WWU MÜNSTER

Python in GIS

Processing: QGIS 1

living.knowledge

ifgi
Institut für Geoinformatik
Universität Münster

Learning goals

After this lesson you should be able to use PyQGIS to:

- Create a `QgsVectorLayer()` object from a file on disk and from an active layer
- Access features and their attribute values
- Add and remove features from a vector file
- Add, fill and remove fields in an attribute table
- Know how you can create a plugin in QGIS using Python and PyQGIS

Acknowledgements:

- The PyQGIS Developer Cookbook: https://docs.qgis.org/3.4/en/docs/pyqgis_developer_cookbook/index.html
- Ujaval Gandhi, Building a Python Plugin, https://www.qgistutorials.com/en/docs/3/building_a_python_plugin.html

Applications with PyQGIS

Three possible usages of PyQGIS:

1. Python console/scripts in QGIS: now
2. Standalone Python applications based on QGIS
3. QGIS Python Plugins: later this afternoon

Never use qgis.py as a name for your test script — Python will not be able to import the bindings as the script's name will 'shadow' them.

Sub-libraries in PyQGIS

Main ones:

- core: the main module
- analysis: tools for spatial analysis on vector and raster data
- utils: module with utilities, e.g. for using existing plugins
- gui: GUI components (mainly map canvas) with support for zooming, panning and/or any further custom map tools
- server: adds map server components to QGIS
- 3D: 3D features

In QGIS: Loading and removing layers

Loading layers (1)

qgis.utils.iface provides QgisInterface

```
iface.addVectorLayer(URI, LAYER NAME, TYPE)
```

With:

`URI` - Uniform Resource Identifier = the path to the existing file plus additional info

`LAYER NAME` - the name you want to give to the layer in the interface

`TYPE` – a string representing the file type

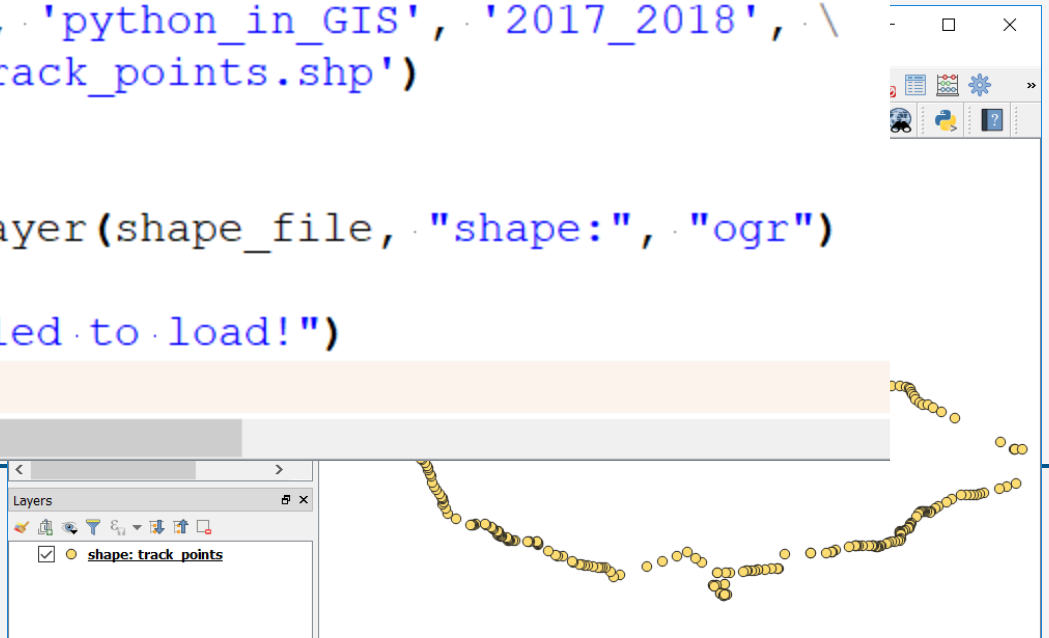
Loading layers

Simplest: loading a shape file to the canvas

QGIS1_loading.py

```python
import os
from qgis.core import *
import qgis.utils

# path to shapefile
shape_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'shapefiles', 'track_points.shp')

# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")
```

we use two sublibraries from pyqgis



Loading layers, URI and Type

File type	URI should contain besides path	String to use as file type
shapefile	-	‘ogr’
dxf (CAD format)	Layername, geometrytype	‘ogr’
PostGIS database	Connection (port etc), data source	‘postgres’
csv	Field names x and y, delimiter	‘delimitedtext’
GPX	Type (track / route / waypoint)	‘GPX’
SpatiaLite database	Schema, table, geometry column	‘spatialite’
MySQL WKB-based geometries	DBname, host, user, password etc.	‘ogr’
WFS connection	Service, request, SRS etc.	‘WFS’

See: https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/loadlayer.html

Loading layers, example 2

`addVectorLayer()` creates a `QgsVectorLayer()` object, with features in it:

QGIS1_loading.py

```python
import os
from qgis.core import *
import qgis.utils

# path to shapefile
shape_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'workshops_conferences', '2019_2020', \
'ILS_Python', 'materials', 'data', 'gps_track_projected.shp')

# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")
else:
    print('layer', layer)
    for feat in layer.getFeatures():
        print('feature', feat)
```

```
layer <qgis._core.QgsVectorLayer object at 0x000002C784E73A68>
feature <qgis._core.QgsFeature object at 0x000002C784E8E828>
```

getFeatures() function

Removing layers

You can remove layers from the canvas by calling:

```
QgsProject.instance().removeMapLayer(ID)
```

with:

`ID` - The identifier of the map, can be obtained from the layer with `QgsVectorLayer().id()`

Note that in the QGIS 2, this function was in QgsMapLayerRegistry instead of QgsProject.

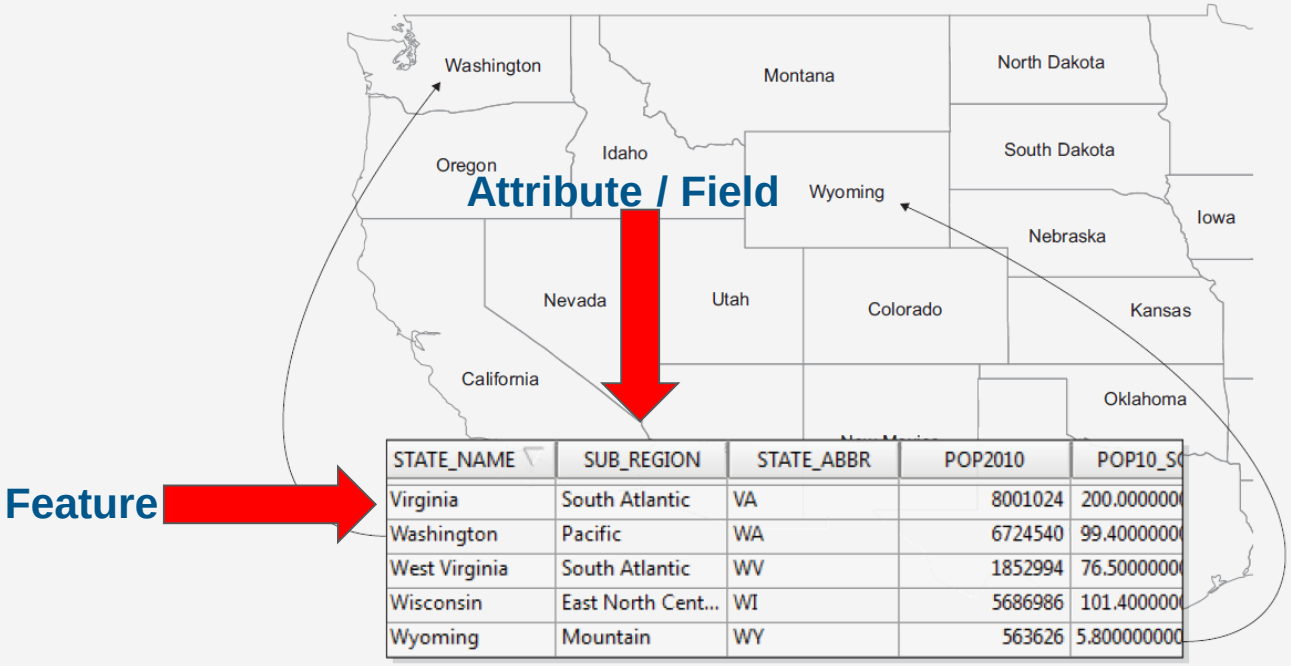
Removing layers

Handy when testing a script, and you do not want the layer to be loaded 100 times!

```
# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")
else:
    print('layer', layer)
    for feat in layer.getFeatures():
        print('feature', feat)

QgsProject.instance().removeMapLayer(layer.id())
```

remove layer

In QGIS: accessing features



STATE_NAME	SUB_REGION	STATE_ABBR	POP2010	POP10_S
Virginia	South Atlantic	VA	8001024	200.000000
Washington	Pacific	WA	6724540	99.4000000
West Virginia	South Atlantic	WV	1852994	76.5000000
Wisconsin	East North Cent...	WI	5686986	101.400000
Wyoming	Mountain	WY	563626	5.800000000

Fields (1)

The `QgsVectorLayer()` class (and thus object) has a `fields()` function

QGIS2_fields_attributes.py

```python
import os
from qgis.core import *
import qgis.utils

# path to shapefile
shape_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'shapefiles', 'track_points.shp')

# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")

for field in layer.fields():
    print(field.name(), field.typeName())

QgsProject.instance().removeMapLayer(layer.id())
```

layer.fields() NOTE: not getFields()

Fields (2)

Python Console

```
2 Use iface to access QGIS
  API interface or Type hel
  p(iface) for more info
3 >>> exec(open('C:/Users/v
  erstege/Documents/educati
  on/python_in_GIS/2017_201
  8/scripts/6_QGIS/QGIS2_fi
  elds_attributes.py'.encod
  e('utf-8')).read())
4 track_fid Integer
5 track_seg_ Integer
6 track_se_1 Integer
7 ele Real
8 time Date
9 magvar Real
10 geoidheigh Real
11 name String
12 cmt String
13 desc String
14 src String
```

```
>>>
```

Iterating over features and attributes

QGIS2_fields_attributes.py

```python
if not layer:
    print("Shapefile failed to load!")

# Show field names of this shapefile
for field in layer.fields():
    print(field.name(), field.typeName())

# Loop over all features
features = layer.getFeatures()
for feature in features:
    # retrieve id and attribute values
    print("Feature ID: %d " % feature.id())
    attributes = feature.attributes()
    # attributes is a list
    for field, attr in zip(layer.fields(), attributes):
        print(field.name(), ':', attr)
```

remember from first script

features have an id and attributes

```
pdop : NULL
ageofdgpsd : NULL
dgpsid : NULL
Feature ID: 6
track_fid : 0
track_seg_ : 0
track_se_1 : 6
ele : 85.0
```

Intermezzo: zip() (1)

zip() makes an iterator that aggregates elements from each of the inputs (iterables).

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]
```

Intermezzo: zip() (1)

Especially useful in loops.

Use only when you know that the inputs have equal lengths!

