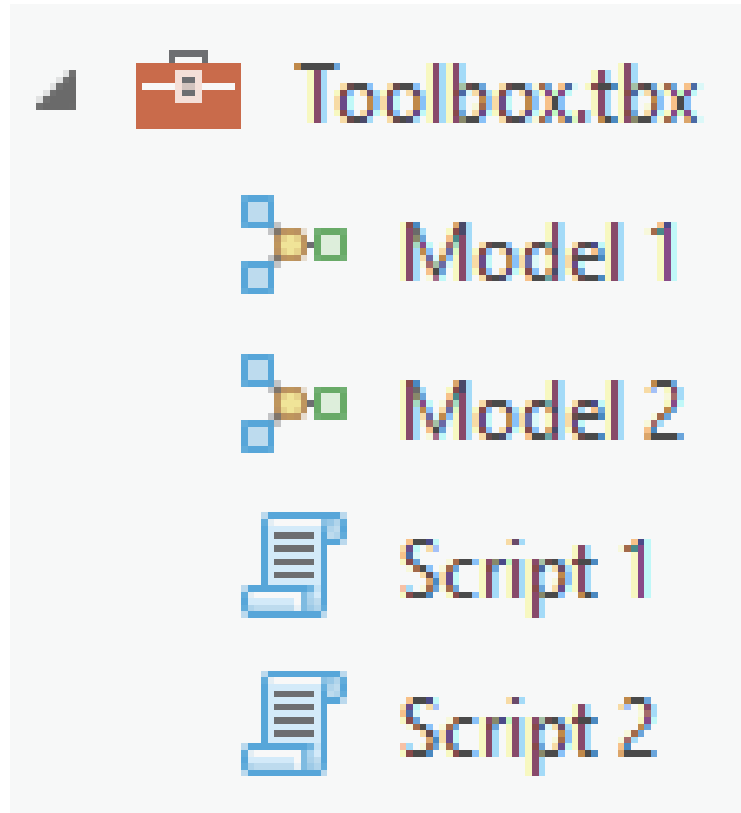
## Each Python module can import other modules & packages.





# The Toolbox Menagerie

Reusing code is good!



# The Toolbox Menagerie

A Python toolbox is a Python file with a goofy extension.  
It can be self-contained...

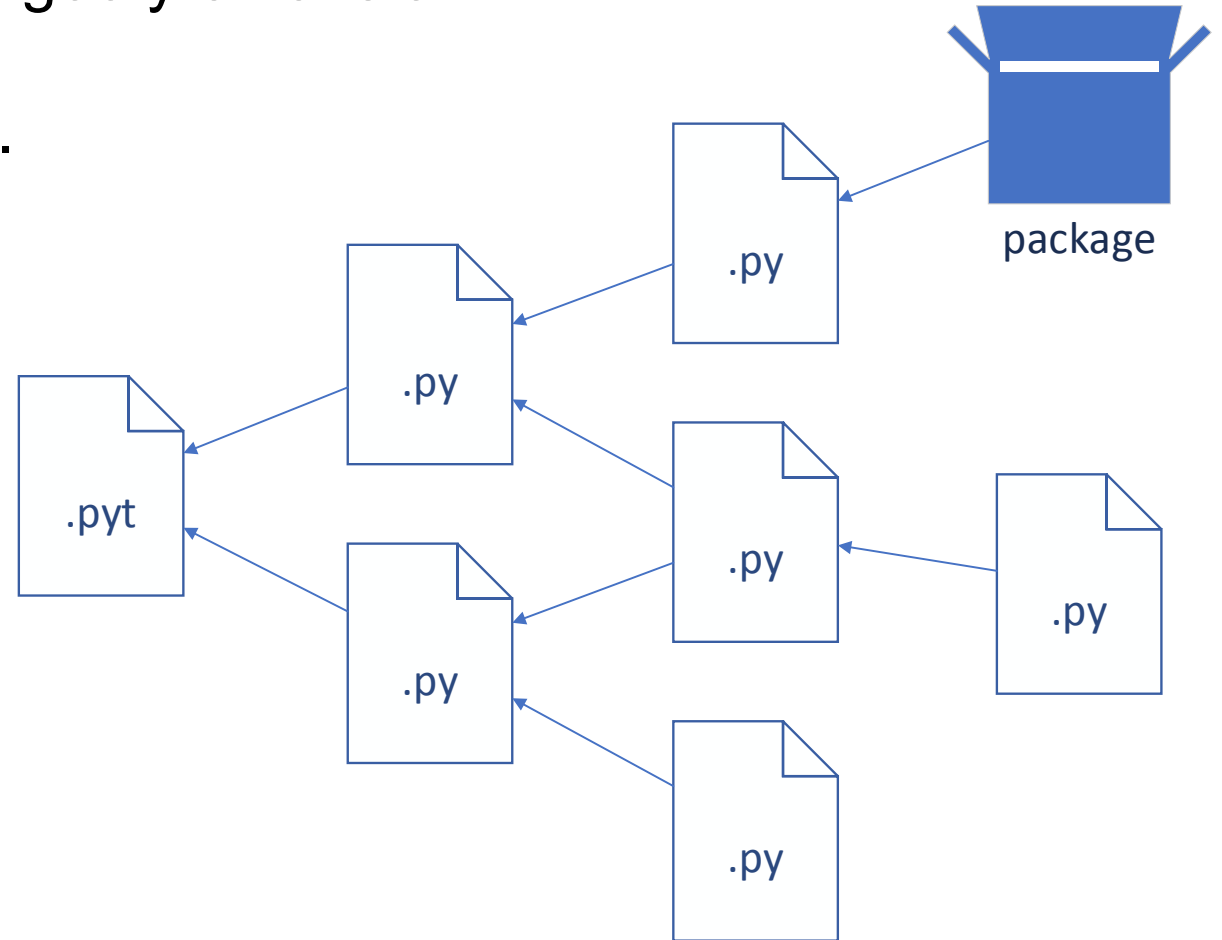


# The Toolbox Menagerie

A Python toolbox is a Python file with a goofy extension.

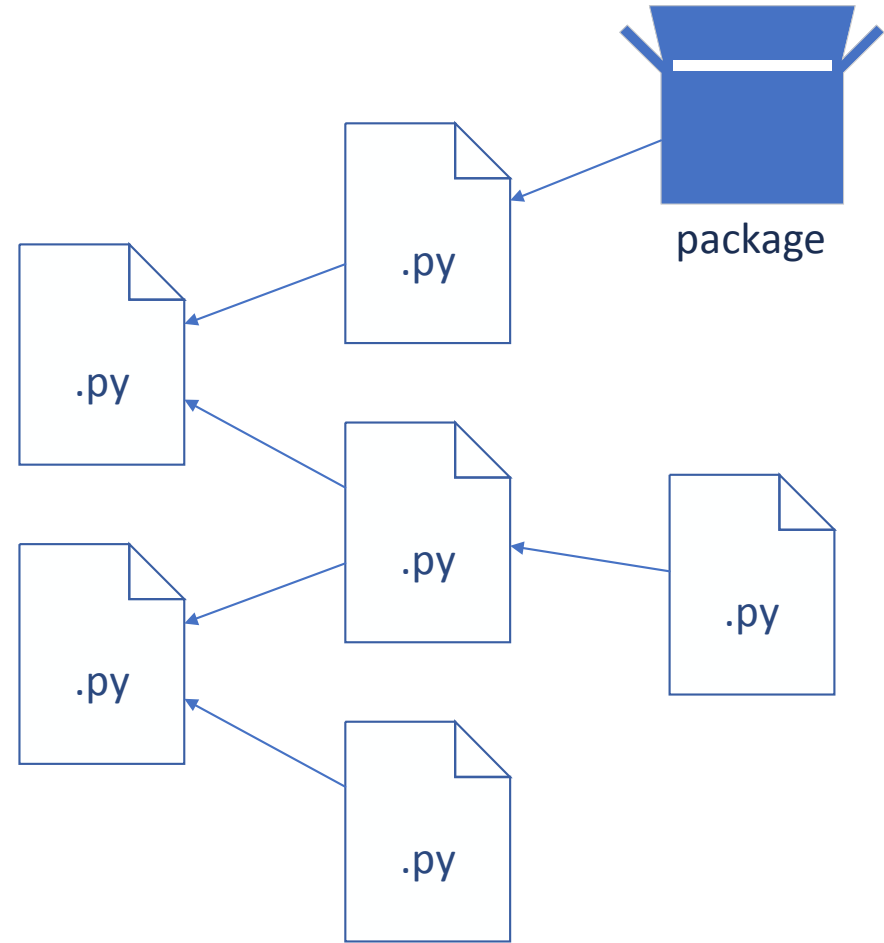
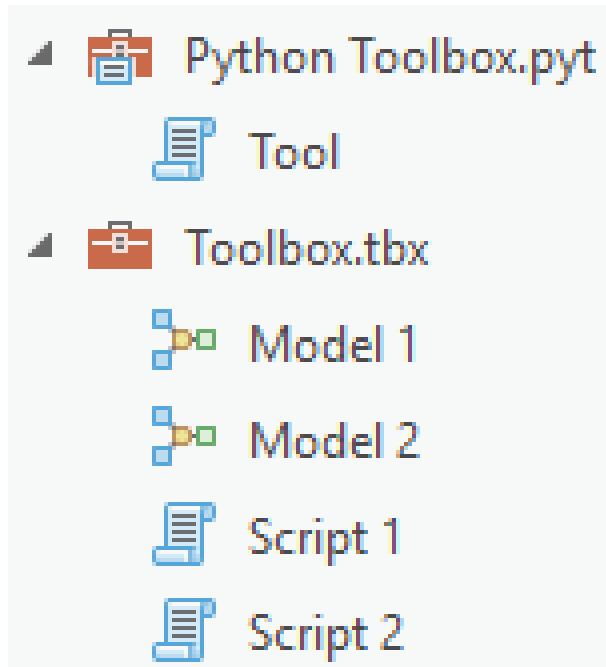
It can be self-contained...

...or import other modules & packages.



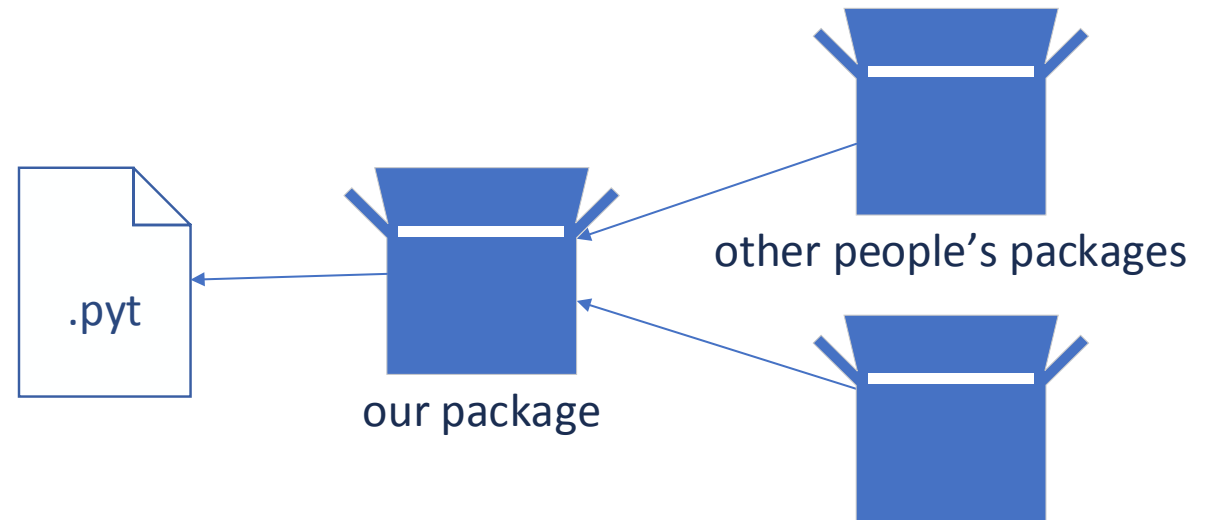
# The Toolbox Menagerie

Distributing these to users can be a pain.  
Updating them is worse.



# Our approach

We'll put all our code in a package, then import it into a Python toolbox.



# Python packages in Pro

ArcGIS Pro comes with lots of handy Python packages that you can import into your code...

Python

```
import pandas
```

# Python packages in Pro

ArcGIS Pro comes with lots of handy Python packages that you can import into your code...  
...but not all of them.

Python

```
import pandas
```

```
import pyodbc
```

```
Traceback (most recent call last):
```

```
  File "<string>", line 1, in <module>
```

```
ModuleNotFoundError: No module named 'pyodbc'
```


# Python packages in Pro


The Python Package Manager makes it easy to get more of them.

## Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can be optionally installed.

Search





Name	Versions
pynacl	1.3.0
pyodbc	4.0.26
pyopengl	3.1.1.1
pyopengl-accelerate	3.1.3.1
py pandoc	1.4
pyprof2calltree	1.4.4
pyproj	2.1.1
pyqtgraph	0.10.0
pyquery	1.4.0
pyramid	1.10.2
pyramid-debugtoolbar	4.4
pyramid-jinja2	2.7
pyramid-mako	1.0.2
pyramid-tm	2.2.1
pyreadline	2.1

**pyodbc**

Version: 4.0.26

DB API Module for ODBC

Homepage

License: MIT

Description

pyodbc is a Python DB API 2 module for ODBC. This project provides an up-to-date, convenient interface to ODBC using native data types like datetime and decimal.

Install



# Python packages in Pro

## Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can be optionally installed.

How about making your own package?

Search

Name	Versions
kealib	1.4.7
keenepyt	0.0.2
keras	2.2.4
keras-applications	1.0.7
keras-base	2.2.4
keras-gpu	2.2.4
keras-preprocessing	1.0.9
kerberos-sspi	0.2
keyrings.alt	3.1.1
keyutils-libs-cos6-i686	1.4
keyutils-libs-cos6-x86_64	1.4
kmod-cos7-ppc64le	20.0
krb5	1.16.1
krb5-libs-cos6-i686	1.10.3
krb5-libs-cos6-x86_64	1.10.3

keenepyt

Version: 0.0.2

Homepage

License:

Description

Package details are not available


Install

# Our approach

Updating will be easy.

