Curso PyQGIS

Matthias Kuhn, Germán Carrillo

Bogotá, 25.2.2018

OPENGIS.ch
ANDROID · [Q]GIS · WEB

Notes

Outline

Notes

Schedule and topics	
08.00 - 08.30	Welcome and course introduction (computer configuration, data, . . .)
08.30 - 09.30	Introduction to the QGIS python console, Introduction to Python
09.30 - 10.00	Introduction to actions in QGIS
10.00 - 10.15	Coffee break
10.15 - 12.00	Development of an action
12.00 - 13.00	Lunch
13.00 - 14.00	Object oriented programming concepts
14.00 - 15.00	PyQt, Widgets, Signals and Slots
15.00 - 15.15	Break
15.15 - 17.00	Creation of a QGIS Plugin

Notes

Matthias Kuhn

- Geographer and application developer
- Core developer QGIS

Notes

Germán Carrillo

- Works at BSF Swissphoto - INCIGE SAS
- Administrator of GeoTux
- Core contributor QGIS

Notes

Presentation participants

Notes

Configuration of the computers

- WiFi
- QGIS
- QtDesigner
- Text editor
- PyCharm

Notes

Data

- https://goo.gl/mCL4EC

Notes

QGIS development

- QGIS is developed with a plugin architecture.
- QGIS is developed in C++ but there is a Python API which allows to interact with its elements (interface, layers, features, ...)
- QGIS also offers a Python console which allows to execute Python code directly inside QGIS without the need to use an external Python editor.

Notes

What is Python

- Python is a powerful programming language, easy to learn.
- Python is an ideal language to write scripts and for rapid development in many types of applications and on many platforms.
- **Python 2 vs Python 3**

Notes

The python console

- How to open
- Integrated editor
- The options

Notes

First Python script

```
1 # My first script
2 import sys
3 print(sys.platform)
4 print(2 ** 100)
5 x = 'Spam!'
6 print(x * 8)
```

Notes

Python's core data type

- Numbers
- Strings
- Lists
- Dictionaries
- Tuples
- Files
- Sets
- Other core types (booleans, types, None)
- Program unit types (functions, modules, classes)
- Implementation-related types (compiled code, stack tracebacks)

Notes

Numbers and arithmetic operators

```
# Constants and variables
2 + 2 # Addition d'entiers
2 - 2 # Soustraction d'entiers
10 / 3 # Division, in Python 2 returns an integer in
    Python 3 a float
10 / 3.0 # Floating point division
int(10 / 3.0) # Integer division
10 // 3 # Integer division
10 % 3 # The rest of a division (modulo)
2 * 2 # Multiplication
2 ** 100 # 2 to the power of 100
a = 100 # Assign a variable
a / 2 # Dividing the content of a variable
```

Notes

Strings

```
'spam eggs'  # single quotes
"spam eggs"  # double quotes
```

```
# 3 times 'un', followed by 'ium'
print(3 * 'un' + 'ium')
prefix = 'Py'
print(prefix + 'thon')
```

```
'unununium'
'Python'
```

Notes

Indexing strings

```
word = 'Python'
print(word[0])  # character in position 0
print(word[5])  # character in position 5
print(word[-1])  # last character
print(word[-2])  # second-last character
print(word[-6])
```

```
'P'
'n'
'n'
'o'
'P'
```

Notes

Cutting strings

```
print(word[0:2])  # characters from position 0 (
    included) to 2 (excluded)
print(word[2:5])  # characters from position 2 (
    included) to 5 (excluded)
```

```
'Py'
'tho'
```

Notes

Measuring string lengths

```
s = 'supercalifragilisticexpialidocious'
print(len(s))
```

```
34
```

Notes

Structuring data

- Lists
- Ensembles
- Dictionnaires

Notes

Lists

```
squares = [1, 4, 9, 16, 25]
print(squares)
print(squares[0])
print(squares[-1])
print(squares[-3:])
```

```
[1, 4, 9, 16, 25]
1
25
[9, 16, 25]
```