```
alist = ['a1', 'a2', 'a3']
blist = ['b1', 'b2', 'b3']

for a, b in zip(alist, blist):
    print a, b
```

Results:

```
a1 b1
a2 b2
a3 b3
```

A subset of features

QGIS2_fields_attributes.py

```python
# Feature selection with expression
exp = QgsExpression('ele > 100')
request = QgsFeatureRequest(exp)
selection = layer.getFeatures(request)

# Check elevations
for feat in selection:
    print(feat['ele'])

QgsProject.instance().removeMapLayer(layer.id())
```

Python Console

```
12482 138.0
12483 138.0
12484 137.0
12485 114.0
12486 111.0
12487 102.0
12488 104.0
12489 104.0
12490 102.0
12491 102.0
12492 102.0
12493
>>>
```

Exercise #1

In the data folder there is a shapefile gps_track_projected.shp

Write a script, using PyQGIS, to:

- load this file to the QGIS interface
- request a subset of only the first 10 feature
- loop over these features
- print their id and elevation

In QGIS: Adding and removing features

Editing rights

Most vector data support editing of layer data.

Sometimes (e.g. a WFS) they support just a subset of possible editing actions.

Use the `capabilities()` function to find out what set of functionality is supported.

To print layer's capabilities textual description in a comma separated list you can use `capabilitiesString()` as in the following example.

QGIS4_add_remove_feature.py

```python
import os
from qgis.core import *
import qgis.utils

# path to shapefile
shape_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'shapefiles', 'track_points.shp')

# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")

# Check for editing rights (capabilities)
caps = layer.dataProvider().capabilities()
print(caps)
caps_string = layer.dataProvider().capabilitiesString()
print(caps_string)
```

```
5 Add Features, Delete Features, Change A
  ttribute Values, Add Attributes, Delete
   Attributes, Rename Attributes, Create
  Spatial Index, Create Attribute Indexes
  , Fast Access to Features at ID, Change
   Geometries
6
```

Commit changes

In the following examples, the vector layer changes are directly committed to the underlying data store (a file, database etc).

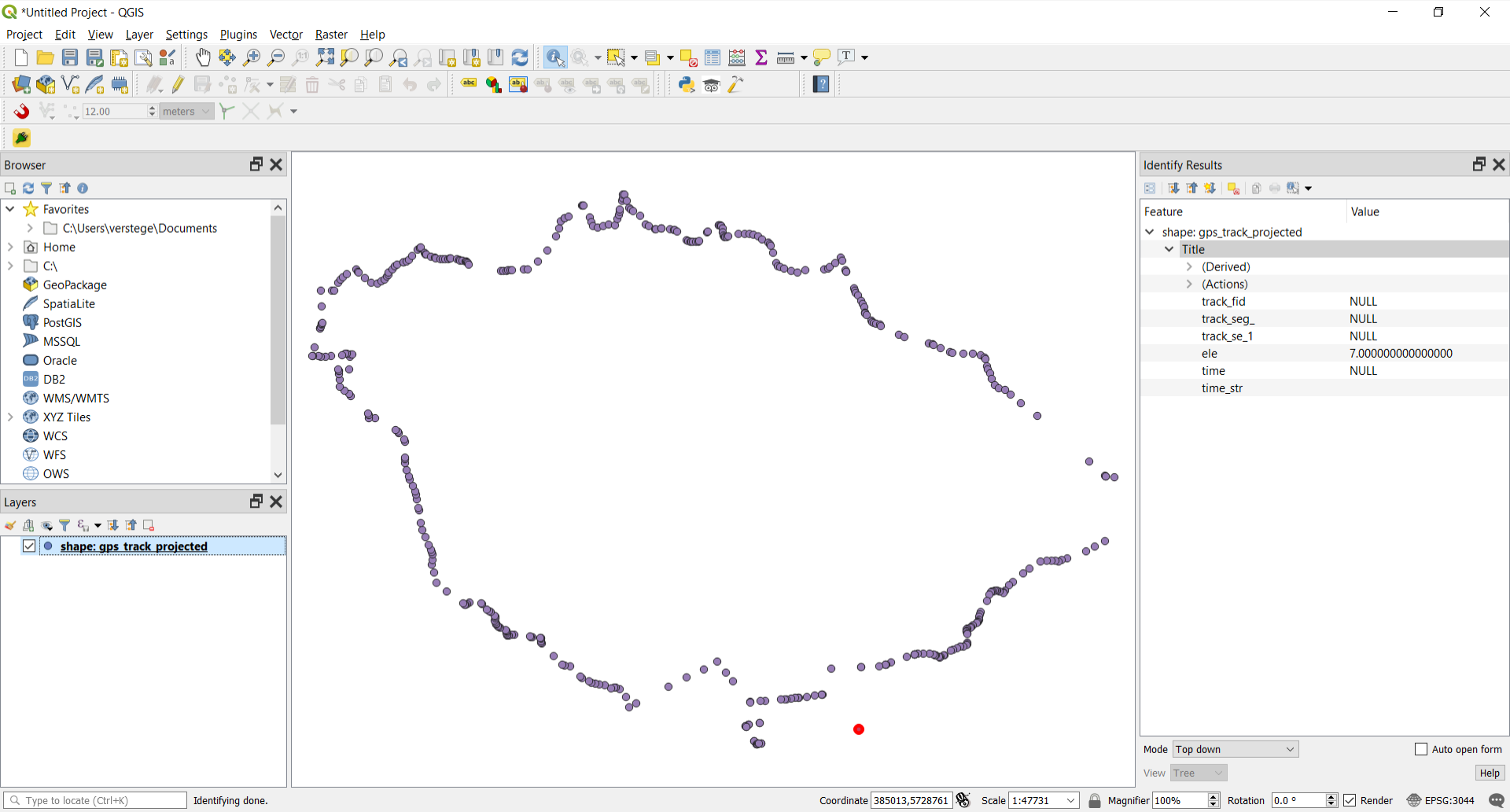
In case you would like to do only temporary changes, you can do modifications with editing buffer, see: https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/vector.html#editing-buffer.

Adding a feature

QGIS4_add_remove_feature.py

```
caps = layer.dataProvider().capabilities()
print(caps)
caps_string = layer.dataProvider().capabilitiesString()
print(caps_string)

# Adding a feature
if caps & QgsVectorDataProvider.AddFeatures:
    feat = QgsFeature(layer.fields())
    # Set a single attribute by key or by index:
    feat.setAttribute('ele', 7)
    feat.setAttribute(3, 7)
    # Set the FID

    feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(381770,5727771)))
    (res, outFeats) = layer.dataProvider().addFeatures([feat])
```

Note AddFeatures


*Untitled Project - QGIS
Project Edit View Layer Settings Plugins Vector Raster Help
12.00
meters
Browser
Favorites
C:\Users\verstege\Documents
Home
C:\
GeoPackage
SpatiaLite
PostGIS
MSSQL
Oracle
DB2
WMS/WMTS
XYZ Tiles
WCS
WFS
OWS
Layers
shape: gps_track_projected
Identify Results
Feature
Value
shape: gps_track_projected
Title
(Derived)
(Actions)
track_fid NULL
track_seg_ NULL
track_se_1 NULL
ele 7.000000000000000
time NULL
time_str
Mode Top down
Auto open form
View Tree
Help
Type to locate (Ctrl+K)
Identifying done.
Coordinate 385013,5728761
Scale 1:47731
Magnifier 100%
Rotation 0.0 °
Render
EPSG:3044

Removing a feature

Removing should be done based on the index

QGIS4_add_remove_feature.py

```
# Deleting the feature
if caps & QgsVectorDataProvider.DeleteFeatures:
    res = layer.dataProvider().deleteFeatures([428])

QgsProject.instance().removeMapLayer(layer.id())
```

Note DeleteFeatures

In QGIS: Adding and removing fields or attribute values

Adding a field (1)

QGIS5_add_remove_field.py

Same beginning as previous script

```python
import os
from qgis.core import *
import qgis.utils

# path to shapefile
shape_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'shapefiles', 'track_points.shp')

# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")

# Check for editing rights (capabilities)
caps = layer.dataProvider().capabilities()
print(caps)
caps_string = layer.dataProvider().capabilitiesString()
print(caps_string)
```

Adding a field (2)

Use the `addAttributes()` function of the layer's `dataProvider` class

- Set the name of the field
- Set the field type using `QVariant()` object

QGIS5_add_remove_field.py

```
print(caps_string)

# Adding a field (attribute)
if caps & QgsVectorDataProvider.AddAttributes:
    # Adding as a list of items, each with
    # the name and the data type of the field
    res = layer.dataProvider().addAttributes(
        [QgsField("mytext", QVariant.String),
        QgsField("myint", QVariant.Int)])

# update to propagate the changes
layer.updateFields()
```

Note AddAttributes

Adding a field (3)

Other data types available in `QVariant()` (selection):

- Bool
- Date
- DateTime
- Double
- Int
- String
- Time

Adding a field (4)

Result

shape: gps_track_projected :: Features Total: 428, Filtered: 428, Selected: 0

	_	track_se_1	ele	time	time_str	mytext	myint
1	0	412	66.0000000000...	2010-01-01	03:02:31		NULL
2	0	413	70.0000000000...	2010-01-01	03:02:53		NULL
3	0	402	55.0000000000...	2010-01-01	02:55:38		NULL
4	0	403	63.0000000000...	2010-01-01	02:57:57		NULL
5	0	400	50.0000000000...	2010-01-01	02:54:15		NULL
6	0	401	55.0000000000...	2010-01-01	02:55:38		NULL
7	0	406	69.0000000000...	2010-01-01	03:00:06		NULL
8	0	407	70.0000000000...	2010-01-01	03:00:39		NULL
9	0	404	69.0000000000...	2010-01-01	03:00:03		NULL
10	0	405	69.0000000000...	2010-01-01	03:00:05		NULL

Show All Features

Removing a field (1)

Using the same `dataProvider` class, you can remove a field, only by index!

QGIS5_add_remove_field.py

```
31
32  # Now remove the fields again because we do not need them
33  # We need the field index
34  # Unfortunately backward counting (-1 and -2) does not work
35  if caps & QgsVectorDataProvider.DeleteAttributes:
36      res = layer.dataProvider().deleteAttributes([26, 27])
37
38  # update to propagate the changes
39  layer.updateFields()
40
41  ##QgsProject.instance().removeMapLayer(layer.id())
42
```

Note DeleteAttributes

Removing a field (2)

Result

shape: gps_track_projected :: Features Total: 428, Filtered: 428, Selected: 0

	track_fid	track_seg_	track_se_1	ele	time	time_str
1	0	0	0	81.0000000000...	2010-01-01	00:00:00
2	0	0	1	81.0000000000...	2010-01-01	00:00:00
3	0	0	2	83.0000000000...	2010-01-01	00:00:12
4	0	0	3	83.0000000000...	2010-01-01	00:00:16
5	0	0	4	83.0000000000...	2010-01-01	00:00:20
6	0	0	5	83.0000000000...	2010-01-01	00:00:39
7	0	0	6	85.0000000000...	2010-01-01	00:01:53
8	0	0	7	85.0000000000...	2010-01-01	00:01:59
9	0	0	8	86.0000000000...	2010-01-01	00:02:15
10	0	0	9	80.0000000000...	2010-01-01	00:03:03

Show All Features

Removing a field (3)

If you do not know the index of your field, search for it by name.

You can use something like:

```
field_name_i_search = “mytext”
fields = layer.dataProvider().fields()
index = 0
for field in fields:
    if field.name() == field_name_i_search:
        break
    index += 1
```

Editing attr. values of existing features (1)

QGIS6_edit_attr_values.py

Same start of the script as before

```python
import os
from qgis.core import *
import qgis.utils

# path to shapefile
shape_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'shapefiles', 'track_points.shp')

# load the shapefile
layer = iface.addVectorLayer(shape_file, "shape:", "ogr")
if not layer:
    print("Shapefile failed to load!")