## Update Packages

The following list of Python packages are installed with ArcGIS Pro and have recent updates.

Updates: 84 Update All 

Name	Version
jupyter_console	5.2.0
keenepyt	0.0.3
keyring	13.2.1
kiwisolver	1.0.1
libpng	1.6.34
markupsafe	1.0
matplotlib	2.2.3
mistune	0.8.3
mkl	2018.0.3
mkl_fft	1.0.4
mkl_random	1.0.1
more-itertools	4.3.0
mpmath	1.0.0
nbconvert	5.3.1
netcdf4	1.4.1
notebook	5.6.0
numexpr	2.6.8
numpy	1.15.1
numpy-base	1.15.1

### keenepyt

Version: 0.0.3 Update

[Homepage](#) License:

**Description**  
Package details are not available

# Our approach

Esri doesn't want us to modify the default environment, so we cloned it.

## Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can



Note: Cannot modify the default Python environment. Clone and activate a new environment.

# Our approach

Esri doesn't want us to modify the default environment, so we cloned it.  
Think of an “environment” as a Python installation.

## Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can



Note: Cannot modify the default Python environment. Clone and activate a new environment.

# Our approach

Esri doesn't want us to modify the default environment, so we cloned it.

## Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can



Note: Cannot modify the default Python environment. Clone and activate a new environment.

Note: We could get around this restriction by doing it outside of Pro, but we won't.

# How Pro gets packages

What kind of “packages” are these?

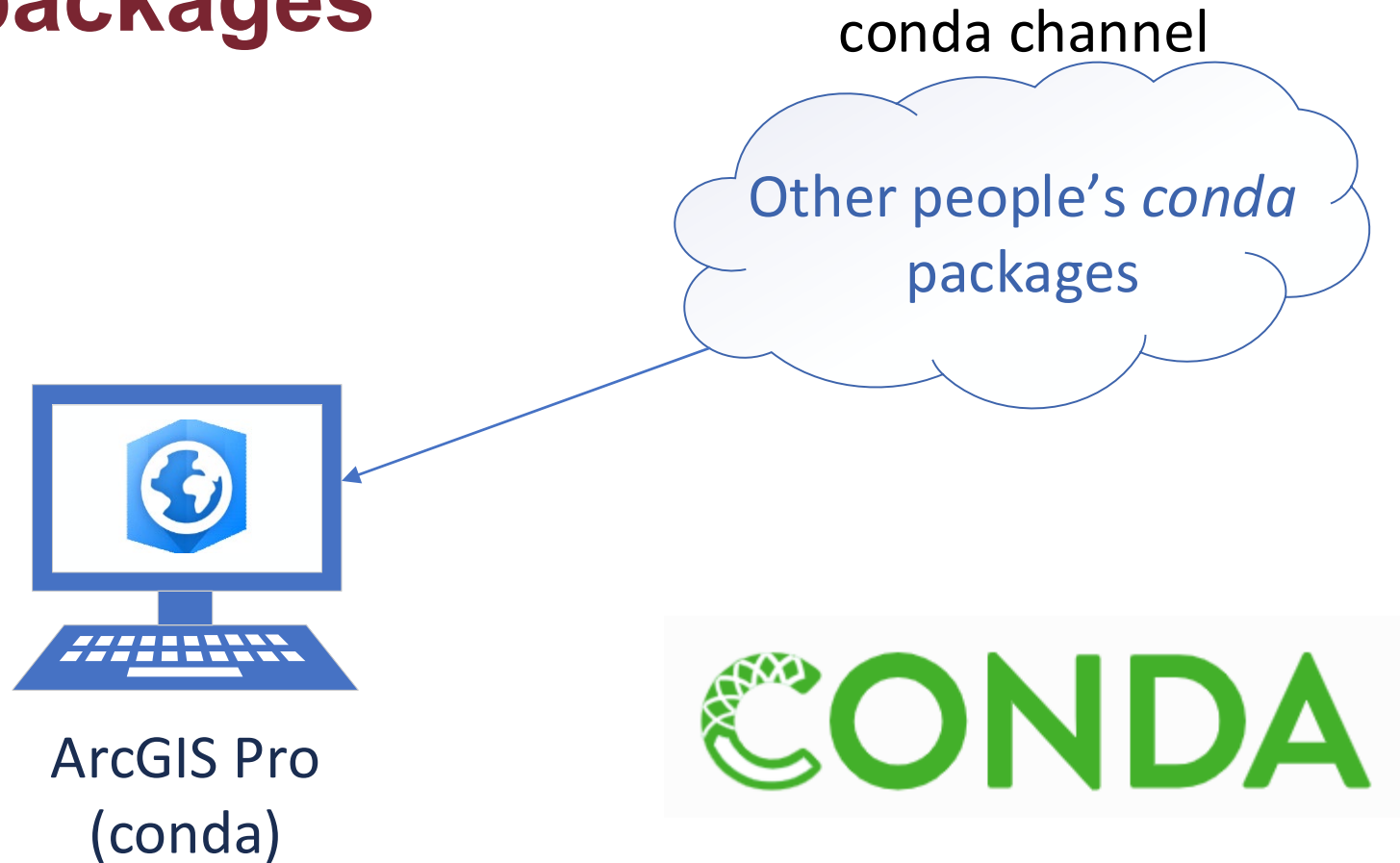


ArcGIS Pro

Other people's packages

# How Pro gets packages

They're conda packages.



# How Pro gets packages

They're conda packages.

Conda is open-source software for managing environments and packages.



ArcGIS Pro  
(conda)

conda channel

Other people's *conda*  
packages

The word "CONDA" in a bold, green, sans-serif font. The letter 'C' is stylized with a green circular arrow around it, and the letter 'O' contains a green DNA double helix pattern.



# How Pro gets packages

They're conda packages.

Conda is open-source software for managing environments and packages.

The "c" is lowercase.



ArcGIS Pro  
(conda)

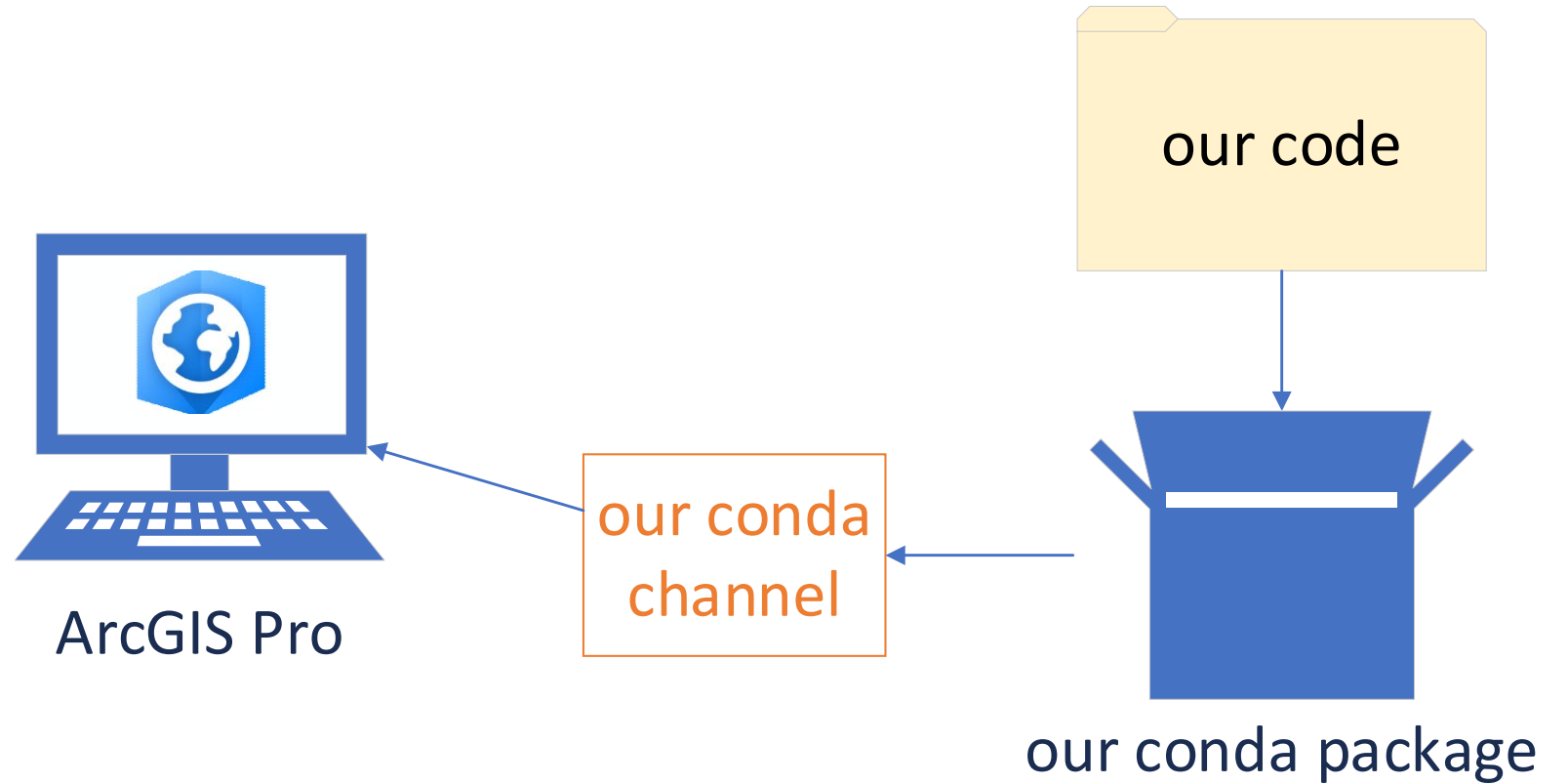
conda channel

Other people's *conda* packages

The word "CONDA" in green, with a green circular icon containing a white DNA helix structure to the left of the "C".

# How Pro gets packages

We'll make our own conda package and channel.



# Our environments

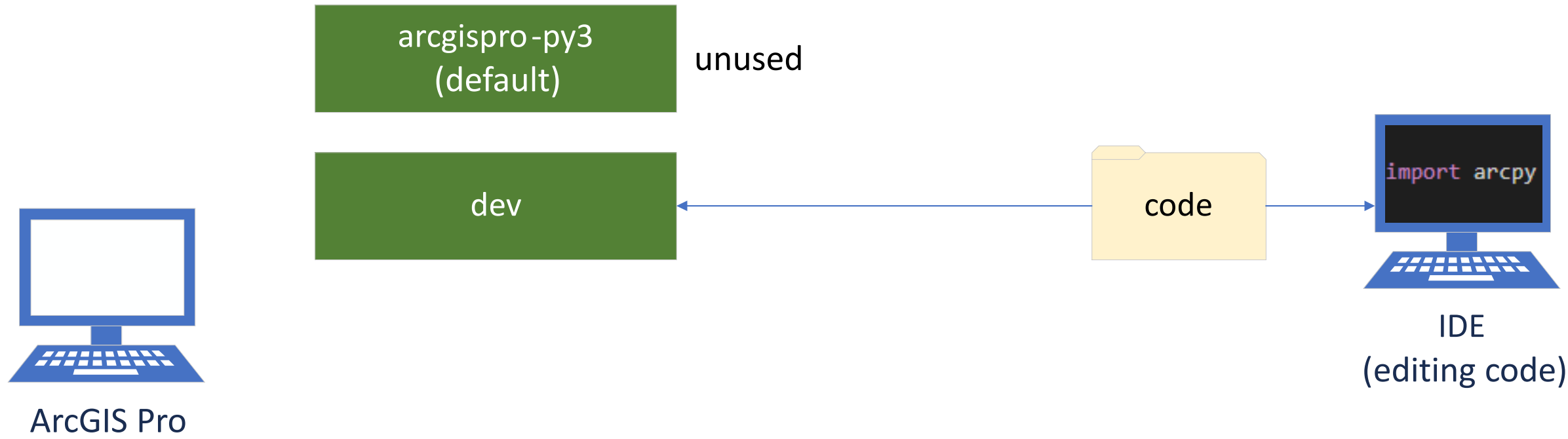
arcgispro-py3  
(default)