Notes

Sets

```
basket = ['apple', 'orange', 'apple', 'pear', 'orange'
    , 'banana']
fruit = set(basket) # create a set without duplicates
print(fruit)
print('orange' in fruit) # fast membership testing
print('kumquat' in fruit)
```

```
set(['orange', 'pear', 'apple', 'banana'])
True
False
```

Notes

Dictionaries

```
tel = {'Jack': 4098, 'John': 4139}
tel['Guido'] = 4127
print(tel)
print(tel['Jack'])
```

```
{'John': 4139, 'Guido': 4127, 'Jack': 4098}
4098
```

Notes

For loop

```
# Measure some strings:
words = ['cat', 'window', 'defenestrate']
for w in words:
    print w, len(w)
```

```
cat 3
window 6
defenestrate 12
```

Notes

range() function

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
    for i in range(len(a)):
        print i, a[i]
```

```
0 Mary
1 had
2 a
3 little
4 lamb
```

Notes

While loop

```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
```

```
1
1
2
3
5
8
```

Notes

Conditions

```
x = 42
if x < 0:
    x = 0
    print 'Negative changed to zero'
elif x == 0:
    print 'Zero'
elif x == 1:
    print 'Single'
else:
    print 'More'
```

```
More
```

Notes

Functions

```
def distance(point1, point2):
    return math.sqrt(
            (point1[0] - point2[0]) ** 2
            +
            (point1[1] - point2[1]) ** 2
        )

point1 = [10, 15]
point2 = [20, 25]
distance(point1, point2)
```

```
14.142135623730951
```

Notes

Exceptions

```
try:
    a = int('string')
except:
    print('Value not valid')
```

```
Value not valid
```

Notes

Where to get Python help (1)

```
dir()
```

Without any arguments, dir produces a list of names within the local namespace. With an argument, dir produces a list of valid attribute names for an object.

```
help()
```

Invokes the native help system. This function is meant to be used in interactive mode. Without a parameter, an interactive help system is started. If the parameter is a string, a module, a function, a class, a method, a keyword or a subject of the documentation, the help system is searched and a help page is printed to the console.

Notes

Where to get Python help (2)

- `https://www.python.org`: This is the primary help page for python. It contains code snippets, help and links to related pages on the internet.
- `https://pypi.python.org/pypi`: «The Python Package Index», is a catalog of Python modules available for download, made by users.
- `https://code.activestate.com/recipes/langs/python/`: «The Python Cookbook» is a very good information source with code examples, modules and scripts.

Notes

Exercises

1. Print three different strings
2. Print the 3 first characters of a string
3. Print the numbers from 0 to 100
4. Create a program that prints the text "small" if a variable is smaller than 10 and "big" if the variable is greater or equal to 10.
5. Create a program that divides two numbers and prints a) the integer result and b) the rest of the division.

Notes

Interaction with QGIS elements

- Python API documentation
  `https://python.qgis.org/api/`
- C++ API documentation
  `http://qgis.org/api/index.html`
- List of API changes from QGIS 2 to 3
  `https://qgis.org/api/api_break.html`

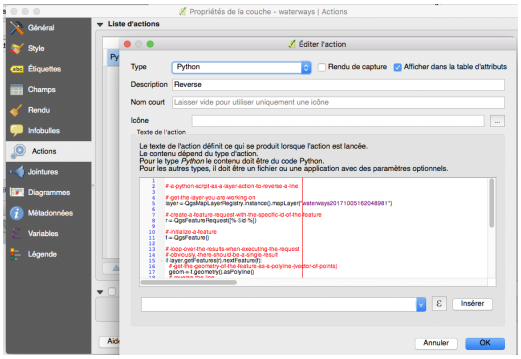
Notes

Layer actions

- actions allow to execute code on the objects of a layer
- they are ideal to create an own "*map tool*"
- the integrate very well into QGIS workflows
- they are integrated in the project (no need to deploy a separate plugin)

Notes

Layer actions



Notes

Exercice: Invert the direction of a line

1. request the identified feature
2. get the geometry and invert the line
3. replace the geometry in the feature
4. refresh the map

Notes

1. request the identified feature