# Check for editing rights (capabilities)
caps = layer.dataProvider().capabilities()
print(caps)
caps_string = layer.dataProvider().capabilitiesString()
print(caps_string)
```

Editing attr. values of existing features (2)

Need the `dataProvider` class again, and two dictionaries:

- one with dict1 = {field_index : new_value}
- and one with dict2 = {feature_id : dict1}

QGIS6_edit_attr_values.py

```
caps_string = layer.dataProvider().capabilitiesString()
print(caps_string)

# Changing a single attribute value
fid = 10   # INDEX of the feature we will modify
if caps & QgsVectorDataProvider.ChangeAttributeValues:
    # field index : new value
    attrs = {1:12}
    # Note the dictionary of dictionaries!
    layer.dataProvider().changeAttributeValues({fid:attrs})
```

Note ChangeAttr

Editing attr. values of existing features (3)

Result

shape: gps_track_projected :: Features Total: 428, Filtered: 428, Selected: 0

	track_fid	track_seg_	track_se_1	ele	time	time_str
7	0	0	6	85.0000000000...	2010-01-01	00:01:53
8	0	0	7	85.0000000000...	2010-01-01	00:01:59
9	0	0	8	86.0000000000...	2010-01-01	00:02:15
10	0	0	9	80.0000000000...	2010-01-01	00:03:03
11	0	12	10	79.0000000000...	2010-01-01	00:04:48
12	0	0	11	69.0000000000...	2010-01-01	00:06:53
13	0	0	12	68.0000000000...	2010-01-01	00:07:45
14	0	0	13	66.0000000000...	2010-01-01	00:08:52
15	0	0	14	66.0000000000...	2010-01-01	00:10:03
16	0	0	15	62.0000000000...	2010-01-01	00:11:30

Show All Features

Editing attr. values of existing features (4)

And change it back, just in case

QGIS6_edit_attr_values.py

```
    layer.dataProvider().changeAttributeValues({fid:attrs})

# Change it back
if caps & QgsVectorDataProvider.ChangeAttributeValues:
    attrs = {1:0}
    # Note the dictionary of dictionaries!
    layer.dataProvider().changeAttributeValues({fid:attrs})

QgsProject.instance().removeMapLayer(layer.id())
```

Exercise #2

Use again the shapefile gps_track_projected.shp

Write a script, using PyQGIS, to:

- add a field elediff
- update each feature, such that this field contains the elevation difference between this feature and the previous feature

shape: gps_track_projected :: Features Total: 428, Filtered: 428, Selected: 0

	track_fid	track_seg_	track_se_1	ele	
7	0	0	6	85.0000000000...	0
8	0	0	7	85.0000000000...	0
9	0	0	8	86.0000000000...	1
10	0	0	9	80.0000000000...	-6
11	0	12	10	79.0000000000...	
12	0	0	11	69.0000000000...	
13	0	0	12	68.0000000000...	

QGIS plugins

QGIS plugins

Plugins in QGIS add functionality to the software.

Plugins are written by QGIS developers or independent users who want to extend the core functions of the software. Plugins can be made available in QGIS for all users.

QGIS cookbook:

“It is possible to create plugins in Python programming language. In comparison with classical plugins written in C++ these should be easier to write, understand, maintain and distribute due the dynamic nature of the Python language.”

File location of plugins

Python plugins are listed together with C++ plugins in QGIS plugin manager. QGIS searches for plugins in these paths:

~\AppData\Roaming\QGIS\QGIS3\python\plugins

and

C:\Program Files\QGIS 3.0\apps\qgis\python\plugins

Home directory (denoted by above ~) on Windows is usually something like C:\Documents and Settings\(user) (on Windows XP or earlier) or C:\Users\(user).

By setting QGIS_PLUGINPATH to an existing directory path, you can add this path to the list of paths that are searched for plugins.

Basic steps to create your own plugin

1. Idea: What you want to do with your new QGIS plugin? Is there already another plugin for that problem?
2. Create files: Create the files described next.
    a. A starting point (__init__.py).
    b. The Plugin metadata (metadata.txt)
    c. A form in QT-Designer (form.ui), with its resources.qrc.
    d. A main python plugin body (mainplugin.py).
3. Write code: Write the code inside the mainplugin.py
4. Test: Close and re-open QGIS and import your plugin again (or Reloader). Check it.
5. (Publish: Publish your plugin in QGIS repository or make your own repository as an “arsenal” of personal “GIS weapons”.)

What is our idea

Remove all fields from a vector file that are empty or contain only zeroes

Files that form the plugin (1)

Directory structure of an example plugin from the cookbook:

```
PYTHON_PLUGINS_PATH/
  MyPlugin/
    __init__.py     --> *required*
    mainPlugin.py   --> *required*
    metadata.txt    --> *required*
    resources.qrc   --> *likely useful*
    resources.py    --> *compiled version, likely useful*
    form.ui         --> *likely useful*
    form.py         --> *compiled version, likely useful*
```

+ several more to be allowed to enter it into the official Plugin Repository!

Files that form the plugin (2)

What is the meaning of the files:

- `__init__.py` = The starting point of the plugin. It has to have the `classFactory()` method and may have any other initialization code.
- `mainPlugin.py` = The main working code of the plugin. Contains all the information about the actions of the plugin and the main code.
- `resources.qrc` = The .xml document created by Qt Designer. Contains relative paths to resources of the forms.
- `resources.py` = The translation of the .qrc file described above to Python.
- `form.ui` = The GUI created by Qt Designer.
- `form.py` = The translation of the `form.ui` described above to Python.
- `metadata.txt` = Contains general info, version, name and some other metadata used by plugins website and plugin infrastructure.

What software do we need?

1. QGIS Plugin Builder Plugin (creates all input files automatically!)
2. Qt Creator/Designer (GUI) → comes with QGIS
3. PyUIC5 → .ui to .py (command line tool)

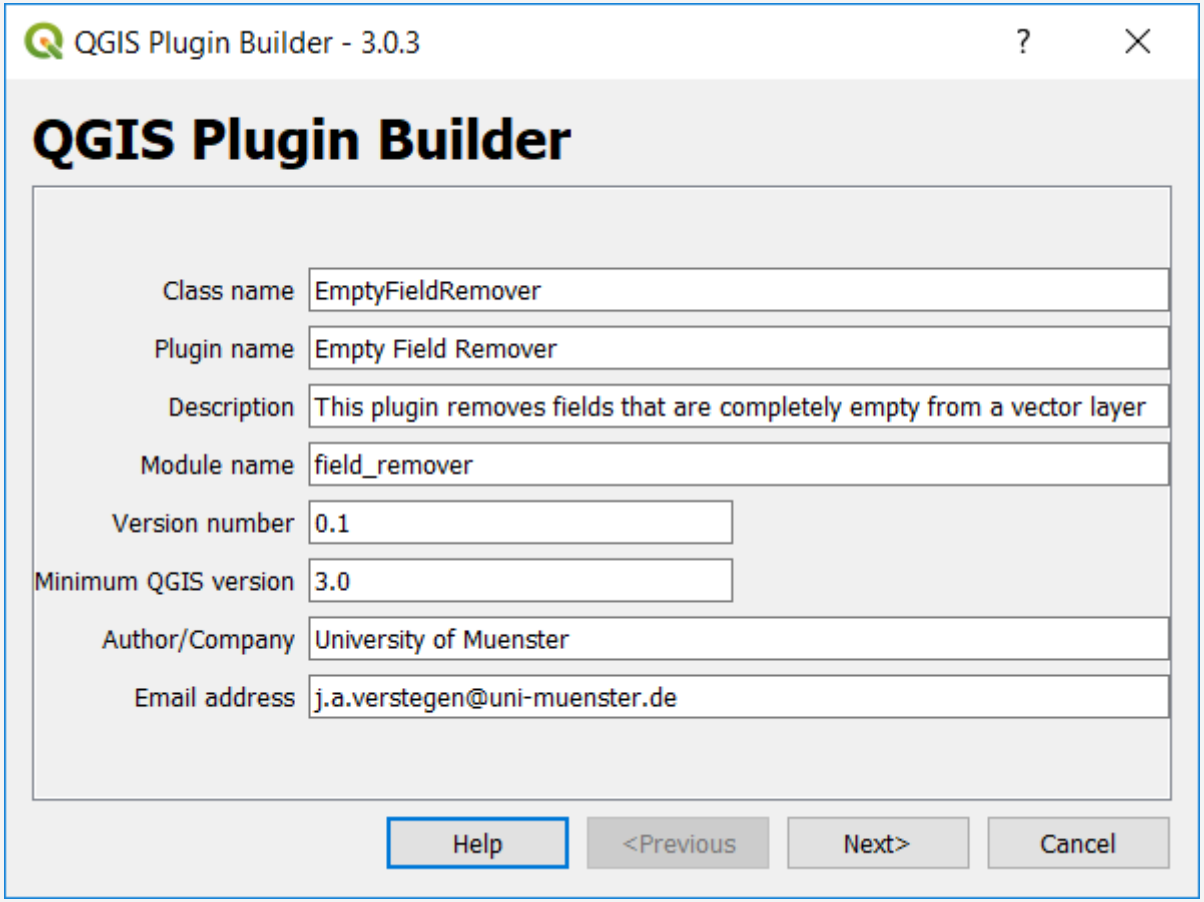
   ```
   pyuic5 [.ui file] > [.py file]
   ```

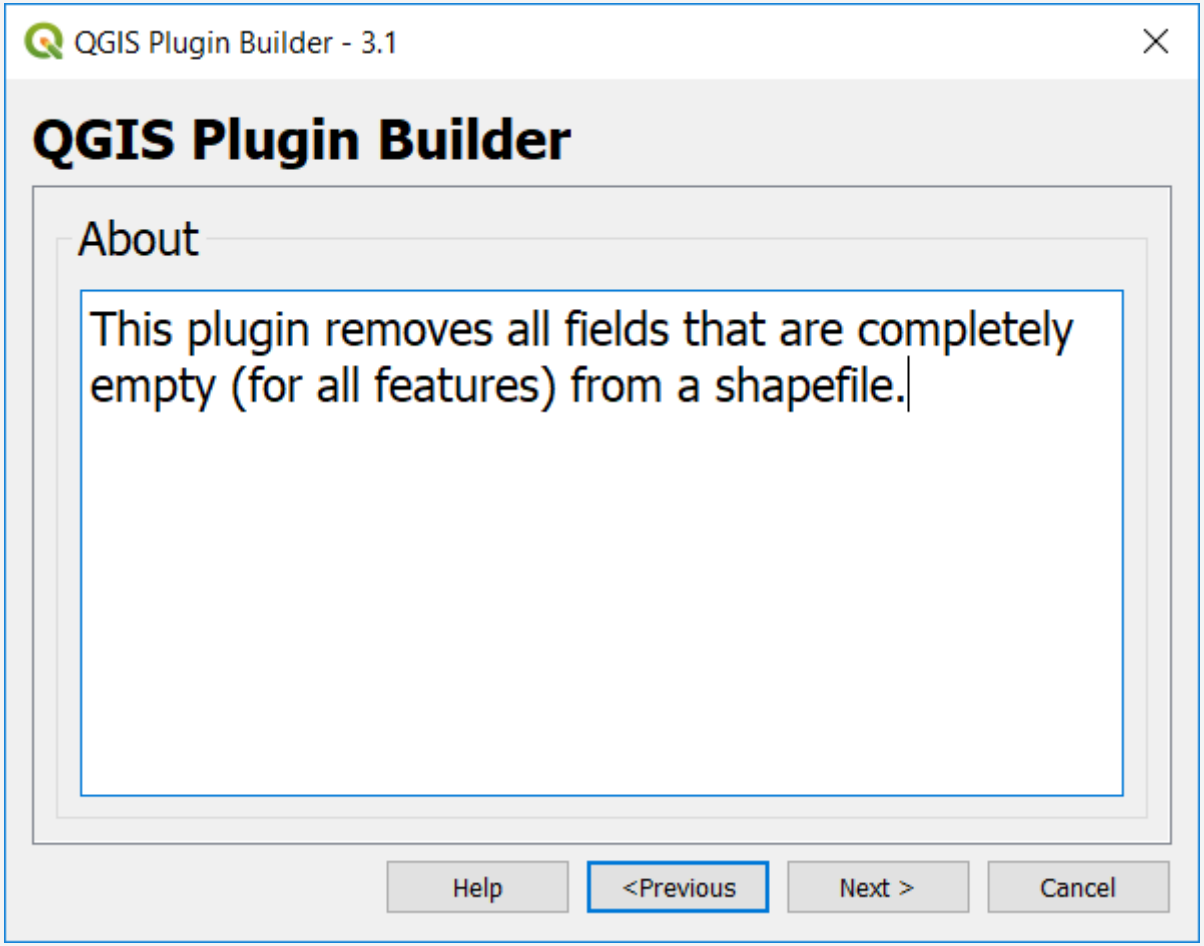
4. PyRCC5 → .qrc to .py (command line tool)

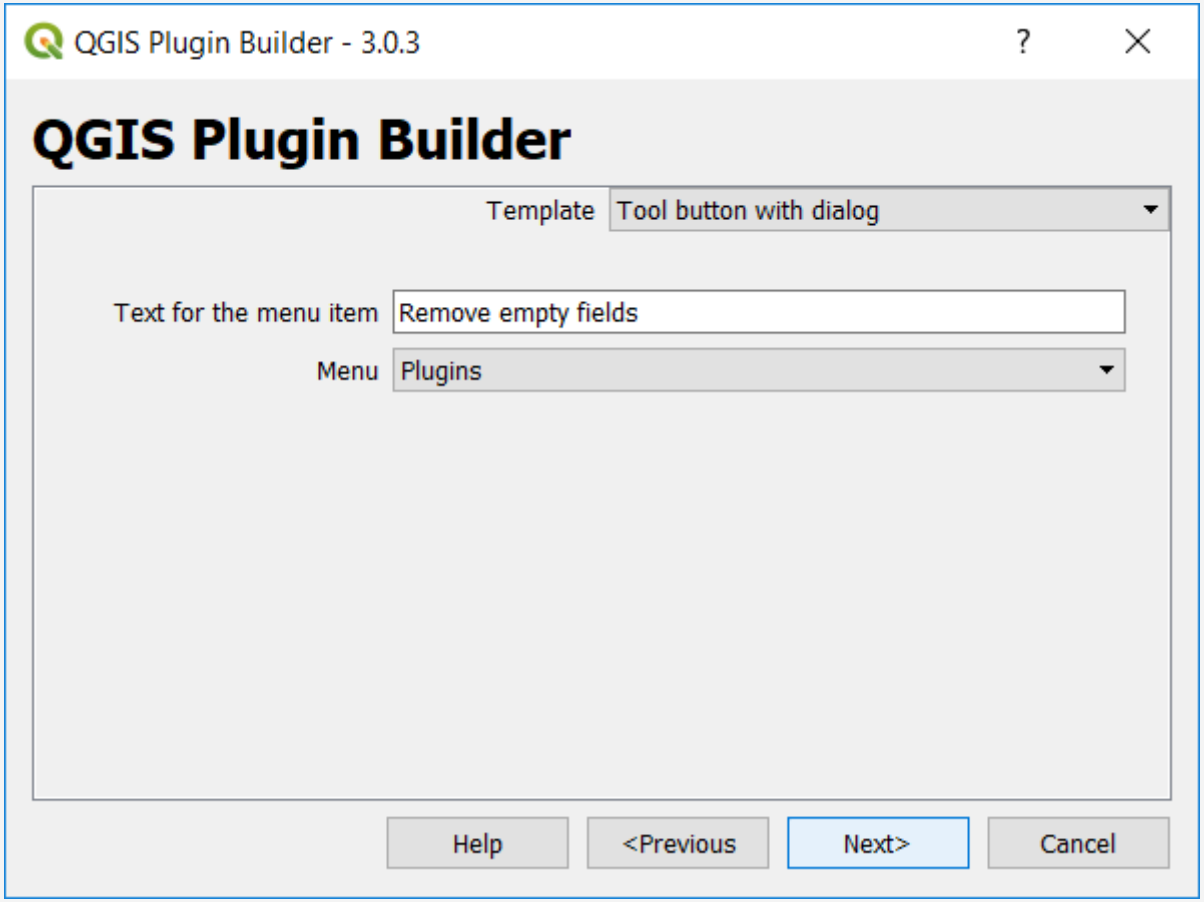
   ```
   pyrcc5 -o [.qrc file] [.py file]
   ```

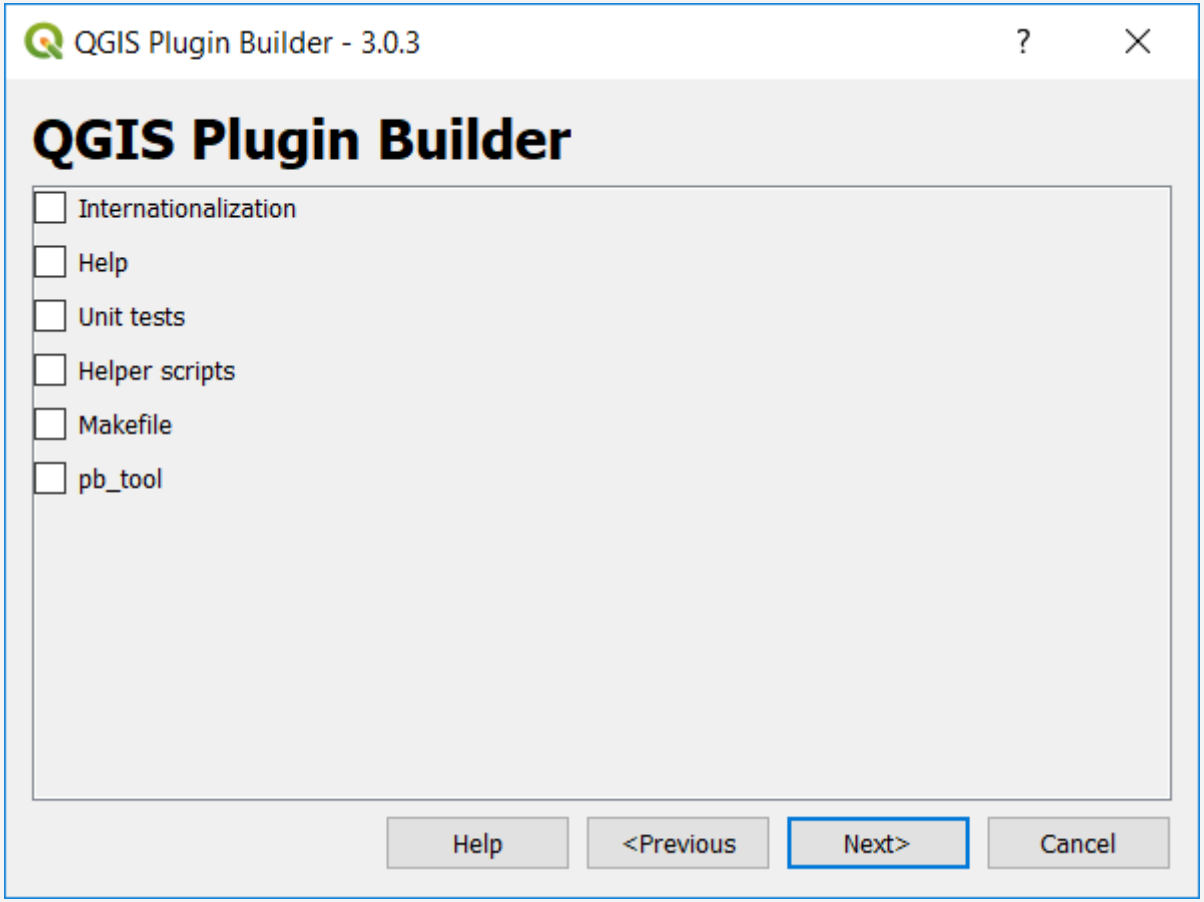
5. (QGIS plugin reloader plugin)
6. Editor for main Python script

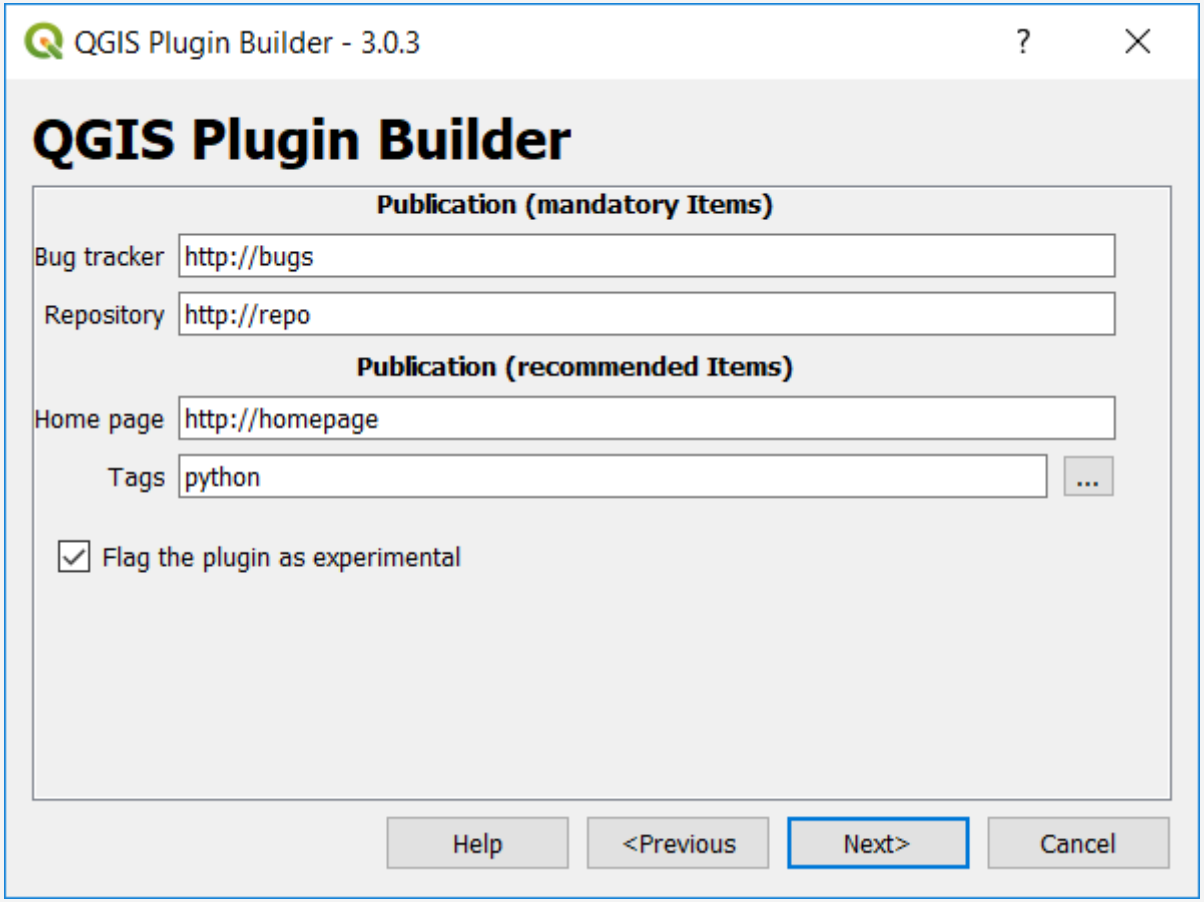
1. Plugin Builder


QGIS Plugin Builder - 3.0.3
QGIS Plugin Builder
Class name
EmptyFieldRemover
Plugin name
Empty Field Remover
Description
This plugin removes fields that are completely empty from a vector layer
Module name
field_remover
Version number
0.1
Minimum QGIS version
3.0
Author/Company
University of Muenster
Email address
j.a.verstegen@uni-muenster.de
Help
<Previous
Next>
Cancel

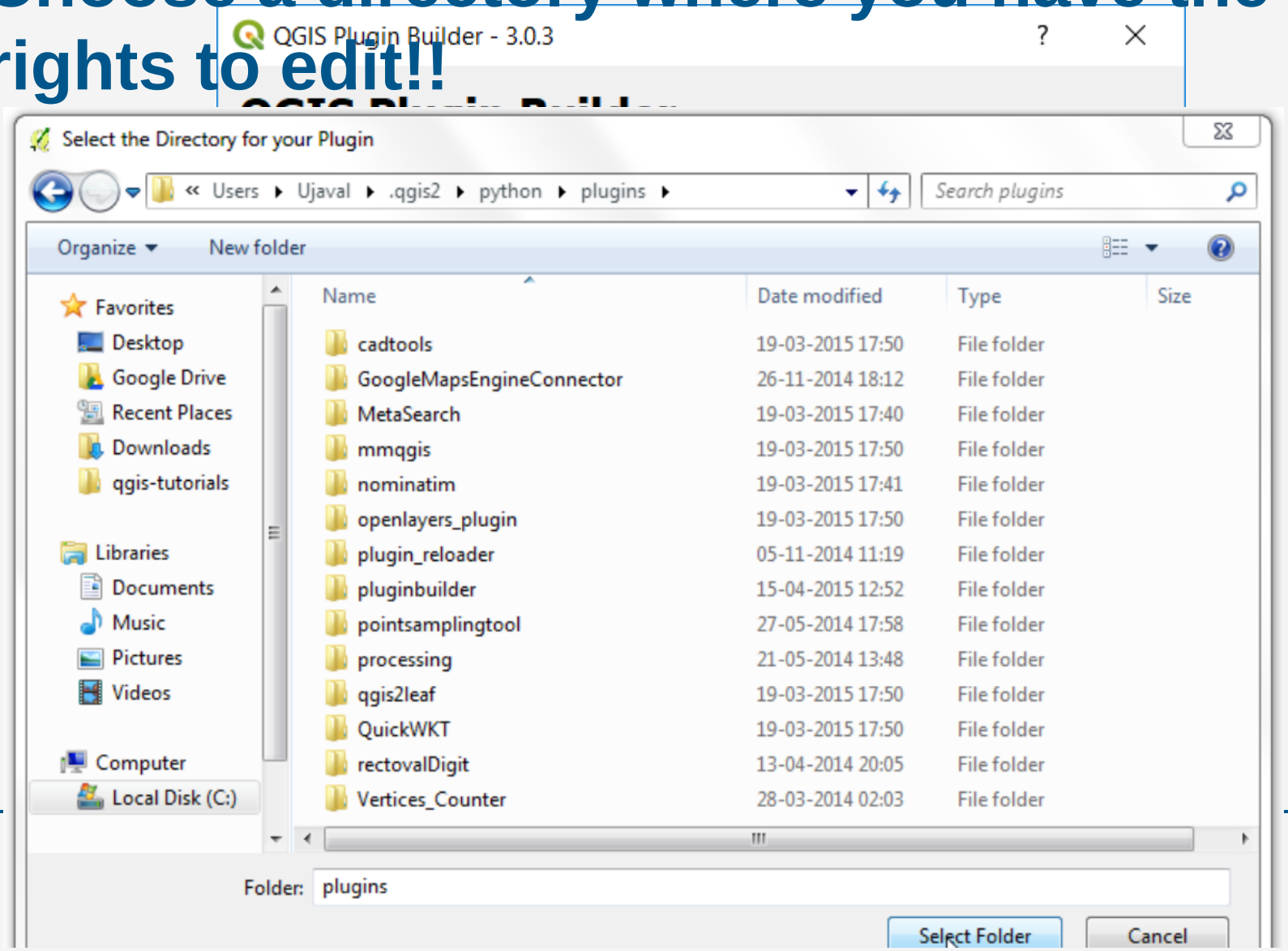

QGIS Plugin Builder - 3.1
QGIS Plugin Builder
About
This plugin removes all fields that are completely empty (for all features) from a shapefile.
Help
<Previous
Next >
Cancel


QGIS Plugin Builder - 3.0.3
QGIS Plugin Builder
Template
Tool button with dialog
Text for the menu item
Remove empty fields
Menu
Plugins
Help
<Previous
Next>
Cancel

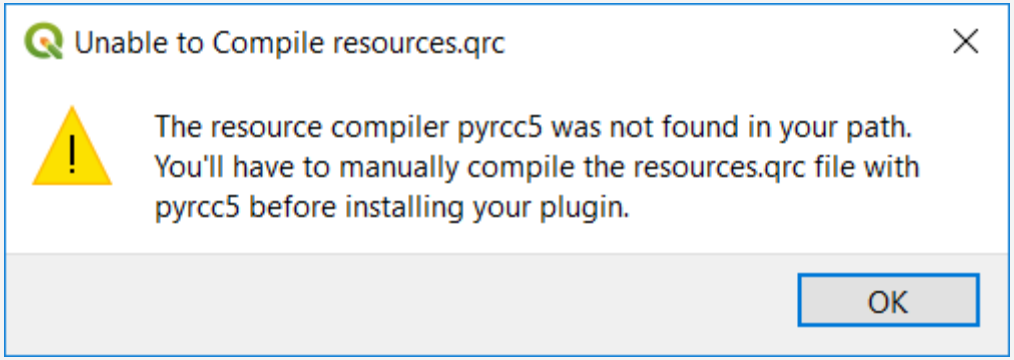

QGIS Plugin Builder - 3.0.3
QGIS Plugin Builder
Internationalization
Help
Unit tests
Helper scripts
Makefile
pb_tool
Help
<Previous
Next>
Cancel


QGIS Plugin Builder - 3.0.3
QGIS Plugin Builder
Publication (mandatory Items)
Bug tracker
http://bugs
Repository
http://repo
Publication (recommended Items)
Home page
http://homepage
Tags
python
...
Flag the plugin as experimental
Help
<Previous
Next>
Cancel

Choose a directory where you have the rights to edit!!



You may get this message



Plugin Builder Results

Plugin Builder Results

Congratulations! You just built a plugin for QGIS!

Your plugin **EmptyFieldRemover** was created in:
C:/Users/verstege/Documents/education/python_in_GIS/2018_2019/scripts/6_emptyfieldremover\field_remover

Your QGIS plugin directory is located at:
C:/Users/verstege/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins

What's Next

1. If resources.py is not present in your plugin directory, compile the resources file using pyrcc5 (simply use **pb_tool** or **make** if you have automake)
2. Optionally, test the generated sources using **make test** (or run tests from your IDE)
3. Copy the entire directory containing your new plugin to the QGIS plugin directory (see Notes below)
4. Test the plugin by enabling it in the QGIS plugin manager
5. Customize it by editing the implementation file **field_remover.py**
6. Create your own custom icon, replacing the default **icon.png**
7. Modify your user interface by opening **field_remover_dialog_base.ui** in Qt Designer

Notes:

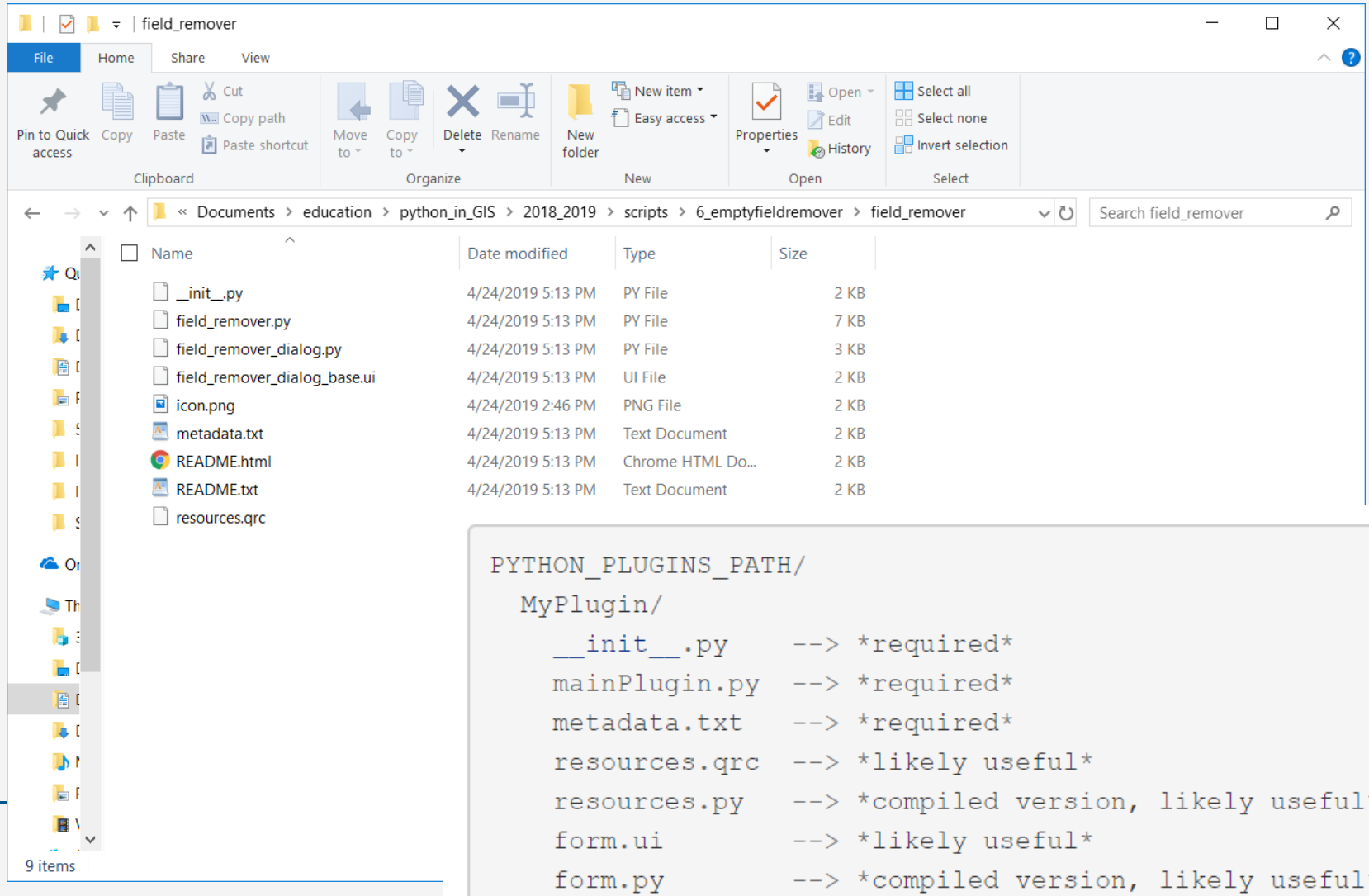
- You can use **pb_tool** to compile, deploy, and manage your plugin. Tweak the *pb_tool.cfg* file included with your plugin as you add files. Install **pb_tool** using *pip* or *easy_install*. See **http://loc8.cc/pb_tool** for more information.
- You can also use the **Makefile** to compile and deploy when you make changes. This requires GNU make (gmake). The Makefile is ready to use, however you will have to edit it to add addional Python source files, dialogs, and translations.

For information on writing PyQGIS code, see **http://loc8.cc/pyqgis_resources** for a list of resources.

OK

Copy this!!

Notice we now have <u>some</u> of the files