unused

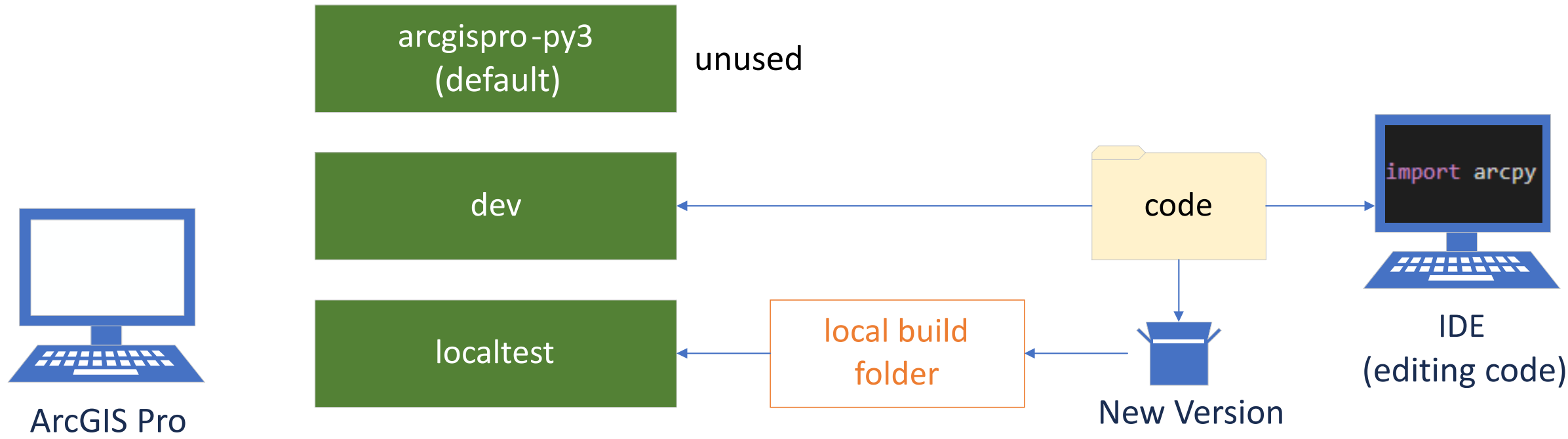


ArcGIS Pro

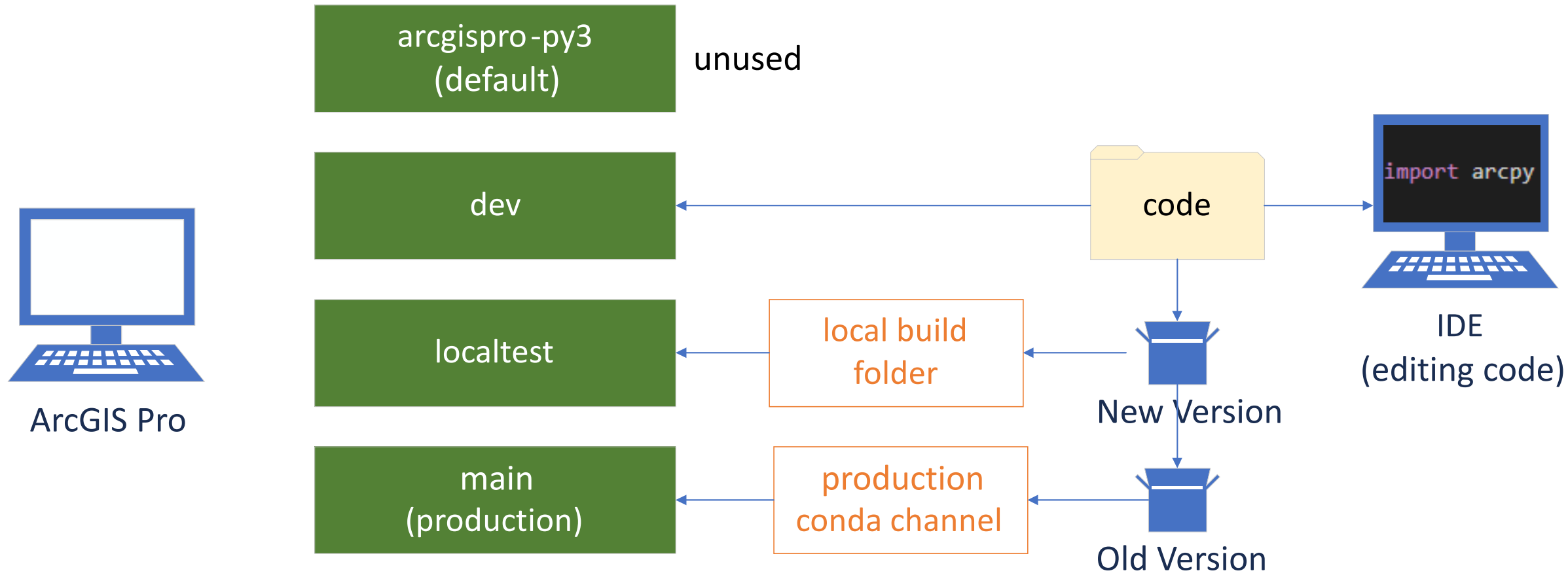
# Our environments



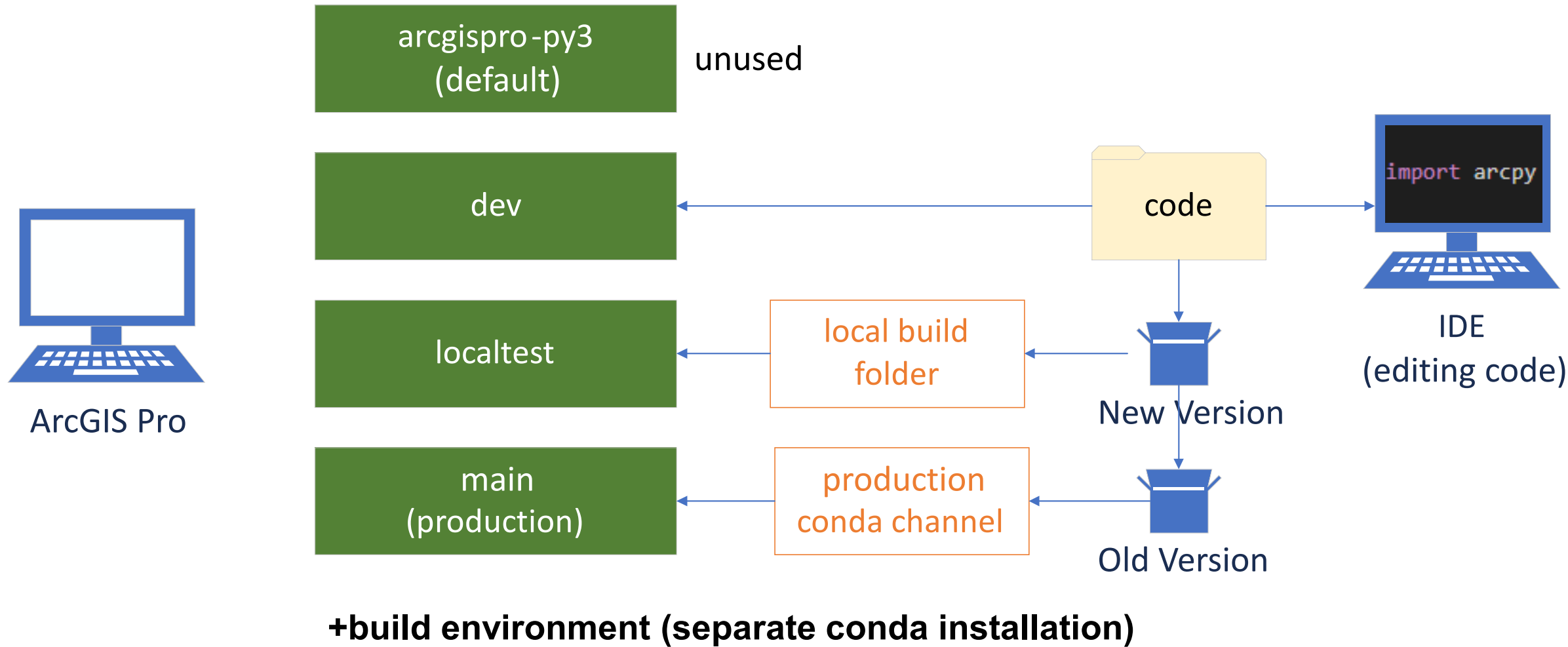
# Our environments



# Our environments



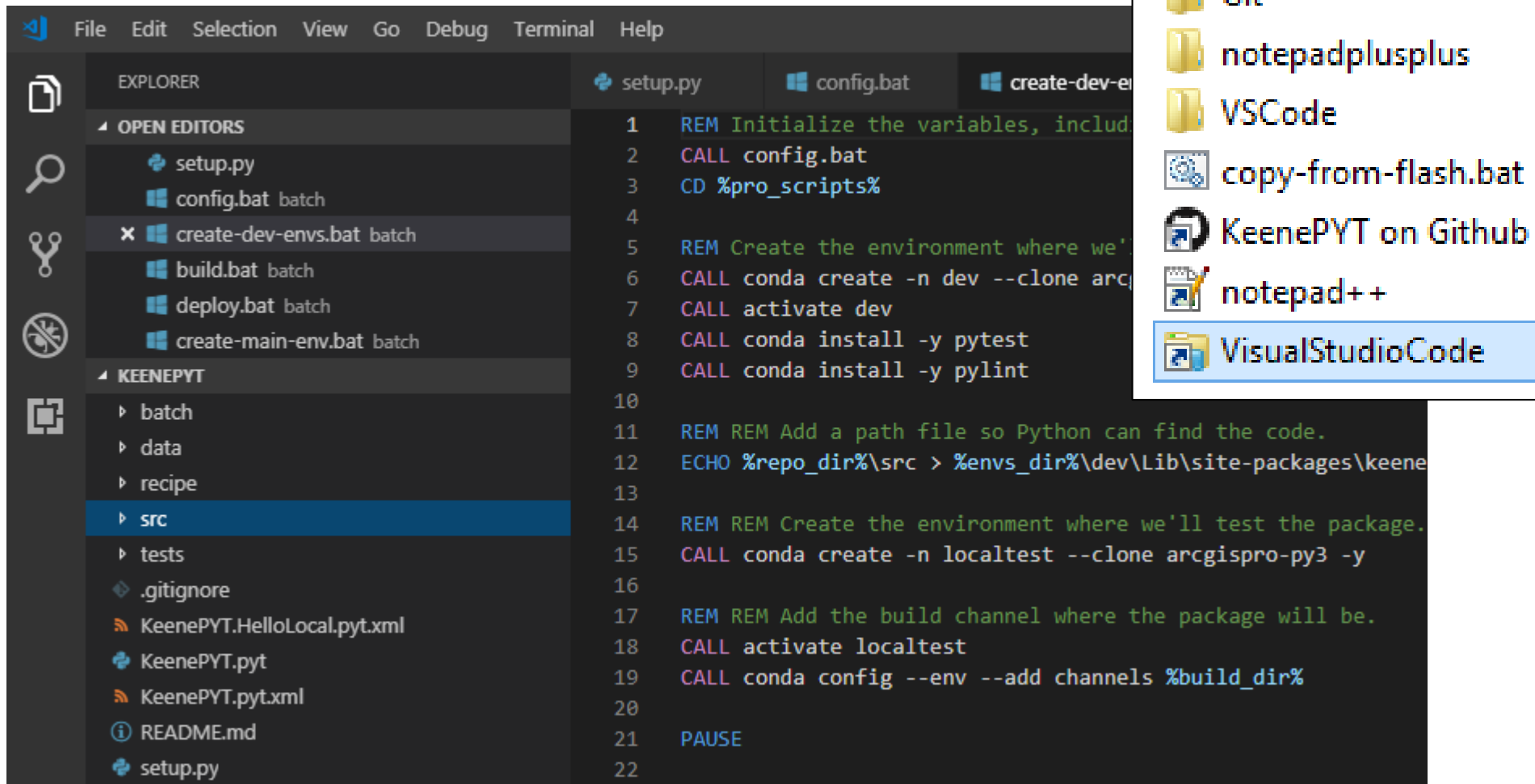
# Our environments



# Open Visual Studio Code

(or use your favorite IDE to open

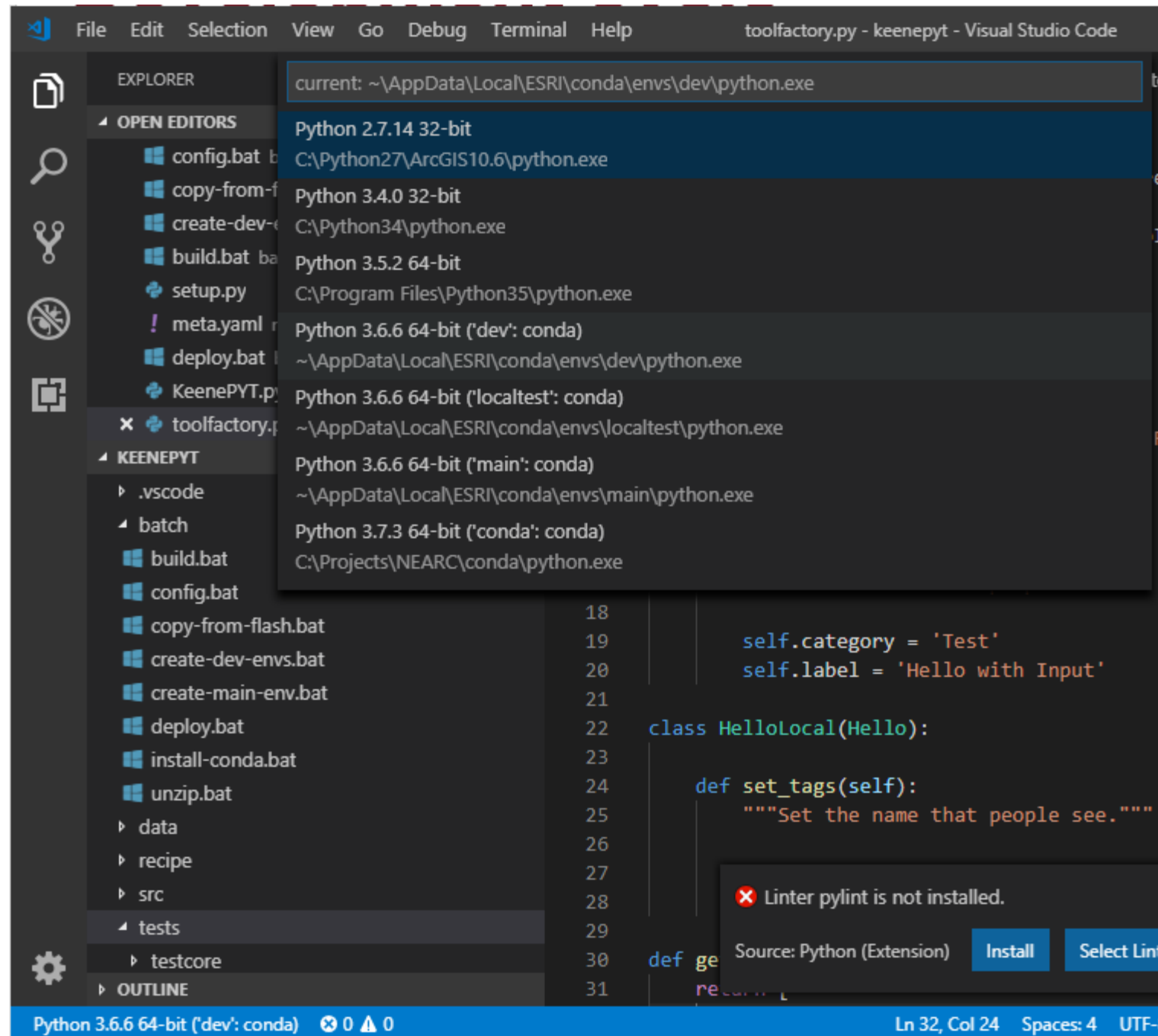
**C:\Projects\NEARC\code\keenepyt)**





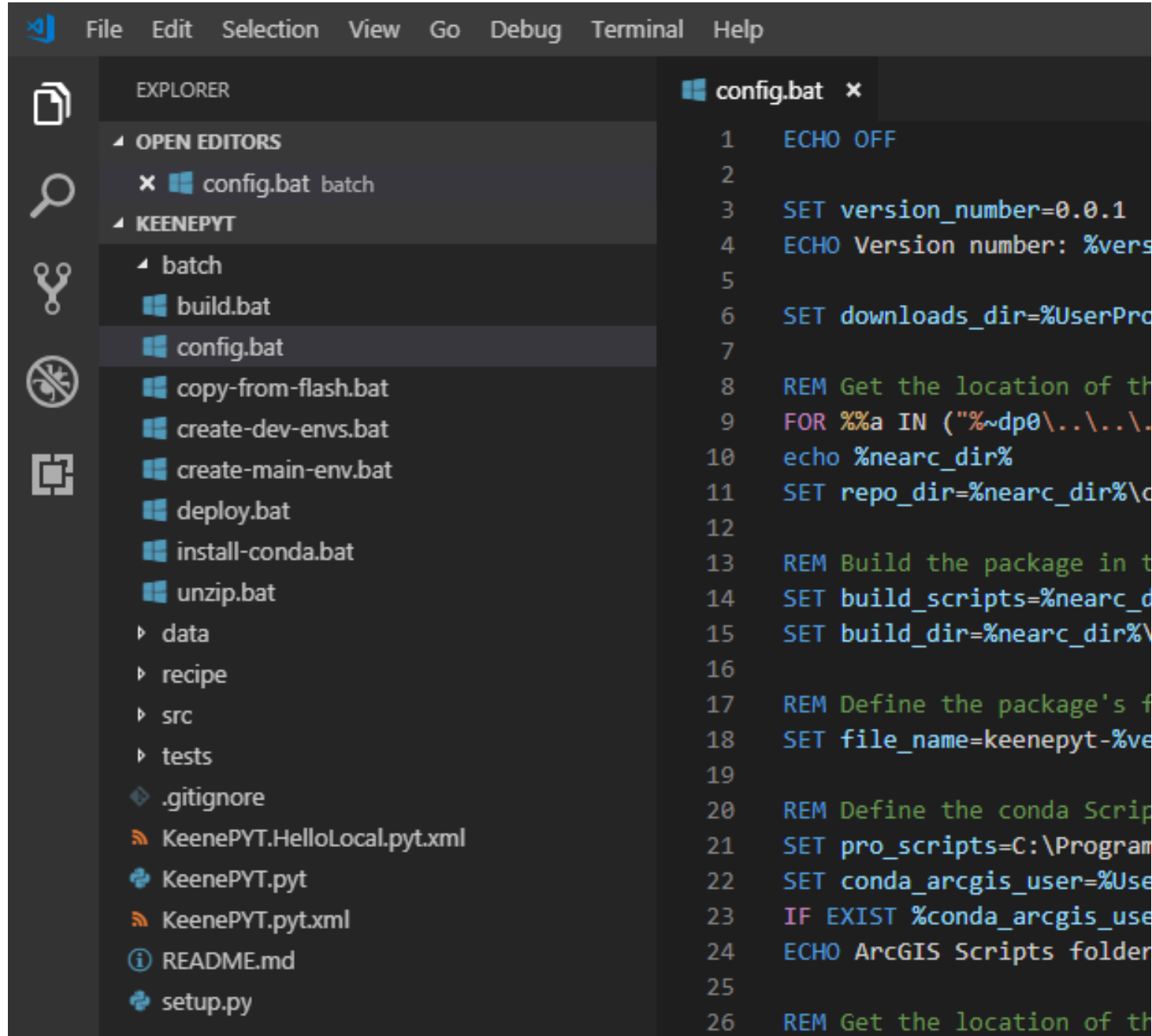
# Set the Python interpreter.

Click on “Python” in the lower-left corner, then choose “dev.”



## View the batch files.

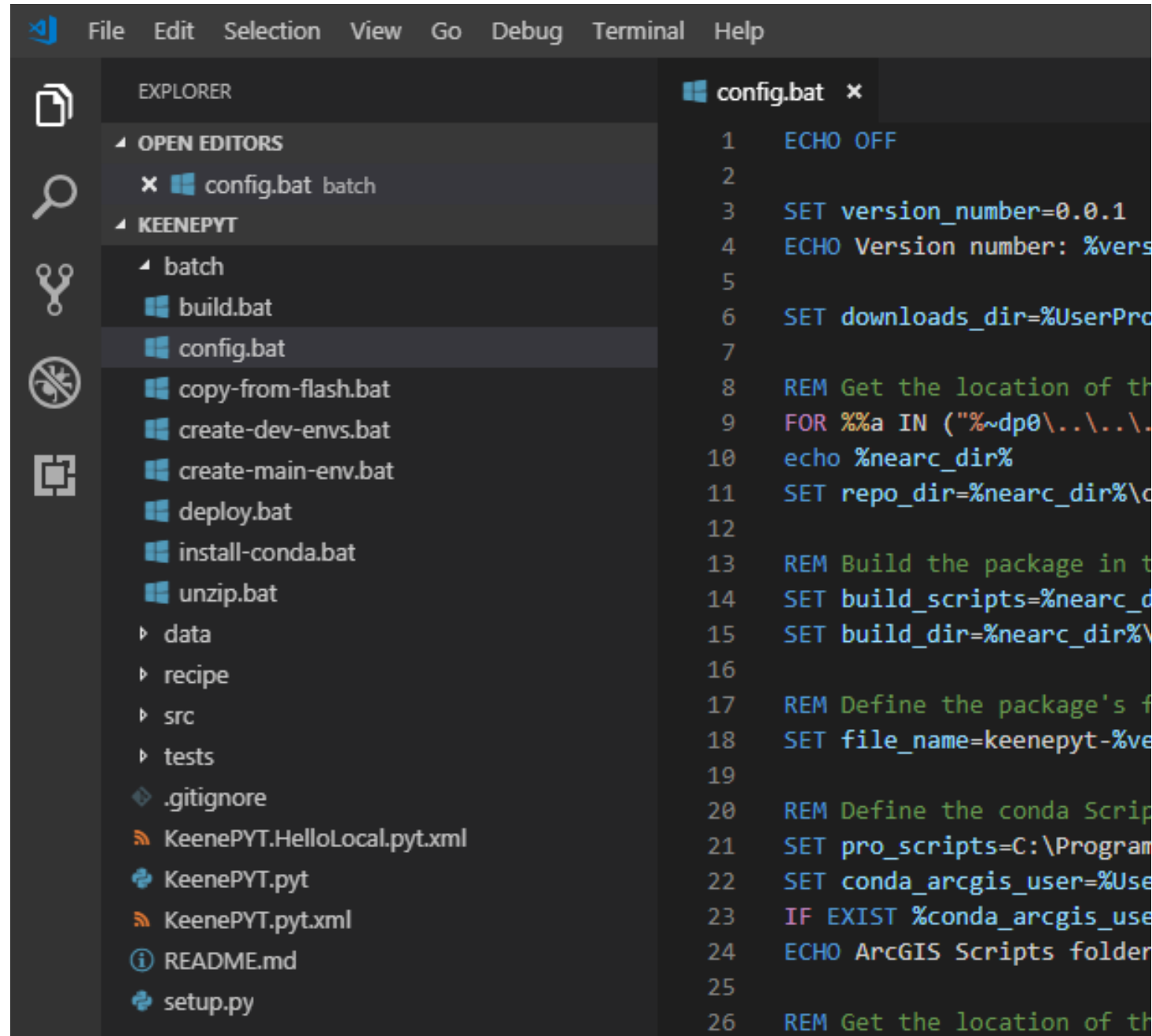
In your IDE, open **batch\config.bat**.



## View the batch files.

In your IDE, open **batch\config.bat**.

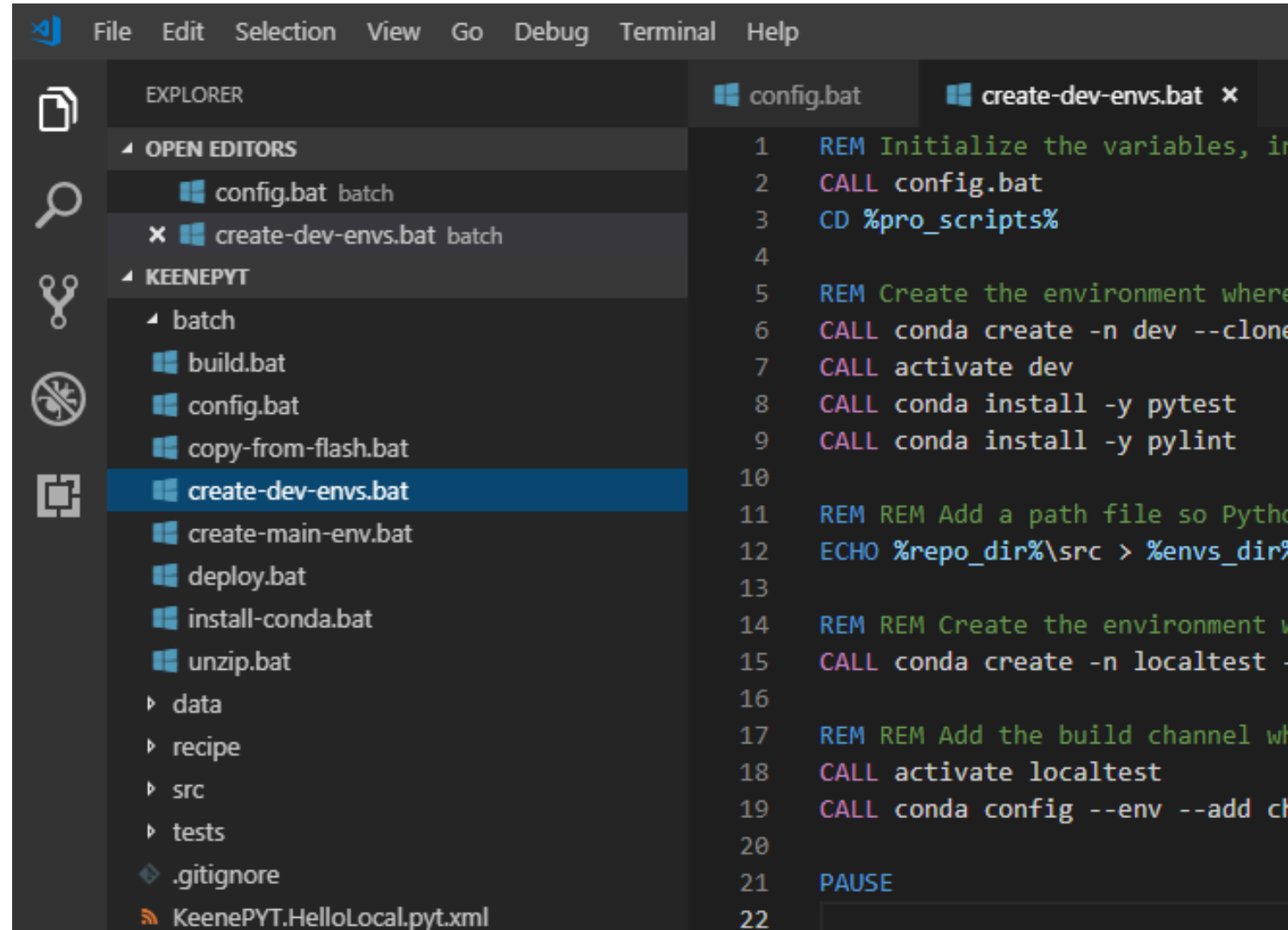
This batch file sets a bunch of variables that other batch files will use, including the version number of our package.



# View the batch files.

Open **create-dev-envs.bat**.

You ran this one earlier.



```
File Edit Selection View Go Debug Terminal Help

EXPLORER
├─ OPEN EDITORS
│   ├── config.bat batch
│   └─ ✕ create-dev-envs.bat batch
├─ KEENEPTY
│   └─ batch
│       ├── build.bat