```
# get the layer you are working on
layer = QgsProject.instance().mapLayer("
    _your_layer_id_")

# create a feature request with the specific id of the
    feature
r = QgsFeatureRequest([% $id %])

f = next(layer.getFeatures(r))
```

Notes

2. get the geometry and invert the line

```
# get the geometry of the feature as a multi polyline
    (list of list of points)
lines = f.geometry().asMultiPolyline()

# reverse the line
for line in lines:
    line.reverse()

# recreate a geometry
geom = QgsGeometry.fromMultiPolylineXY(lines)
```

Notes

3. replace the geometry in the feature

```
# edit the geometry
if layer.changeGeometry([% $id %], geom):
    # trigger repaint on success
    layer.triggerRepaint()
else:
    # print error otherwise
    print("could not edit geometry, turn editing on")
```

Notes

Let's evaluate

1. which geometry types does this work for?
2. what about Z/M and curves?
3. performance implications

Notes

Optimized version

```
geom = QgsGeometry(feature.geometry().constGet().
    reversed())
```

Notes

Go further...

- Ensure that the obtained feature is valid to avoid errors
- show nice messages to the user (Hint: QgsMessageBar)

Notes

Exercises PyQGIS

1. Create 3 layers, polygon, line and point and insert 2 features in each layer.
2. Save the 3 layers each in a separate file.
3. Measure the distance between the 2 points on the point layer
4. Change the colour of the two polygons of the previous exercise
5. Add an attribute to the lines with a name

Notes

Geometry: Points, Lines, Polygons

- Geometry is an abstract concept
- Geometry has attributes and methods that points, lines and polygons share (e.g., SRS, is_empty(), is_valid())
- Child classes have specialized attributes and methods (e.g., points have a single pair of coordinates, only polygons have area, etc.)
- Therefore, Geometry is a parent class whereas Point, Line and Polygon are children classes

Notes

A simple class in Python

```
class MyClass:
    def __init__(self):
        self.a = 10

    def my_method(self):
        print("Attribute 'a':{}".format(self.a))

instance = MyClass()
print(instance.a)
instance.my_method()
```

```
10
Attribute 'a':10
```

Notes

1. defining the Geometry class

```
class Geometry:
    def __init__(self):
        self.epsg = 3116
        self.type = "Unknown"

    def print_details(self):
        print("EPSG: {}, TYPE: {}, VALID: {}".format(
            self.epsg, self.type, self.is_valid()))

    def is_valid(self):
        return False
```

Notes

2. define the Point class (inheriting from Geometry)

```
class Point(Geometry):
    def __init__(self, x=None, y=None):
        super().__init__()
        self.x = x
        self.y = y
        self.type = "Point"

    def distance_to_point(self, point2):
        if self.is_valid():
            return math.sqrt((point2.x - self.x) ** 2
                + (point2.y - self.y) ** 2)
        else:
            return None

    def is_valid(self):
        if self.x is None or self.y is None:
            return False

        return True
```

Notes

3. let's create and use points

```
# Create a point and print its details
p1 = Point(10,10)
p1.print_details()
```

```
EPSG: 3116, TYPE: Point, VALID: True
```

```
# Create another point and get distance between point
    1 and point 2
p2 = Point(15, 15)
print("Distance to second point: {}".format(p2.
    distance_to_point(p1)))
```

```
Distance to second point: 7.0710678118654755
```

Notes

4. write the Line class (inheriting from Geometry)

```
class Line(Geometry):
    def __init__(self, ?):
        super().__init__()
        self.points = [point1, point2]
        ...

    def length(self):
        ...

    def is_valid(self):
        ...
```

Notes

5. let's create and use lines

```
# Create a line based on our created points
line = Line(p1, p2)
line.print_details()
print("Line length: {}".format(line.length()))
```