```
PYTHON_PLUGINS_PATH/
  MyPlugin/
    __init__.py     --> *required*
    mainPlugin.py   --> *required*
    metadata.txt    --> *required*
    resources.qrc   --> *likely useful*
    resources.py    --> *compiled version, likely useful*
    form.ui         --> *likely useful*
    form.py         --> *compiled version, likely useful*
```

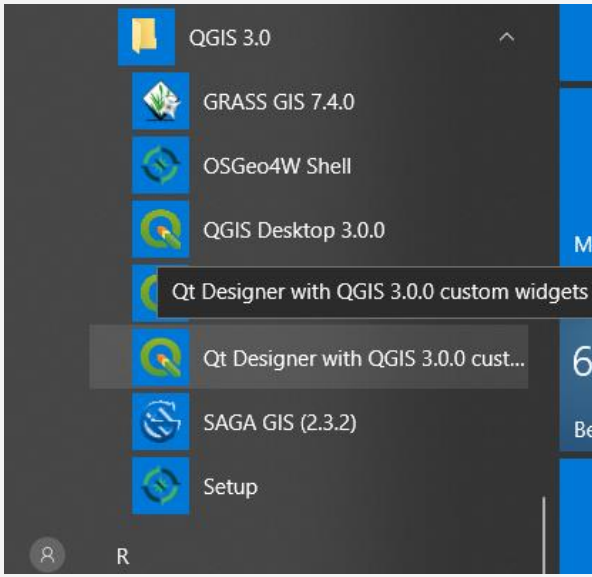
metadata.txt

metadata.txt - WordPad