│       ├── config.bat
│       ├── copy-from-flash.bat
│       └─ create-dev-envs.bat
│           ├── create-main-env.bat
│           ├── deploy.bat
│           ├── install-conda.bat
│           ├── unzip.bat
│           ├── data
│           ├── recipe
│           ├── src
│           ├── tests
│           ├── .gitignore
│           └─ KeenePYT.HelloLocal.pyt.xml
└─
```

```
1  REM Initialize the variables, in
2  CALL config.bat
3  CD %pro_scripts%
4
5  REM Create the environment where
6  CALL conda create -n dev --clone
7  CALL activate dev
8  CALL conda install -y pytest
9  CALL conda install -y pylint
10
11  REM REM Add a path file so Python
12  ECHO %repo_dir%\src > %envs_dir%
13
14  REM REM Create the environment w
15  CALL conda create -n localtest -
16
17  REM REM Add the build channel wh
18  CALL activate localtest
19  CALL conda config --env --add ch
20
21  PAUSE
22
```

# View the batch files.

Open **create-dev-envs.bat**.







Note this line:

```
11  REM Add a path file so Python can find the code.  
12  %repo_dir%\src > %envs_dir%\dev\Lib\site-packages\keenepyt.pth
```

It creates a path file (.pth) that contains the path to your code.