```
EPSG: 3116, TYPE: Line, VALID: True
Line length: 7.0710678118654755
```

Notes

Qt

- object oriented API, coded in C++ and multi-platforms
- First release in 1995, Qt 4 in 2005, Qt 5 in 2012
- Documentation `https://doc.qt.io/qt-5/qtwidgets-index.html`
- base library for QGIS user-interface
- bindings for many languages: Ada, C#, Java, Ruby, Visual Basic, Python...

Notes

PyQt

- Qt Bindings for Python
- Free (as in free beer) for open-source development
- Other bindings have been deprecated (PySide) or are less widely spread (PythonQt)

Notes

Creation of a graphic user interface (GUI)

- Use Qt Designer (Qt Creator module)

Notes

Widgets

- QLabel for simple text (HTML handling)
- QLineEdit to prompt user for text
- QCheckBox for binary choices
- QComboBox for list-based choices
- QGroupBox and QTabWidget to visually group elements
- Double-click or right-click to edit elements

Notes

Layouts

- Grid Layout
- Form Layout
- Vertical Layout
- Horizontal Layout

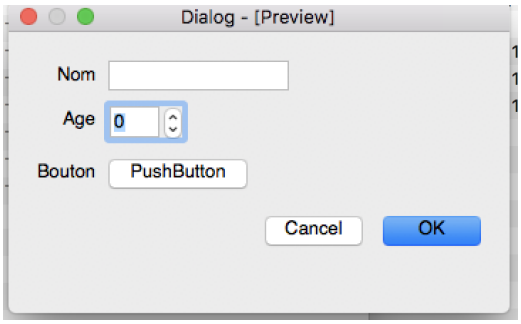
Notes

Properties

- specific (*text*, *checkedState*, ...)
- *toolTip* to provide context informations
- *enabled* to prevent editing
- *objectName* as identifier
- and many others!

Notes

Qt Designer exercise



Notes

Loading the graphic interface

```
from PyQt5 import uic

# Load the .ui file
DialogUi, DialogType = uic.loadUiType('/home/mkuhn/dev
    /PyQGIS-course/code/widgets/dialog.ui')

# A class for logic defined in the user interface
class MyDialog(DialogType, DialogUi):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

dialog = MyDialog()

dialog.show()
# or
# dialog.exec_()
```

Notes

pyuic vs uic.loadUiType

- uic.loadUiType: embed loading in Python
- pyuic: compiling the ui file into Python code
  `pyuic5 -o dialog_ui.py dialog.ui`
- Under Windows, use either *Osgeo4W shell* or *pb tools*
- Under Unix, use *terminal* or *make*

Notes

Interaction

- Action required whenever an event occurs on a widget (e.g. click, value change, loosing focus, etc.)
- Qt introduces Signals / Slots concept
  `http://doc.qt.io/qt-5/signalsandslots.html`
- Definition either in the UI file (discouraged) or in the Python code

Notes

Button

```
7  class MyDialog(DialogType, DialogUi):
8      def __init__(self):
9          super().__init__()
10         self.setupUi(self)
11         self.pushButton.clicked.connect(self.
               onButtonClicked)
12
13     def onButtonClicked(self):
14         iface.messageBar().pushMessage('Well done!
               Here I am.')
15
16
17
18 dialog = MyDialog()
19
20 dialog.show()
21 # or
```

Notes

Exercises

1. add a list menu (QComboBox) with several entries
2. display the text of the menu in the message bar whenever it changes
3. copy the text in the line edit at each change

Notes

Resource file

- *resources.qrc* contains the images
- mandatory compilation through Osgeo4W shell
  `pyrcc5 -o resources.py resources.rc`
  or with *pb tools*

Notes

Bonus: Map Tools