```
# This file contains metadata for your plugin. Since
# version 2.0 of QGIS this is the proper way to supply
# information about a plugin. The old method of
# embedding metadata in __init__.py will
# is no longer supported since version 2.0.

# This file should be included when you package your plugin.#
Mandatory items:

[general]
name=Empty Field Remover
qgisMinimumVersion=3.0
description=This plugin removes empty fields from a vector file
version=0.1
author=University of Muenster
email=j.a.verstegen@uni-muenster.de

about=This plugin removes all fields that are completely empty
from the shapefile.
```

2, 3, 4. Qt with pyuic5 and pyrcc5

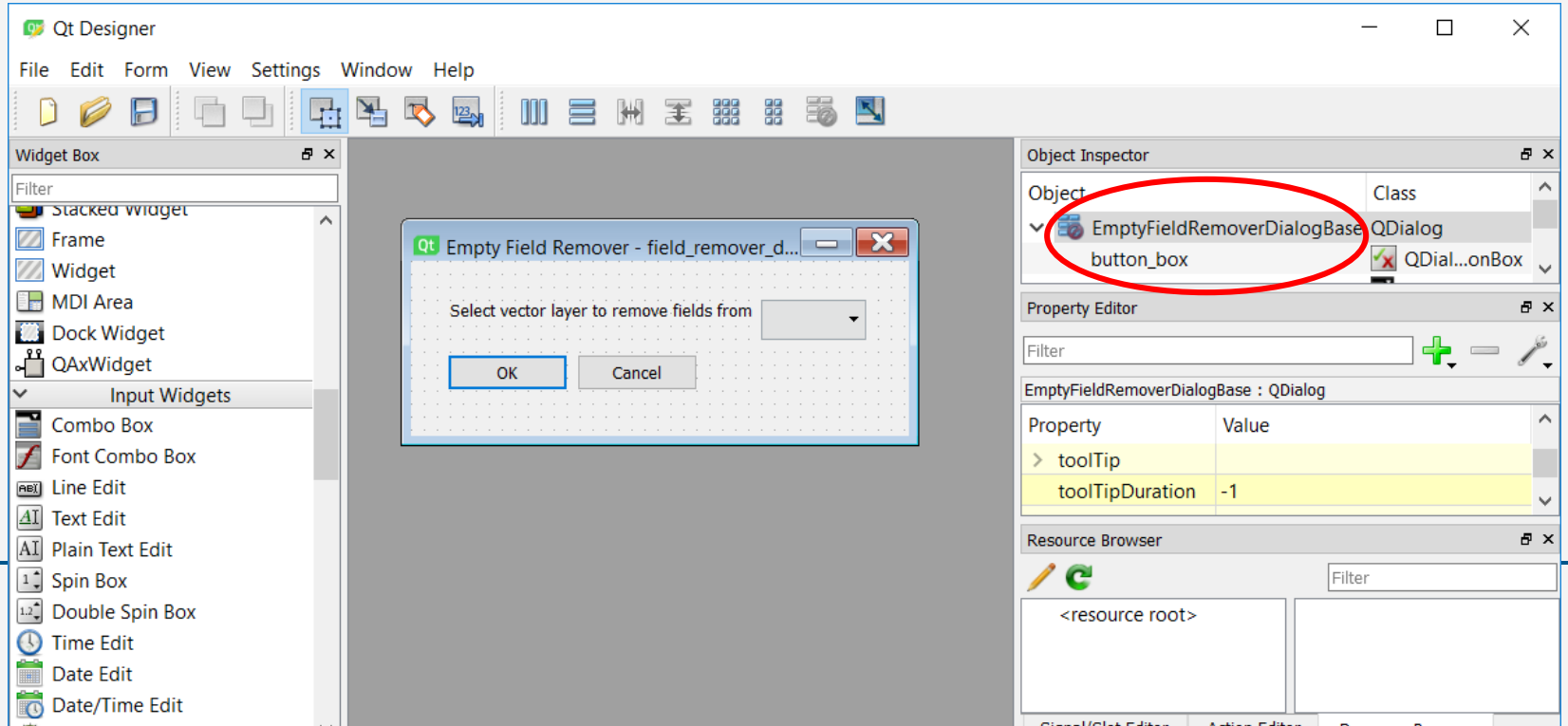
field_remover_dialog_base.ui



Open the file with Qt Designer

Add label + box by dragging and dropping widgets and save!

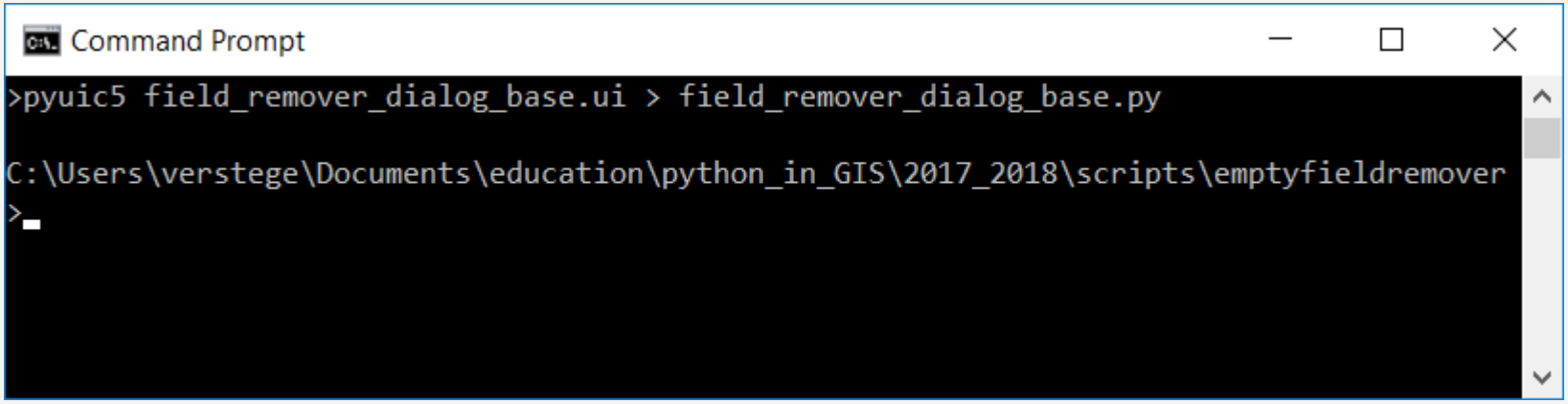
Note the name of the selection object is comboBox. Interact with this object in the Python script by this name.



Once finished, auto-translate .ui to .py

Type at the location of your plugin:

```
pyuic5 field_remover_dialog_base.ui > field_remover_dialog_base.py
```



if you get the error "`'pyuic5' is not recognized as an internal or external command,`", run the commands on the next slide first

If pyrcc5 and/or pyuic5 not recognized (error previous slide):

Try running the following:

```
call "C:\Program Files\QGIS 3.4\bin\o4w_env.bat"
call "C:\Program Files\QGIS 3.4\bin\qt5_env.bat"
call "C:\Program Files\QGIS 3.4\bin\py3_env.bat"
```

Change the path above according to the location of your QGIS installation

This is not permanent, so you have to do it again each time you open a new prompt or save it as a batch file, together with the command on the previous slide.

Result (don’t edit this file!)

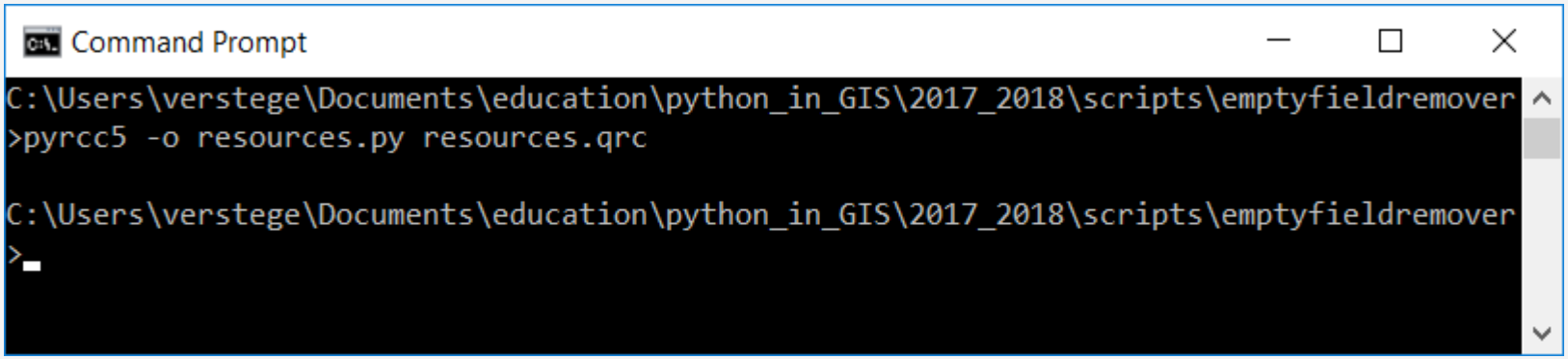
field_remover_dialog_base.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'field_remov
#
# Created by: PyQt5 UI code generator 5.9
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_EmptyFieldRemoverDialogBase(object):
    def setupUi(self, EmptyFieldRemoverDialogBase):
        EmptyFieldRemoverDialogBase.setObjectName("EmptyFieldRemo
        EmptyFieldRemoverDialogBase.resize(416, 173)
        self.button_box = QtWidgets.QDialogButtonBox(EmptyFieldRe
        self.button_box.setGeometry(QtCore.QRect(20, 120, 341, 32
        self.button_box.setOrientation(QtCore.Qt.Horizontal)
        self.button_box.setStandardButtons(QtWidgets.QDialogButto
        self.button_box.setObjectName("button_box")
        self.mQgsFileWidget = QgsFileWidget(EmptyFieldRemoverDial
        self.mQgsFileWidget.setGeometry(QtCore.QRect(30, 60, 331,
```