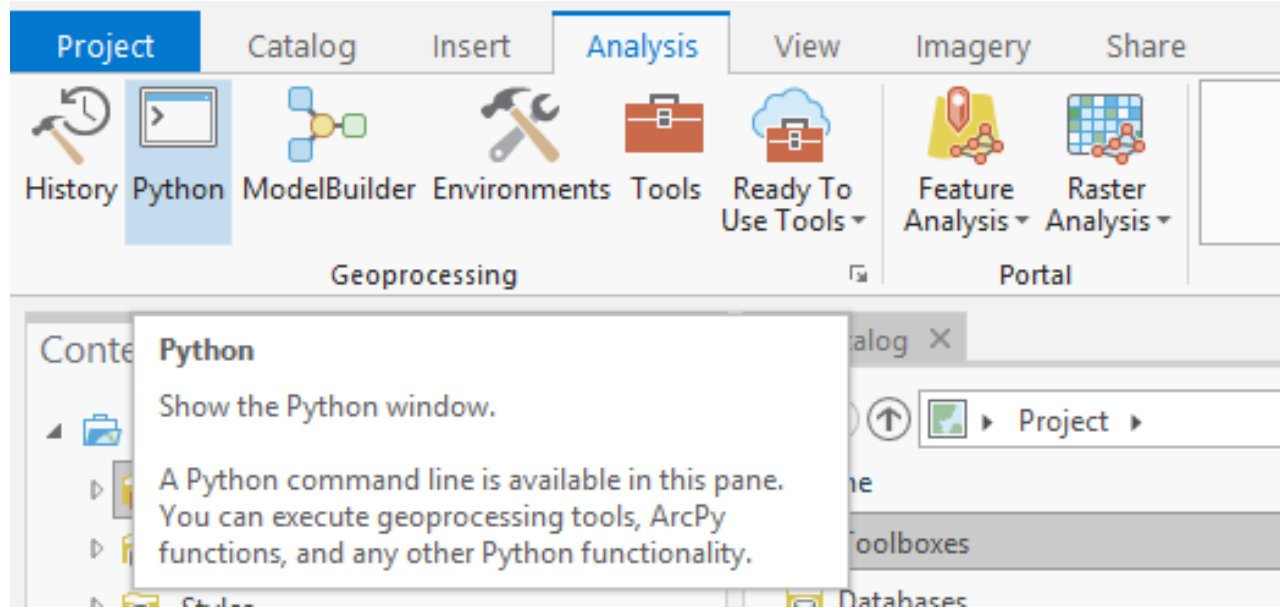
# Activate the dev environment.

Open ArcGIS Pro, and use the Python Package Manager to activate dev.

Active	Environments	Clone	Remove
<input type="radio"/>	<b>arcgispro-py3</b> C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3		
<input checked="" type="radio"/>	<b>dev</b> C:\Users\jswise\AppData\Local\ESRI\conda\envs\dev		
<input type="radio"/>	<b>localtest</b> C:\Users\jswise\AppData\Local\ESRI\conda\envs\localtest		

# Try importing keenepyt.

This should work in the dev environment.

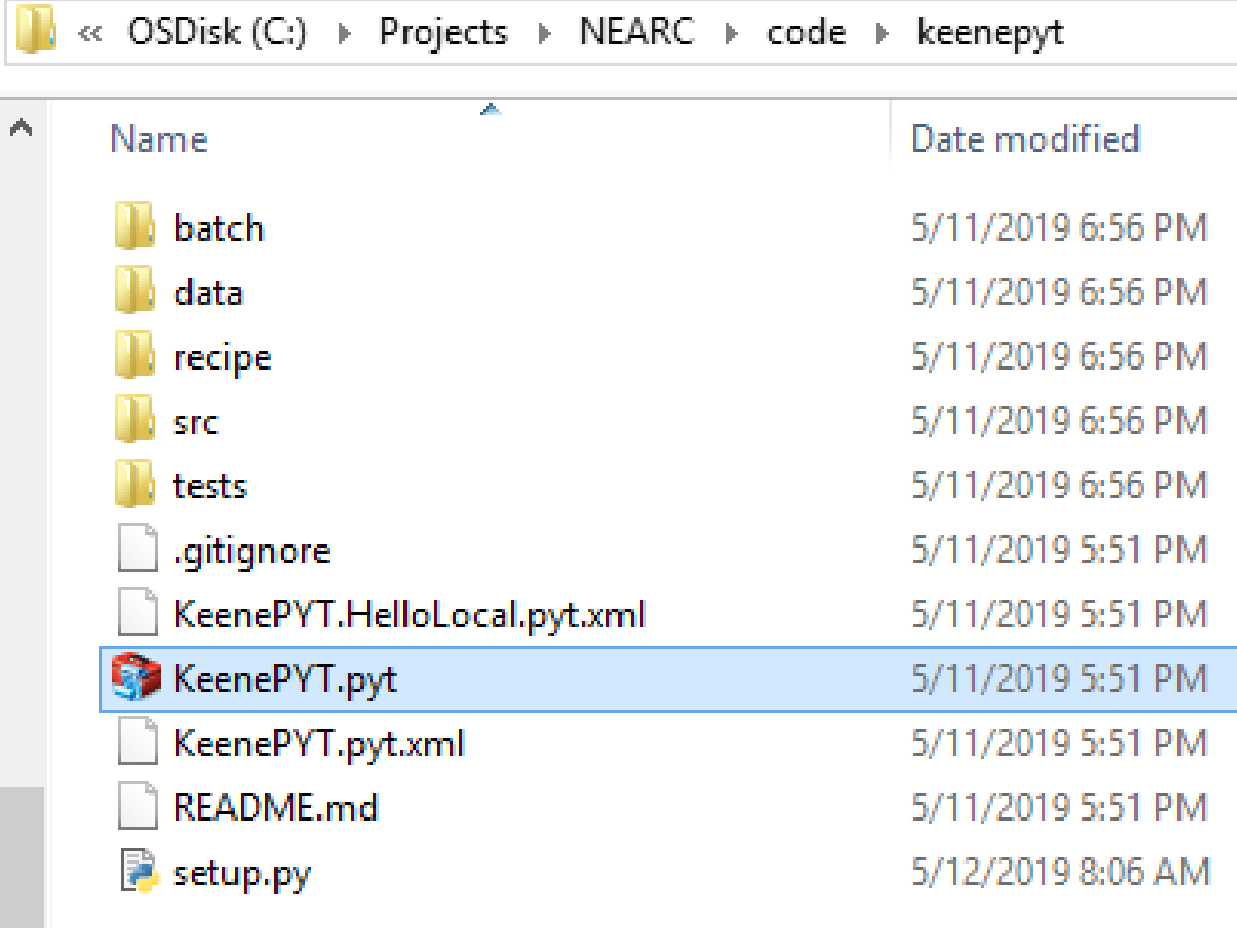


```
import keenepyt
```

# Pretend to distribute the toolbox.

Find KeenePYT.pyt, and copy it to some other place.

This simulates giving your toolbox to somebody.



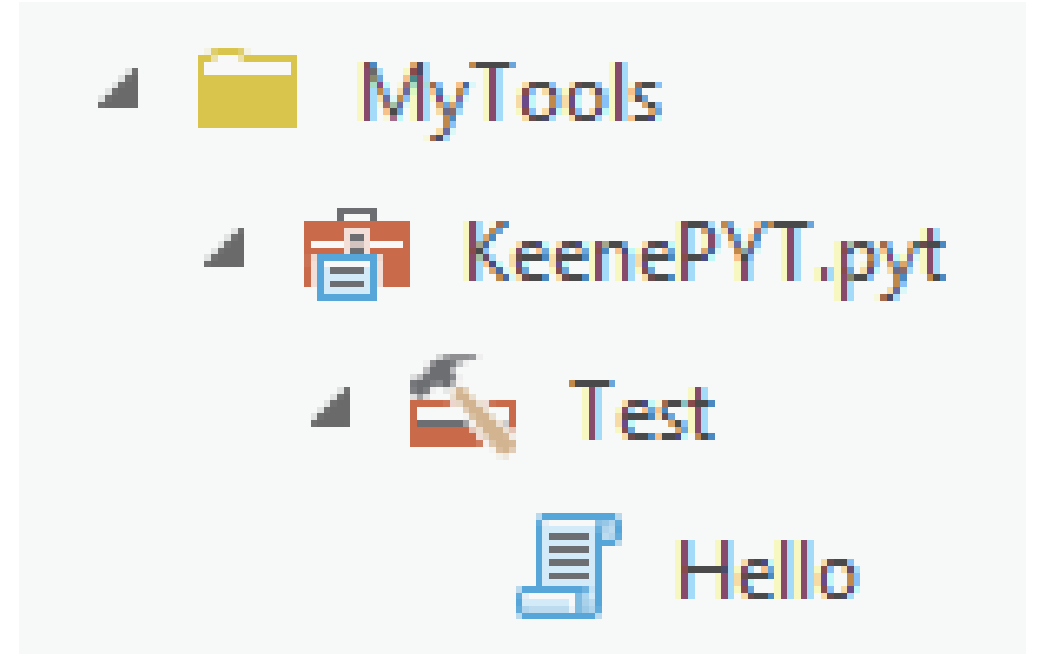
OSDisk (C:) > Projects > NEARC > code > keenepyt

Name	Date modified
batch	5/11/2019 6:56 PM
data	5/11/2019 6:56 PM
recipe	5/11/2019 6:56 PM
src	5/11/2019 6:56 PM
tests	5/11/2019 6:56 PM
.gitignore	5/11/2019 5:51 PM
KeenePYT.HelloLocal.pyt.xml	5/11/2019 5:51 PM
<b>KeenePYT.pyt</b>	5/11/2019 5:51 PM
KeenePYT.pyt.xml	5/11/2019 5:51 PM
README.md	5/11/2019 5:51 PM
setup.py	5/12/2019 8:06 AM



# Open the toolbox.

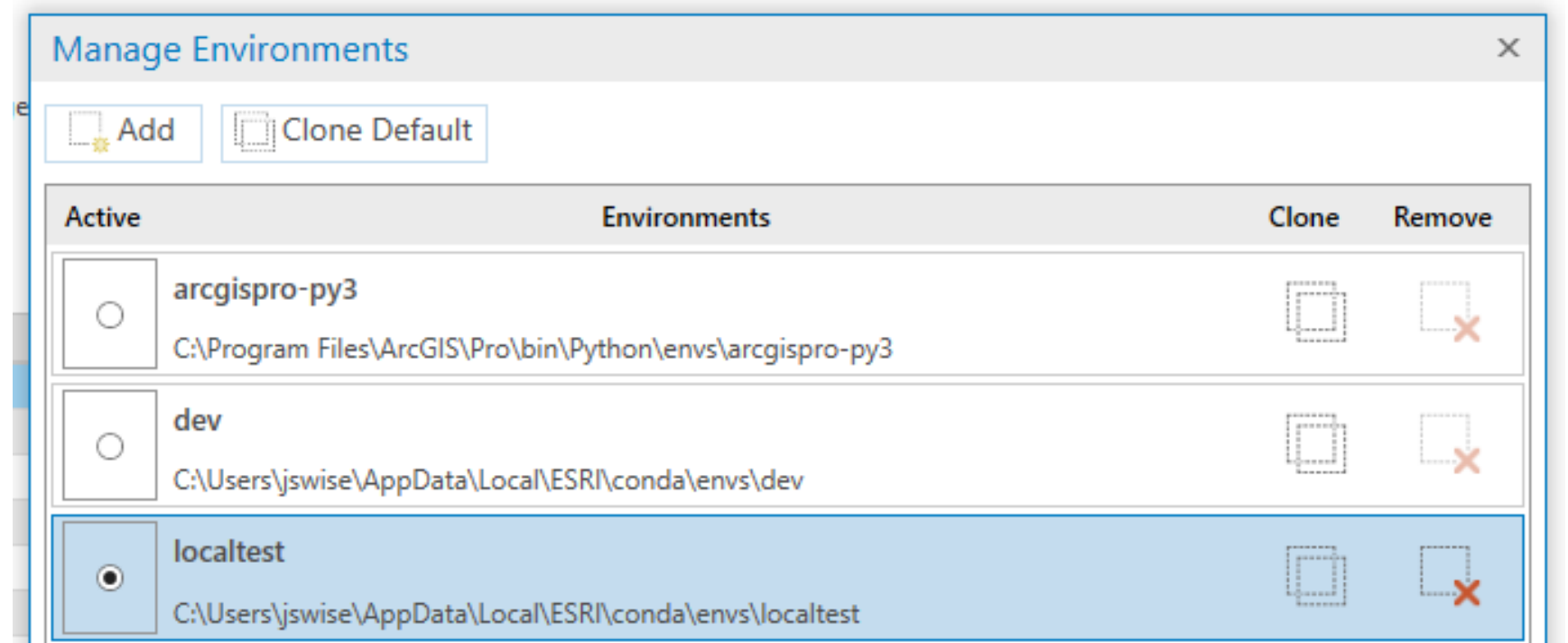
You should be able to open the toolbox in Pro.



# Change environments.

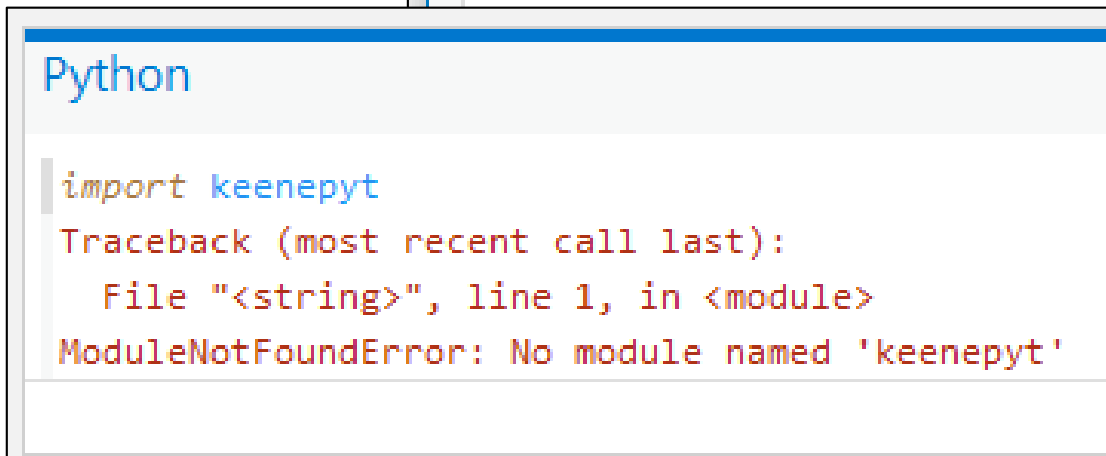
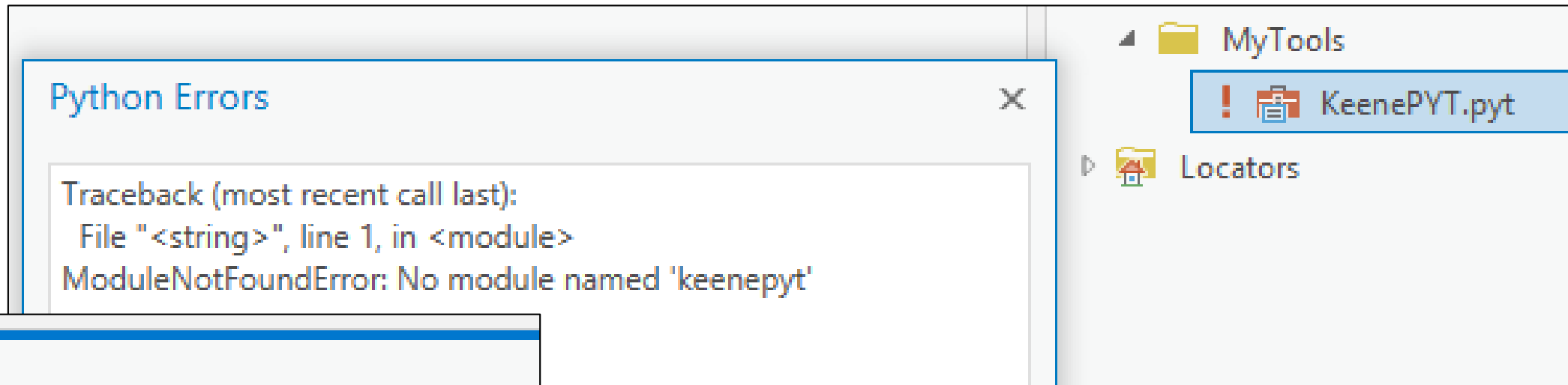
Activate localtest in Pro.