```
def onCanvasClicked(self, point, mouseButton):
    """Calculate the average floor numbers over a 20
       x20 map units (meters for Switzerland) area"""
    request = QgsFeatureRequest()
    request.setFilterRect(QgsRectangle(point.x()-10,
       point.y()-10, point.x()+10, point.y()+10))
    features = self.buildingsLayer.getFeatures(request
       )

    floors = [feature['floors'] for feature in
       features]

    avg = sum(floors)/float(len(floors))
    print('Average floor number in this region is {}'.
       format(avg))

mapTool = QgsMapToolEmitPoint(iface.mapCanvas())
mapTool.canvasClicked.connect(self.onCanvasClicked)
iface.mapCanvas().setMapTool(mapTool)
```

Notes

Demonstration

- Python API documentation
  `https://python.qgis.org/api/`
- C++ API documentation
  `http://qgis.org/api/index.html`
- List of API changes from QGIS 2 to 3
  `https://qgis.org/api/api_break.html`

Notes

Inspect the files in the minimal plugin

- metadata.txt
- __init__.py
- minimal_plugin.py

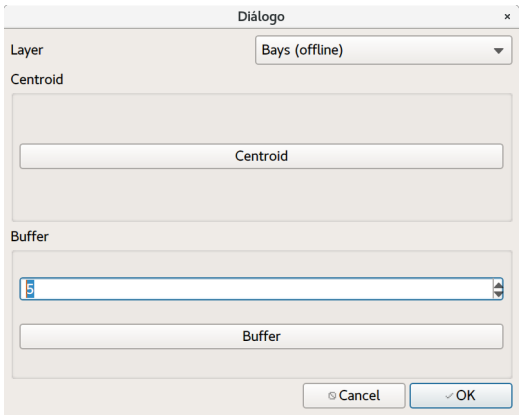
Notes

Create a copy of the plugin and adjust

- Install *Plugin Reloader* (experimental) to be able to reload plugin with code modifications
- Create a copy of the plugin
- Adjust the template files to your liking
- Start QGIS and enable the plugin

Notes

Create the plugin dialog



Notes

Create the code for the dialog

```
from PyQt5 import uic
import os
from qgis.core import QgsVectorLayer

DialogBase, DialogType = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), '
    geometry_operation_dialog_base.ui'))


class GeometryOperationDialog(DialogType, DialogBase):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setupUi(self)
```

Notes

Load the layers

```
def loadLayers(self):
    for layer in QgsProject.instance().mapLayers().
        values():
        self.layerComboBox.addItem(layer.name(), layer
            )
```

Notes

Load the layers

```
def buffer(self):
    layer = self.layerComboBox.currentData()

    # Create a new scratch layer based on the original
        layer
    uri = "Polygon?crs={authid}&index=yes".format(
        authid=layer.crs().authid())
    new_layer = QgsVectorLayer(uri, "Buffer", "memory"
        )

    QgsProject.instance().addMapLayer(new_layer)
```

Notes

Load the layers

```
new_layer.startEditing()

# Add all attribute definitions from the source layer
for field in layer.fields():
    new_layer.addAttribute(field)

# Process all features
for feature in layer.getFeatures():
    # Here is the important call: create a buffer
    geom = feature.geometry().buffer(buffer_size, 5)

    feature.setGeometry(geom)
    new_layer.addFeature(feature)

# Save the features to the memory layer
new_layer.commitChanges()
```

Notes

Now the same for centroids

- It should be easy, right :)

Notes

Make it shine

- Only show vector layers
- Enable and disable group boxes based on the geometry type (wkt type) of the current layer
- Add a checkbox to only process selected features
- Look up other geometry methods in the QGIS documentation
- Use QGIS custom widgets (QgsMapLayerComboBox)
- Add another combobox to choose an attribute and a button to calculate statistics for this attribute

Notes

Where to get inspiration

- Existing plugins `http://plugins.qgis.org`
- QGIS source code `https://github.com/qgis/QGIS`

Notes

Where to get help

- QGIS mailing list (dev / user) `https://www.qgis.org/site/getinvolved/mailinglists.html`
- GIS Stackexchange `https://gis.stackexchange.com/`
- OpenGIS.ch support

Notes