Compile

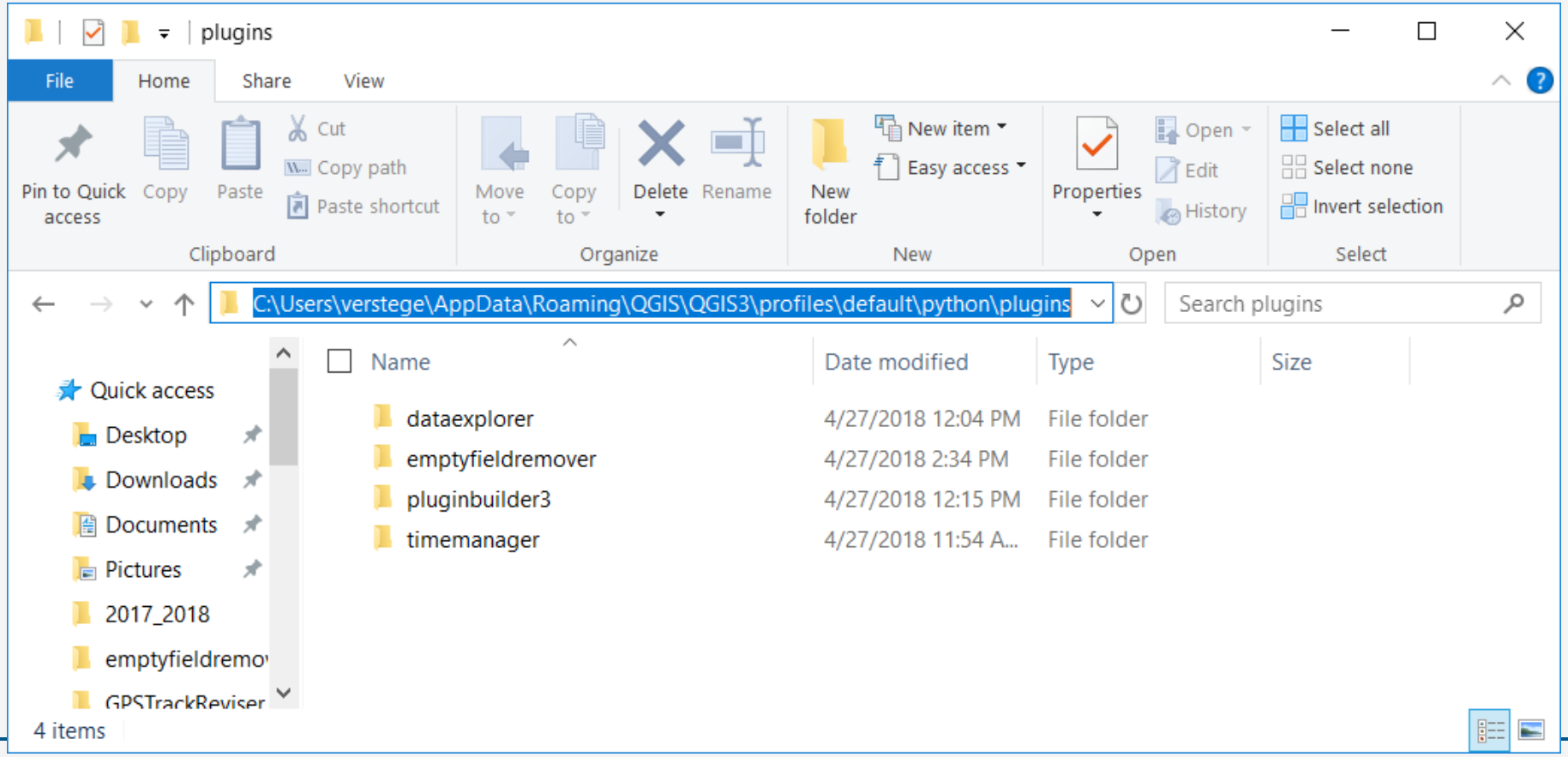
Type at the location of your plugin (in Windows without the >):

```
pyrcc5 -o resources.py > resources.qrc
```

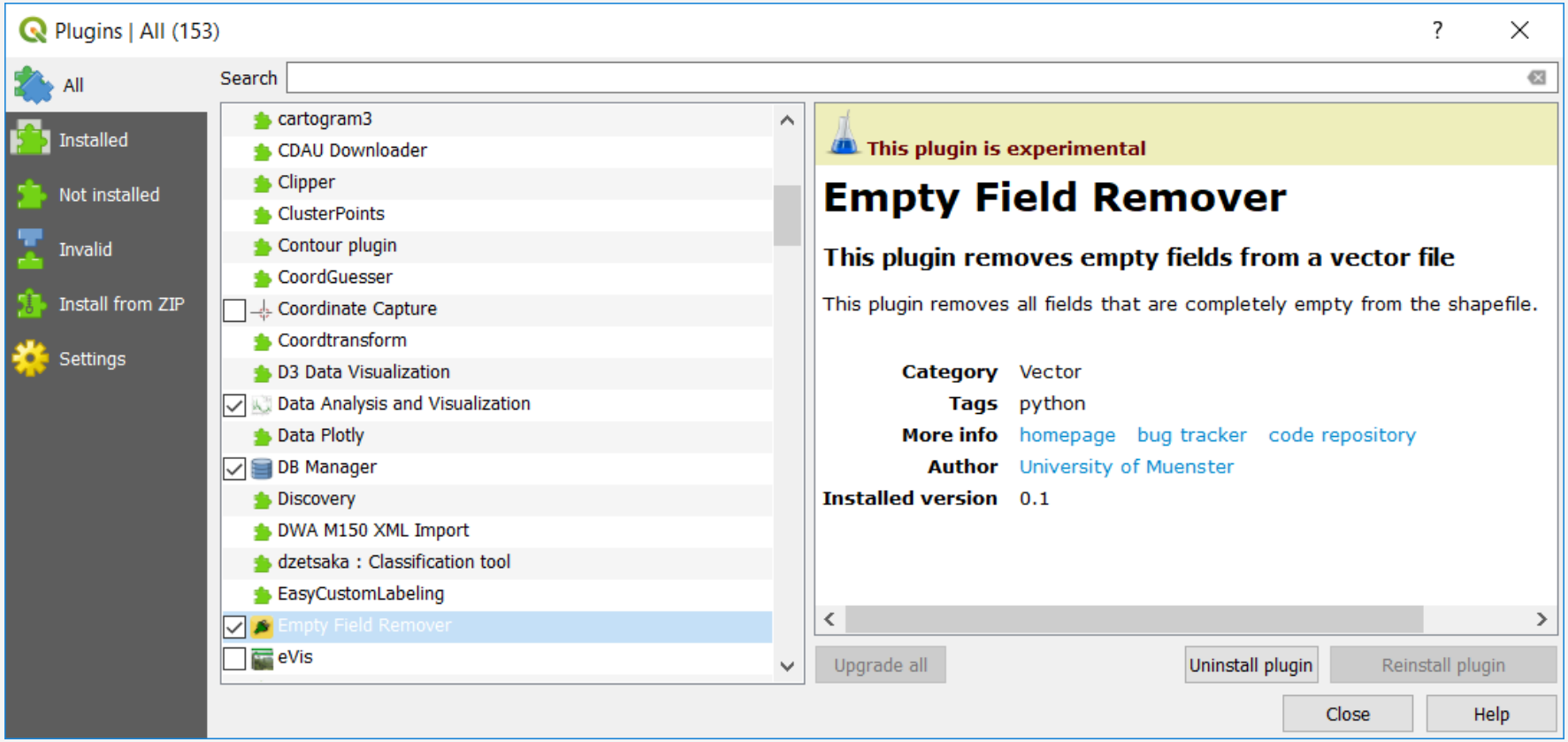


It is advised to make a copy before doing this! If you're unlucky, you need to start again.

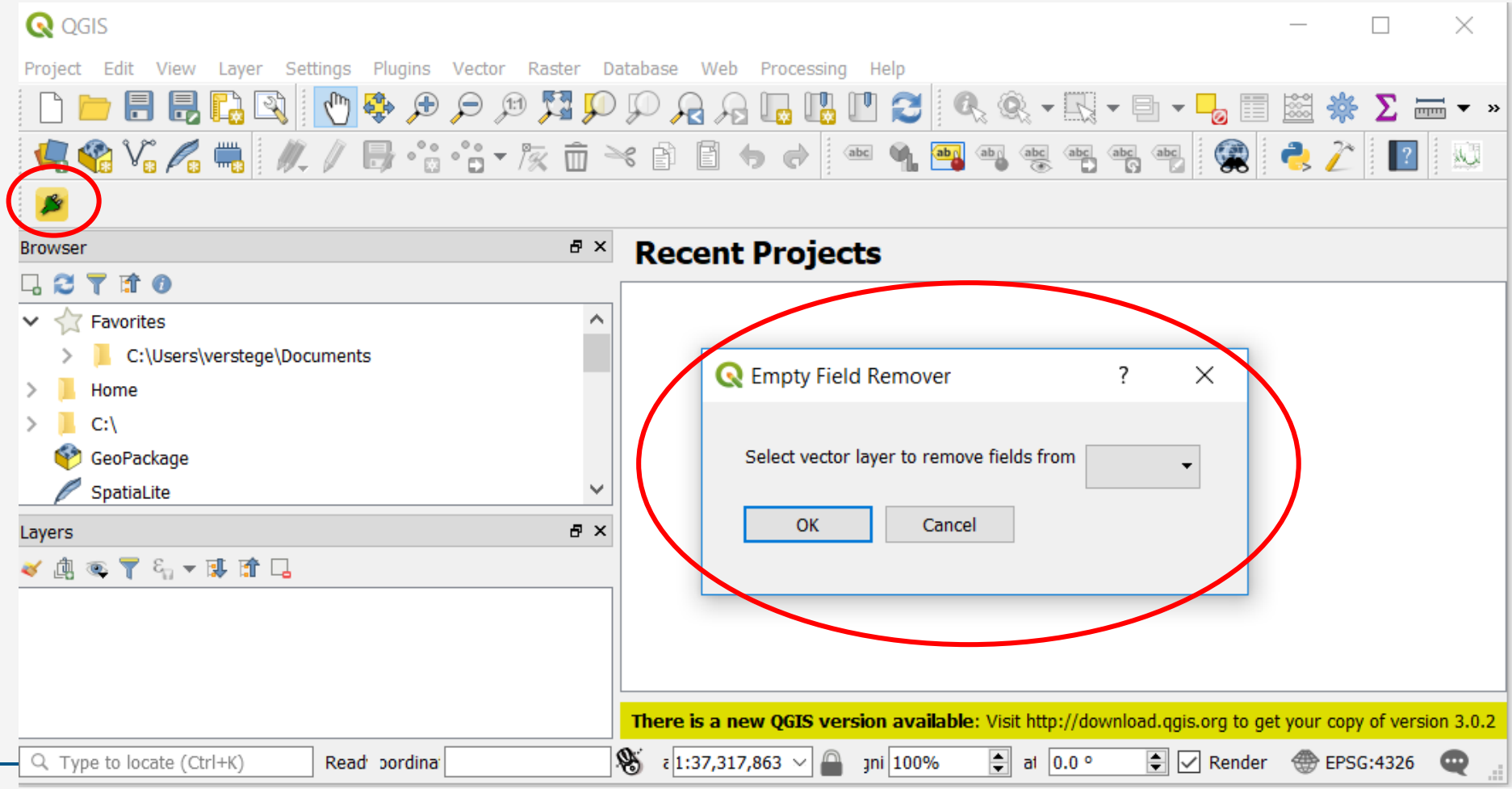
Copy the whole directory to one of the two plugin folders (see slide 9)