Restart Pro.



# Your toolbox is now broken.

The localtest environment doesn't know about your code.



# Look at the toolbox code.

It's short.

It imports *keenepyt.tools.toolfactory* and runs the *get\_tools* method.

```
1  from keenepyt.tools.toolfactory import * # pylint: disable=unused-wildcard-import
2
3  class Toolbox(object):
4      def __init__(self):
5          """Define the toolbox (the name of the toolbox is the name of the
6             .pyt file)."""
7          self.label = 'KeenePYT Tools'
8          self.alias = 'KeenePYT'
9          self.description = 'Demonstration code from a workshop at Spring NEARC 2019.'
10
11         self.tools = get_tools()
12
```

# Look at the toolbox code.

The `get_tools` method in `keenepyt.tools.toolfactory` provides a list of tool classes.

Some are commented out so we can have fun adding them later.

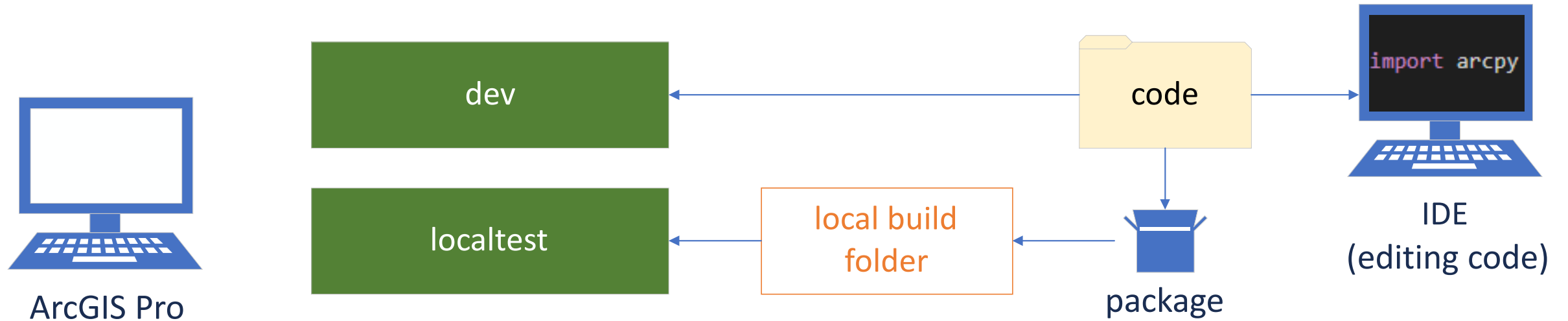
```
30 def get_tools():
31     return [
32         # GetAirepsLocal,
33         # HelloInputLocal,
34         HelloLocal
35     ]
```

# Why is it broken?

In **create-dev-envs.bat**, we told the localtest environment where to find a conda channel instead of the code. There's no package in the channel yet.

```
17  REM Add the build channel where the package will be.  
18  CALL activate localtest  
19  CALL conda config --env --add channels %build_dir%
```

# Now we need to build the package.











# Run build.bat.

This creates a package in

**C:\Projects\NEARC\conda\conda-bld\win-64.**

```
1  REM Initialize the variables, including the
2  CALL config.bat
3
4  REM Build the package.
5  CD %build_scripts%
6  CALL activate base
7  CALL conda build --py 3.6 --croot "%build_c
8  CALL conda build purge
9
10 PAUSE
```

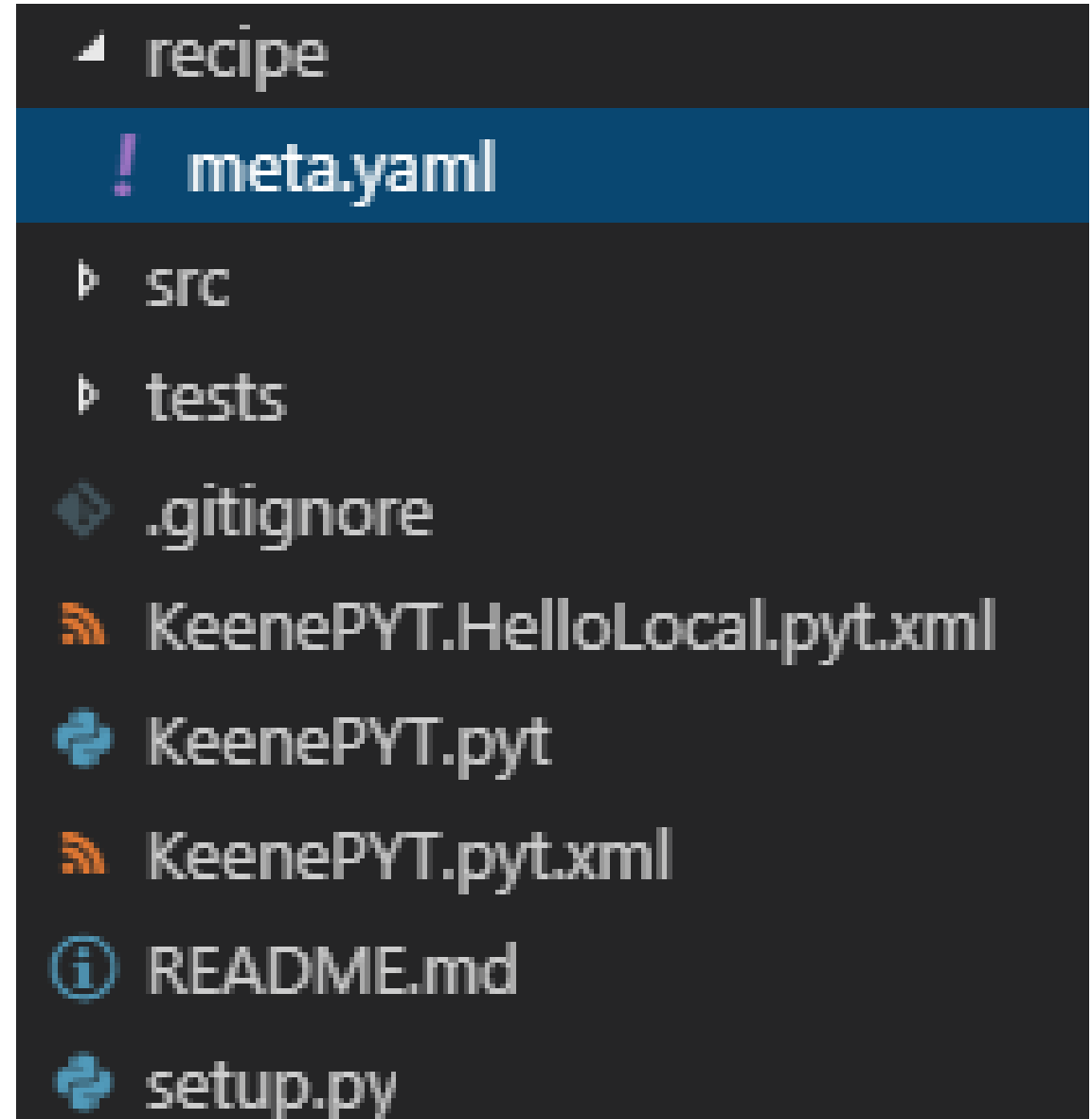
« OSDisk (C:) ▶ Projects ▶ NEARC ▶ code ▶ keenepyt ▶ batch	
Name	Date modified
 build.bat	5/12/2019 8:57 AM
 config.bat	5/12/2019 9:47 AM
 copy-from-flash.bat	5/11/2019 5:51 PM
 create-dev-envs.bat	5/12/2019 10:25 AM
 create-main-env.bat	5/12/2019 9:14 AM
 deploy.bat	5/12/2019 9:00 AM
 install-conda.bat	5/11/2019 5:51 PM
 unzip.bat	5/11/2019 5:51 PM



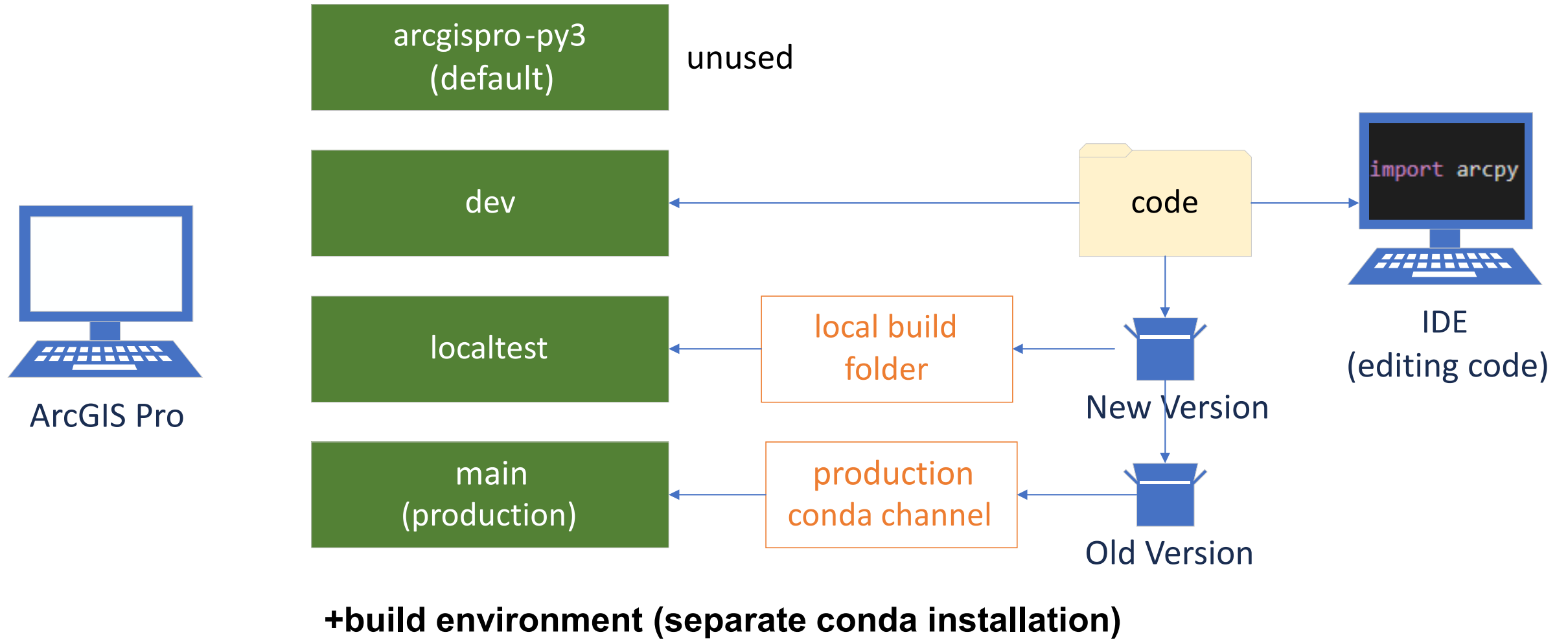
# Run build.bat.

The build process depends on **recipe\meta.yaml** and **setup.py**.

We're using a separate conda installation, because we can't do it in ArcGIS Pro's conda installation.



# Our environments (review)




# Install the package.

You should now be able to install keenepyt in localtest.

This will fix the toolbox.

## Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can be optionally installed.

Search 	
Name	Versions
kealib	1.4.7
keenepyt	0.0.1
keras	2.2.4
keras-applications	1.0.7
keras-base	2.2.4
keras-gpu	2.2.4
keras-preprocessing	1.0.9
kerberos-sspi	0.2
keyrings.alt	3.1.1
keyutils-libs-cos6-i686	1.4
keyutils-libs-cos6-x86_64	1.4
kmod-cos7-ppc64le	20.0
krb5	1.16.1
krb5-libs-cos6-i686	1.10.3
krb5-libs-cos6-x86_64	1.10.3

### keenepyt

Install

Version: 0.0.1

[Homepage](#)

License:









#### Description

Package details are not available

# Create the production environment.

In **C:\Projects\NEARC\code\keenepyt\batch**, run **create-main-env.bat**.

This will clone your default Python environment another new environment, *main*.









Active	Environments	Clone	Remove
<input type="radio"/>	<b>arcgispro-py3</b> C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3		
<input type="radio"/>	<b>dev</b> C:\Users\jswise\AppData\Local\ESRI\conda\envs\dev		
<input type="radio"/>	<b>localtest</b> C:\Users\jswise\AppData\Local\ESRI\conda\envs\localtest		
<input checked="" type="radio"/>	<b>main</b> C:\Users\jswise\AppData\Local\ESRI\conda\envs\main		

# Deploy the package.

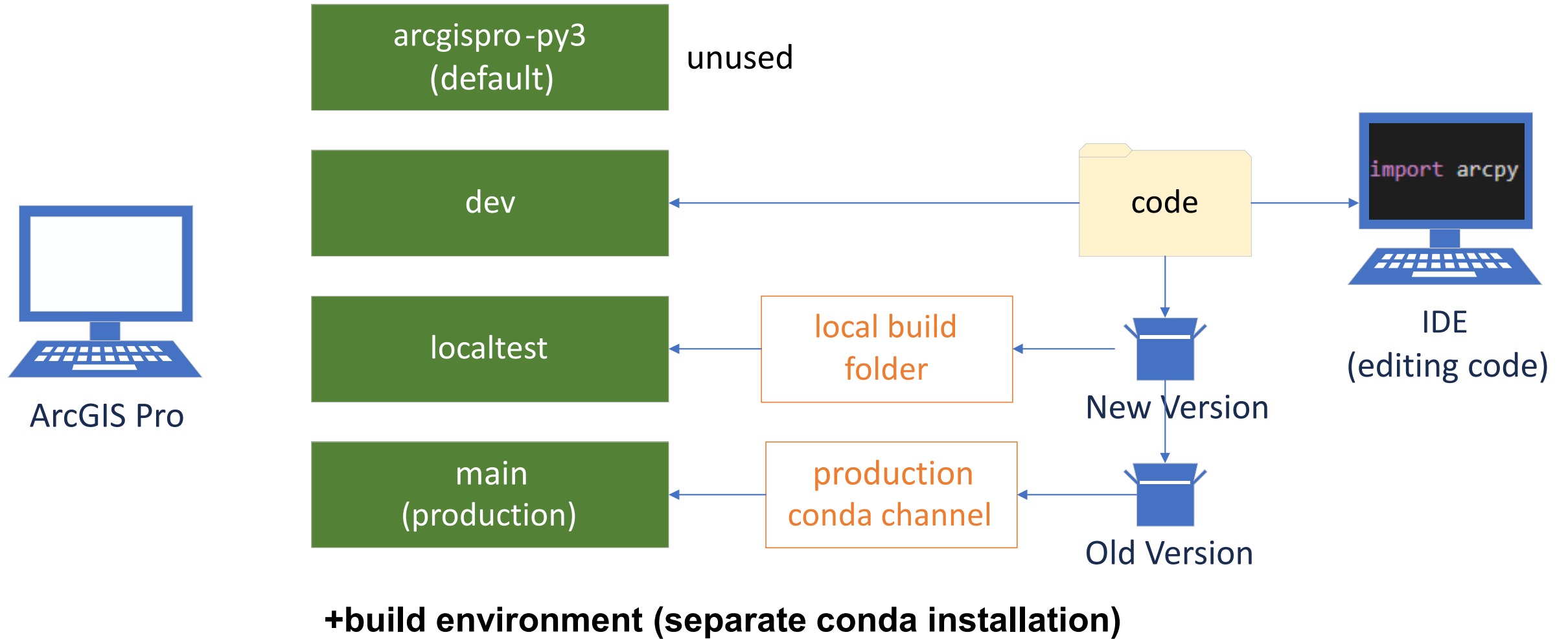
Run **deploy.bat**.

This creates a new channel,  
**C:\Projects\NEARC\channel.**

In real life, this would be on a server that your coworkers can see.

<< OSDisk (C:) ▶ Projects ▶ NEARC ▶ code ▶ keenepyt ▶ batch	
Name	Date modified
 build.bat	5/12/2019 8:57 AM
 config.bat	5/12/2019 9:47 AM
 copy-from-flash.bat	5/11/2019 5:51 PM
 create-dev-envs.bat	5/12/2019 10:25 AM
 create-main-env.bat	5/12/2019 9:14 AM
 <b>deploy.bat</b>	5/12/2019 9:00 AM
 install-conda.bat	5/11/2019 5:51 PM
 unzip.bat	5/11/2019 5:51 PM

# Our environments (review)



# Install the package in main.

In real life, you might want to name this environment with the name of your organization.



# Improve the code.

Add another tool by uncommenting it in *src.keenepyt.tools.toolfactory.get\_tools*.

```
30 def get_tools():
31     return [
32         # GetAirepsLocal,
33         HelloInputLocal,
34         HelloLocal
35     ]
```



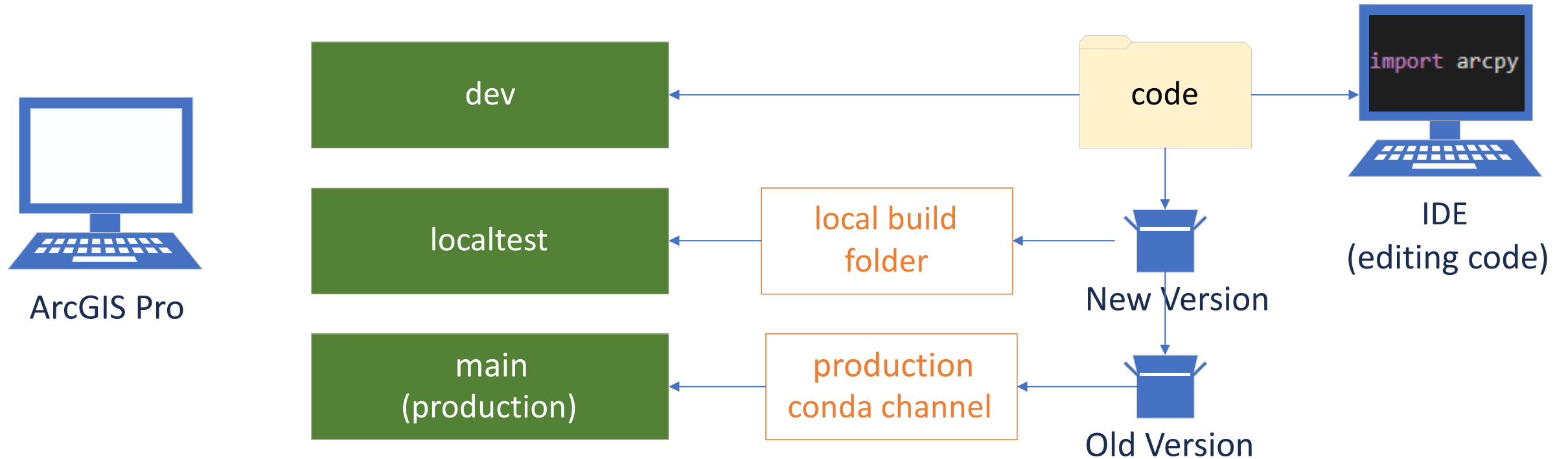
# Improve the code.

In config.bat, update the version number from 0.0.1 to 0.0.2.

```
1  ECHO OFF
2
3  SET version_number=0.0.2
4  ECHO Version number: %version_number%
5
```

# Run build.bat again.

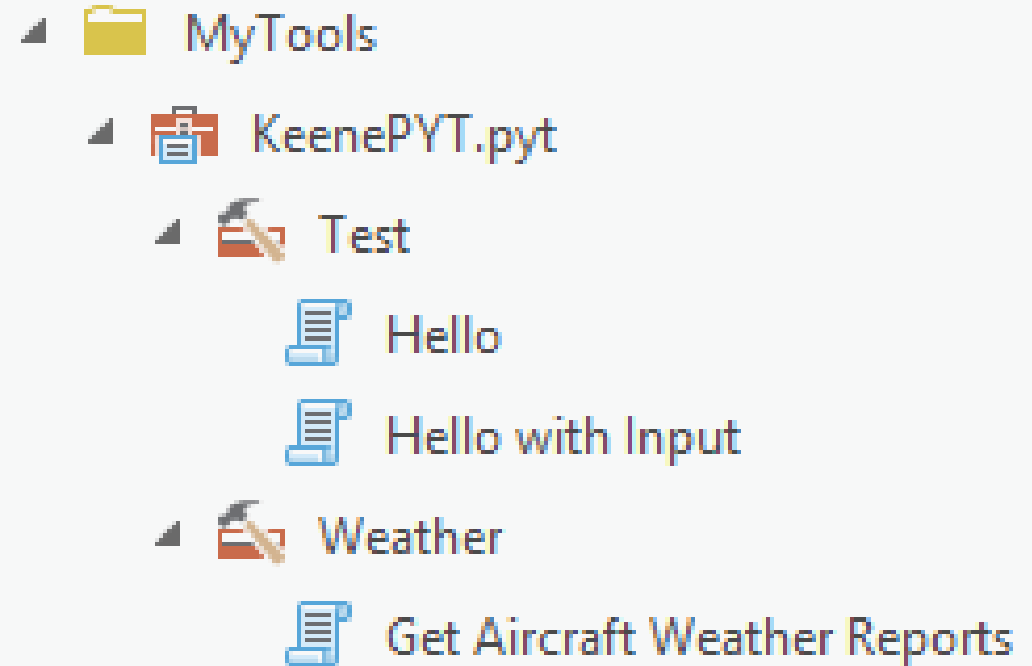
Now localtest is different from main.



# Add another tool.

Now dev, localtest, and main are all different.  
Restart Pro to see the change.

```
30 def get_tools():
31     return [
32         GetAirepsLocal,
33         HelloInputLocal,
34         HelloLocal
35     ]
```



# Add another tool.

Update the version number again.

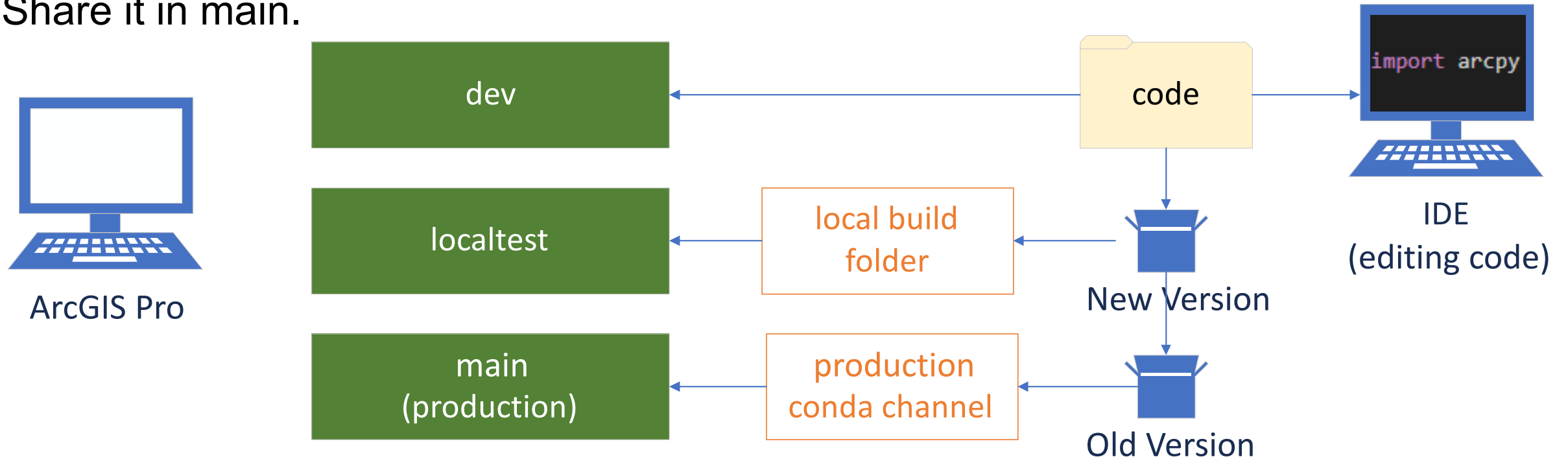
```
1  ECHO OFF
2
3  SET version_number=0.0.3
4  ECHO Version number: %version_number%
5
```

# Development cycle

Write & test code in dev.


Test the package in localtest.

Share it in main.



# More Python fun

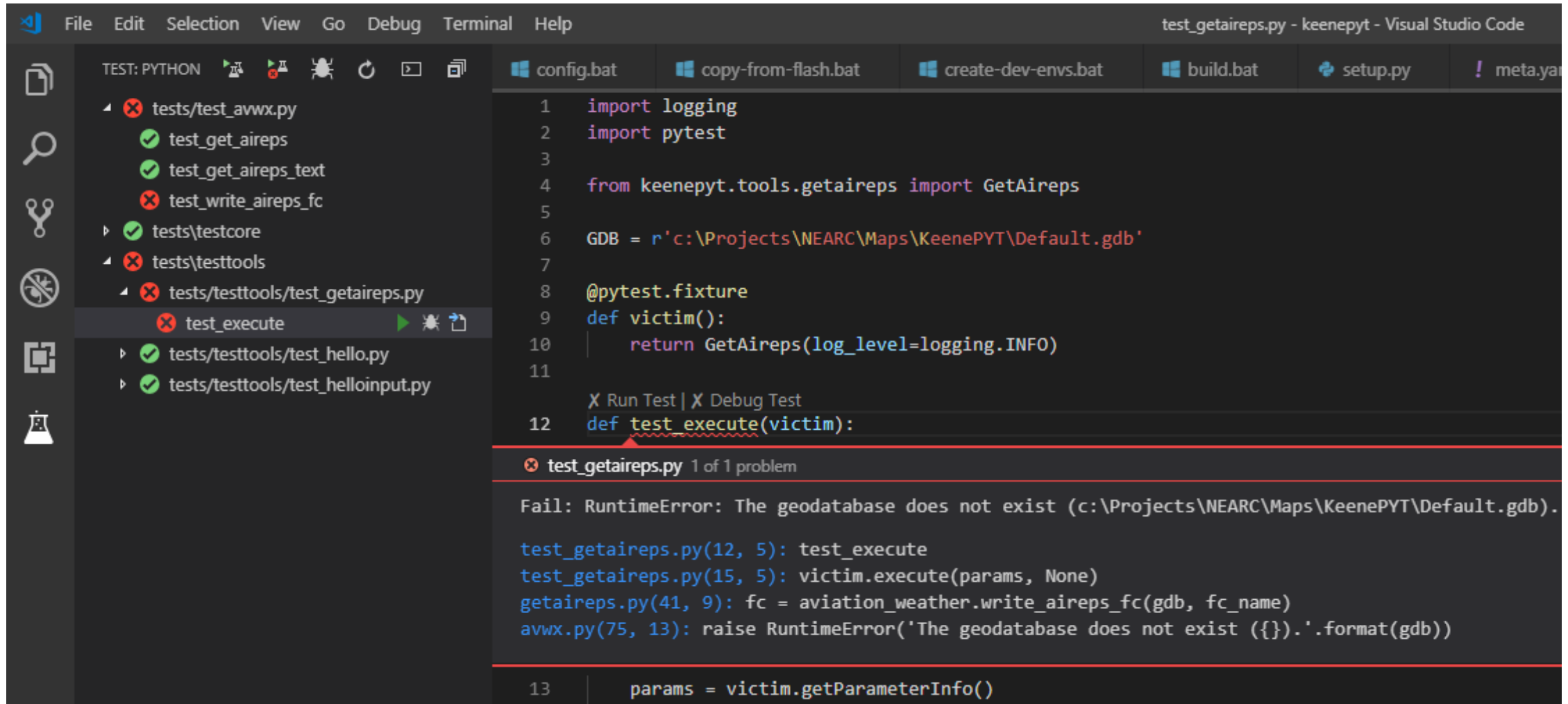
Check out the log file.