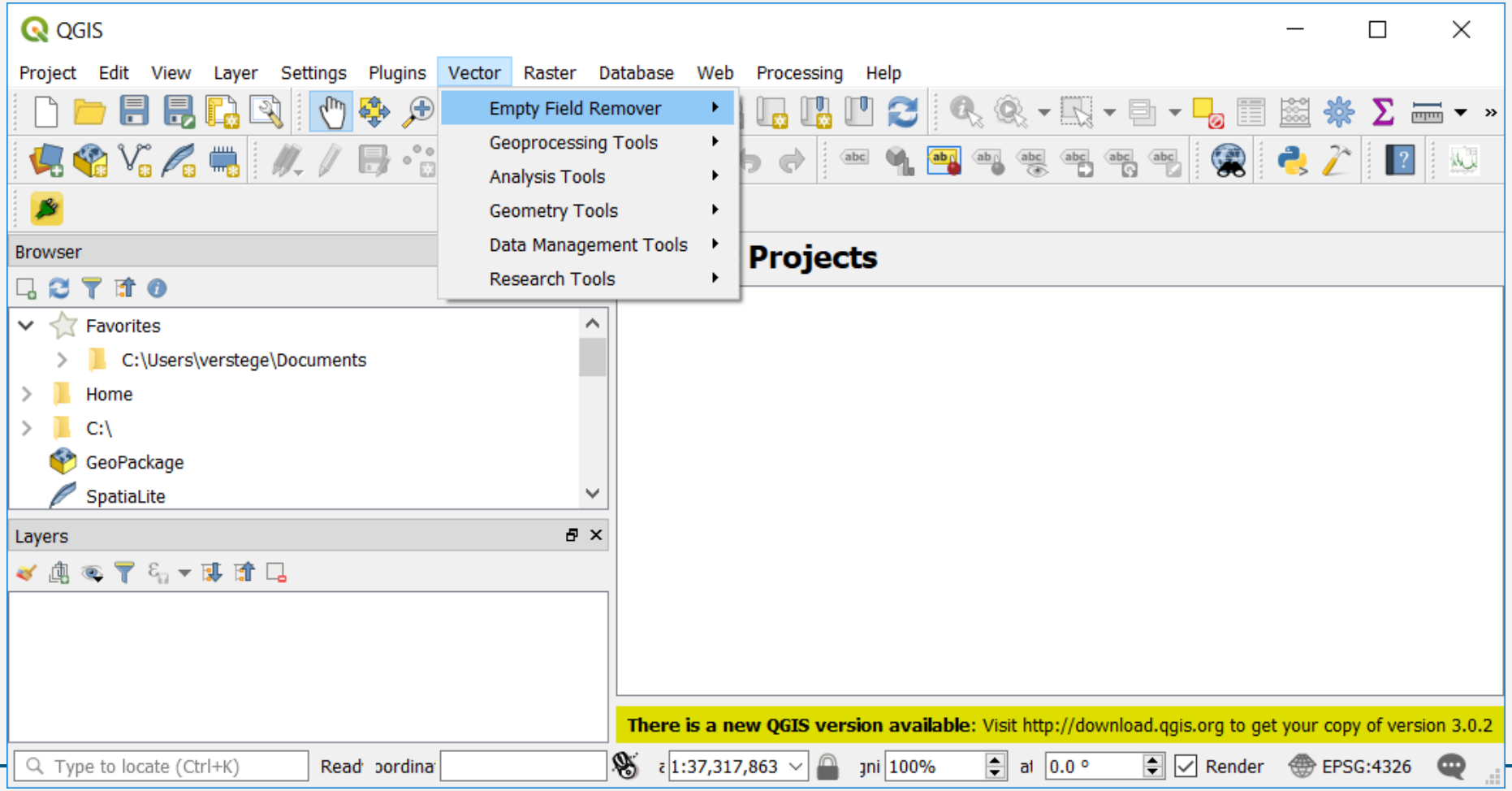
Now the (non-working) plugin can be loaded



Result


QGIS
Project Edit View Layer Settings Plugins Vector Raster Database Web Processing Help
Browser
Favorites
C:\Users\verstege\Documents
Home
C:\
GeoPackage
SpatiaLite
Layers
Recent Projects
Empty Field Remover
Select vector layer to remove fields from
OK
Cancel
There is a new QGIS version available: Visit http://download.qgis.org to get your copy of version 3.0.2
Type to locate (Ctrl+K)
1:37,317,863
100%
0.0 °
Render
EPSG:4326

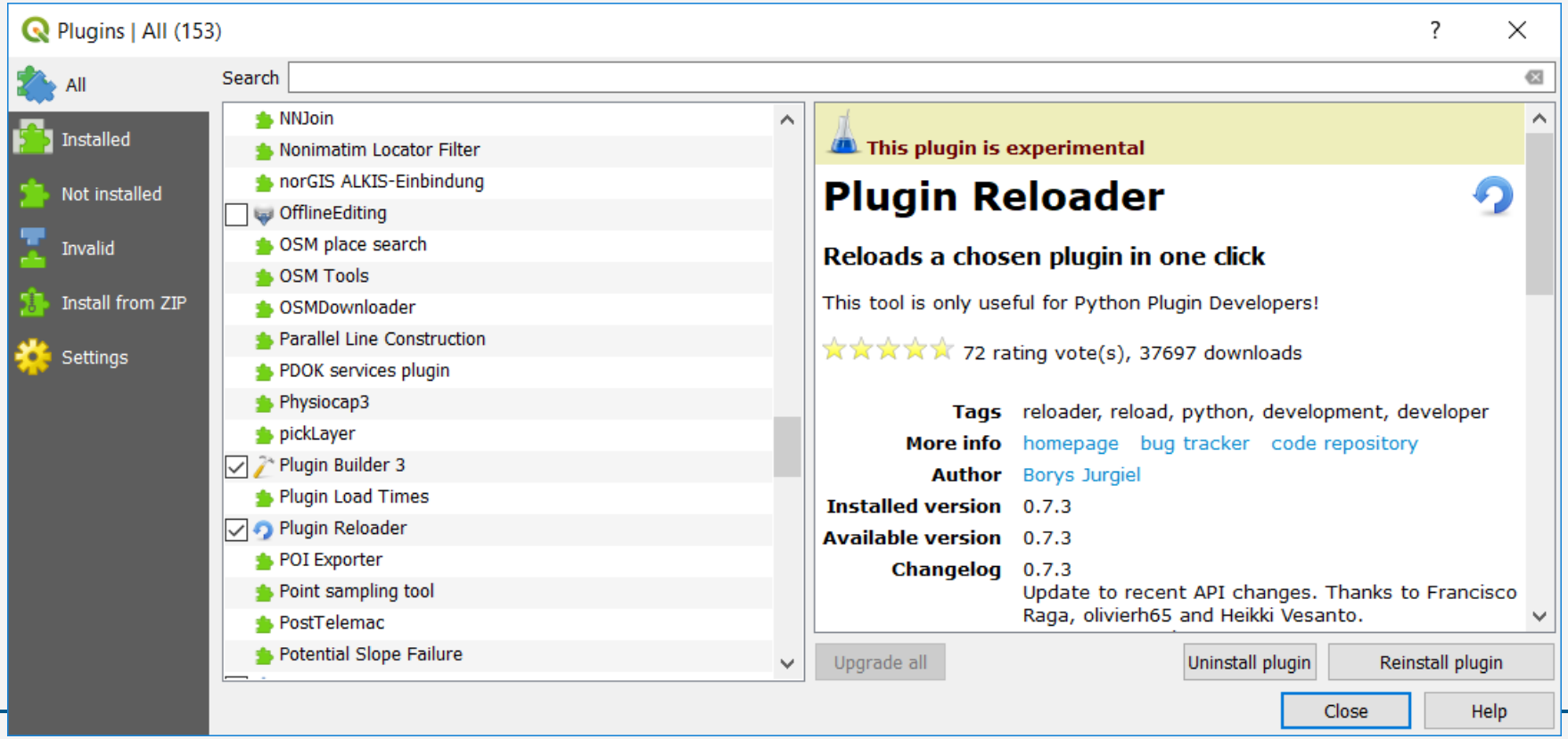
Result


QGIS
Project
Edit
View
Layer
Settings
Plugins
Vector
Raster
Database
Web
Processing
Help
Empty Field Remover
Geoprocessing Tools
Analysis Tools
Geometry Tools
Data Management Tools
Research Tools
Browser
Projects
Favorites
C:\Users\verstege\Documents
Home
C:\
GeoPackage
SpatiaLite
Layers
There is a new QGIS version available: Visit http://download.qgis.org to get your copy of version 3.0.2
Type to locate (Ctrl+K)
1:37,317,863
100%
0.0 °
Render
EPSG:4326

5. Plugin reloader

Now, add the plugin reloader

With this plugin you don’t have to restart QGIS each time you change your plugin



6. Edit main Python script

WWU MÜNSTER

field_remover.py

```
        git sha              : $Format:%H$
        copyright            : (C) 2018 by University of Muenster
        email                : j.a.verstegen@uni-muenster.de
 ***************************************************************************

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************
"""
from PyQt5.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QAction

# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .field_remover_dialog import EmptyFieldRemoverDialog
import os.path

```

Notice: class and constructor

field_remover.py

```
34
35  class EmptyFieldRemover:
36      """QGIS Plugin Implementation."""
37
38      def __init__(self, iface):
39          """Constructor.
40
41          :param iface: An interface instance that will be passed t
42              which provides the hook by which you can manipulate t
43              application at run time.
44          :type iface: QgsInterface
45          """
46          # Save reference to the QGIS interface
47          self.iface = iface
48          # initialize plugin directory
49          self.plugin_dir = os.path.dirname(__file__)
50          # initialize locale
51          locale = QSettings().value('locale/userLocale')[0:2]
```

Connection to the created dialog (1)

Scroll down and find the `run(self)` method. This is what it looks like originally:

field_remover.py

```
179                 action)
180             self.iface.removeToolBarIcon(action)
181         # remove the toolbar
182         del self.toolbar
183
184
185     def run(self):
186         """Run method that performs all the real work"""
187         # show the dialog
188         self.dlg.show()
189         # Run the dialog event loop
190         result = self.dlg.exec_()
191         # See if OK was pressed
192         if result:
193             # Do something useful here - delete the line con
194             # substitute with your code.
195             pass
196
```

Connection to the created dialog (2)

Add the following code:

field_remover.py

```
185     def run(self):
186         """Run method that performs all the real work"""
187         # First clear the combobox
188         # from previous time you ran the plugin
189         self.dlg.comboBox.clear()
190         # Add all layers in the QGIS menu to the dialog
191         layers = self.iface.mapCanvas().layers()
192         layer_list = []
193         for layer in layers:
194             layer_list.append(layer.name())
195         self.dlg.comboBox.addItems(layer_list)
196         # show the dialog
197         self.dlg.show()
198         # Run the dialog event loop
199         result = self.dlg.exec_()
200         # See if OK was pressed
201         if result:
202             # Do something useful here - delete the line con
```

The processing itself (1)

Everything you want your plugin to do should be coded in the `run(self)` method where it says `pass`.

Note that the `comboBox` can return the index of the selected item by:

```
self.dlg.comboBox.currentIndex()
```

The processing itself (2)

So, your script will continue like this:

field_remover.py

```
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            # Do something useful here - delete the line containi
            # substitute with your code.

            # Pick the layer that is selected by the user
            layer = layers[self.dlg.comboBox.currentIndex()]

```

The processing itself (3)

- You may need to import other libraries
- You may create new functions in the class. Notice how to run functions of a class from within the class itself:

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Exercise #3

Repeat all steps to create your own plugin

OR use the already created empty plugin provided in the scripts folder

- Finish the plugin, such that:
  - It loads the shapefile chosen by the user
  - It removes all fields from this shapefile that are **completely empty (no attribute values) or consist of only zeroes**