▸ This PC ▸ OSDisk (C:) ▸ Users ▸ jswise ▸ AppData ▸ Local ▸ KeenePYT ▸ Logs		
Name	Date modified	Type
 KeeneLog	5/12/2019 11:25 AM	File

```
2019-05-12 11:12:19,334 INFO: Caller: ArcGIS
2019-05-12 11:13:36,651 INFO: Writing C:\Users\jswise\Documents\ArcGIS\Projects\MyProject\MyProject.gdb\Aireps.
2019-05-12 11:25:20,334 INFO: _____
```

# More Python fun

Use PyTest to run unit tests.



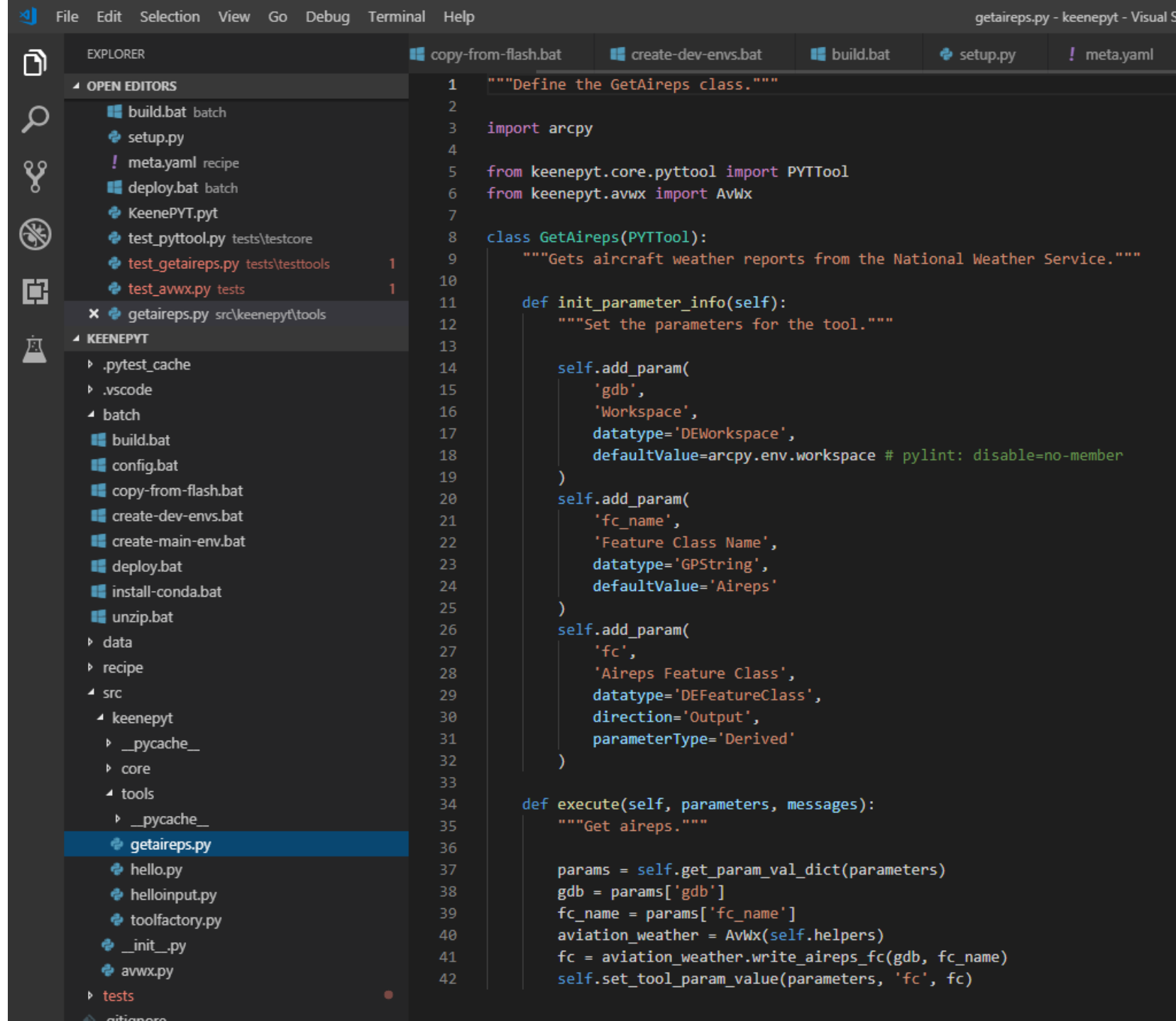
The screenshot shows the Visual Studio Code interface with the file `test_getaireps.py` open. The left sidebar displays a file explorer with a tree view of test files. The main editor shows the code for `test_getaireps.py`, which imports `logging` and `pytest`, and defines a `test_execute` function. A red error bar at the bottom indicates a `RuntimeError: The geodatabase does not exist (c:\Projects\NEARC\Maps\KeenePYT\Default.gdb).` The error message includes the stack trace:

```
test_getaireps.py(12, 5): test_execute
test_getaireps.py(15, 5): victim.execute(params, None)
getaireps.py(41, 9): fc = aviation_weather.write_aireps_fc(gdb, fc_name)
avwx.py(75, 13): raise RuntimeError('The geodatabase does not exist ({})'.format(gdb))
```

# More Python fun

See how these tools work.

They're not like the examples you'll see elsewhere.



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure with folders like .pytest\_cache, .vscode, batch, data, recipe, src, and tests. The file getaireps.py is selected under the src directory. The main editor window shows the code for getaireps.py, which defines a GetAireps class. The code includes imports for arcpy and modules from keenepyt, and it uses a helper class AvWx. The class GetAireps inherits from PYTTool and has an init\_parameter\_info method that defines parameters for gdb, workspace, and feature class name. It also has an execute method that uses these parameters to write aircraft weather data.

```
File Edit Selection View Go Debug Terminal Help
getaireps.py - keenepyt - Visual S

EXPLORER
copy-from-flash.bat create-dev-envs.bat build.bat setup.py meta.yaml

OPEN EDITORS
build.bat batch
setup.py
meta.yaml recipe
deploy.bat batch
KeenePYT.pyt
test_pyttool.py tests\testcore
test_getaireps.py tests\testtools 1
test_avwx.py tests 1

getaireps.py src\keenepyt\tools

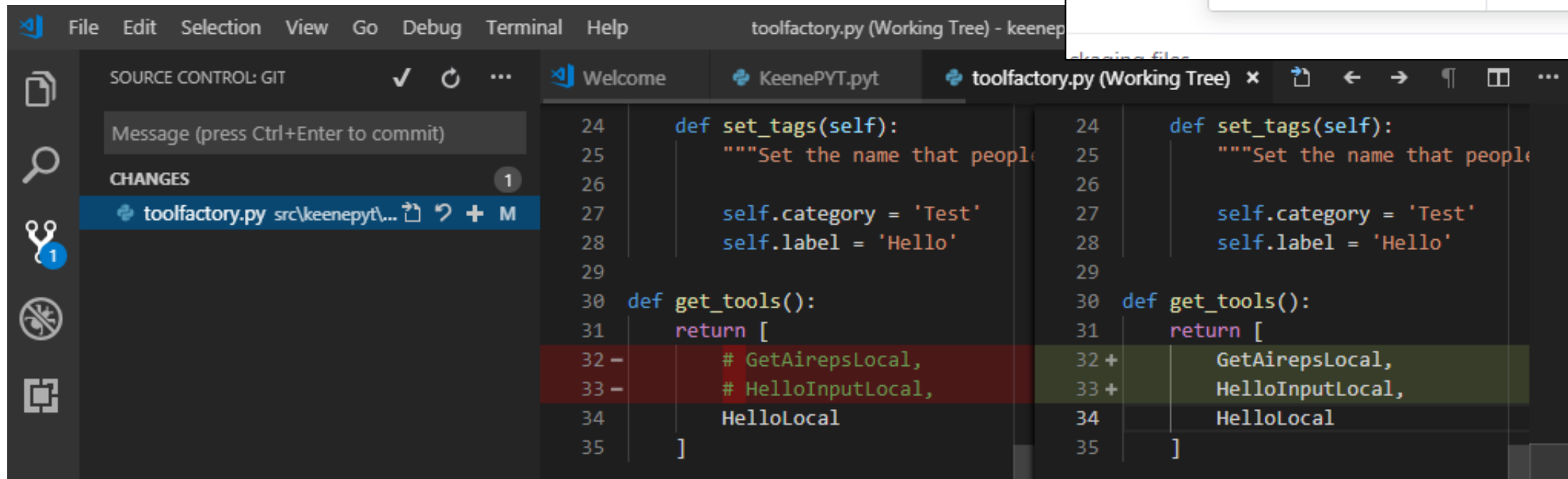
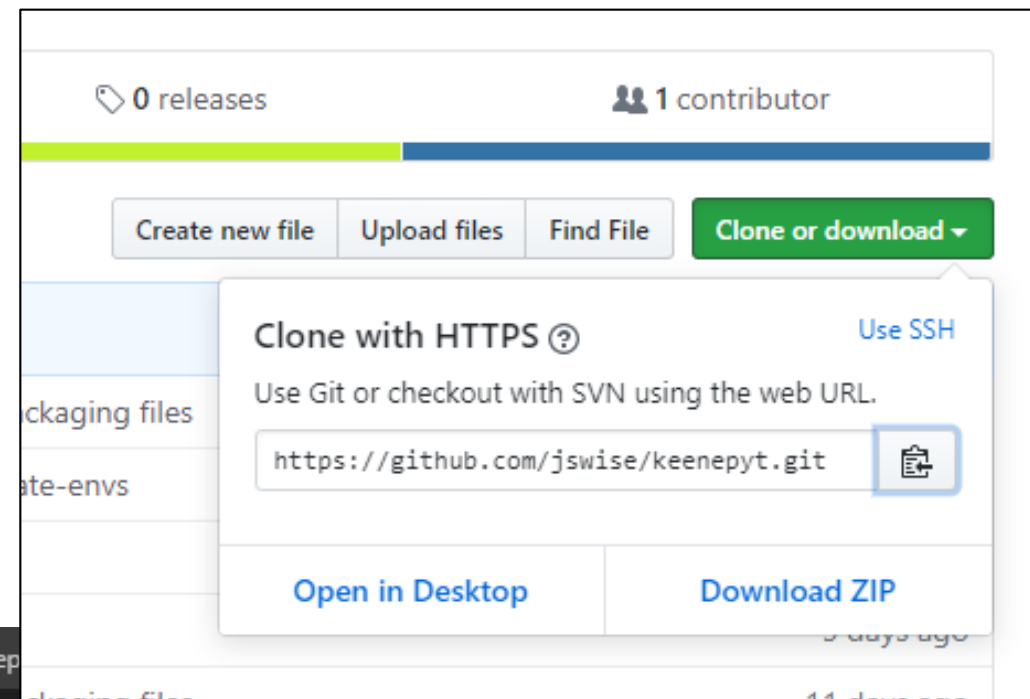
KEENEPTYT
.pytest_cache
.vscode
batch
build.bat
config.bat
copy-from-flash.bat
create-dev-envs.bat
create-main-env.bat
deploy.bat
install-conda.bat
unzip.bat
data
recipe
src
  keenepyt
    __pycache__
    core
    tools
    getaireps.py
    hello.py
    helloinput.py
    toolfactory.py
    __init__.py
    avwx.py
  tests
  citationero
```

```
1 """Define the GetAireps class."""
2
3 import arcpy
4
5 from keenepyt.core.pyttool import PYTTool
6 from keenepyt.avwx import AvWx
7
8 class GetAireps(PYTTool):
9     """Gets aircraft weather reports from the National Weather Service."""
10
11     def init_parameter_info(self):
12         """Set the parameters for the tool."""
13
14         self.add_param(
15             'gdb',
16             'Workspace',
17             datatype='DEWorkspace',
18             defaultValue=arcpy.env.workspace # pylint: disable=no-member
19         )
20         self.add_param(
21             'fc_name',
22             'Feature Class Name',
23             datatype='GPString',
24             defaultValue='Aireps'
25         )
26         self.add_param(
27             'fc',
28             'Aireps Feature Class',
29             datatype='DEFeatureClass',
30             direction='Output',
31             parameterType='Derived'
32         )
33
34     def execute(self, parameters, messages):
35         """Get aireps."""
36
37         params = self.get_param_val_dict(parameters)
38         gdb = params['gdb']
39         fc_name = params['fc_name']
40         aviation_weather = AvWx(self.helpers)
41         fc = aviation_weather.write_aireps_fc(gdb, fc_name)
42         self.set_tool_param_value(parameters, 'fc', fc)
```



# More Python fun

Use a Git repository.



Jason Wise

jason.wise@Terracon.com  
<https://github.com/jswise>

**Terracon**

**RESPONSIVE.  
RESOURCEFUL.  
RELIABLE.**

Environmental



Facilities



Geotechnical



Materials