GeoPython 2018
May 07-09

QGIS Processing Framework: Automating Tasks with Python

Prof. Stefan Keller
Kang Zi Jing

Geometa Lab
HSR Hochschule für Technik
Rapperswil

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Processing Framework: Automating Tasks with Python

Processing Framework - What is it?

- When you Google Processing...
  - Programming language
  - Programming software (IDE)
  - Processing.js (JS port)
  - Big data processing frameworks
  - Software Architecture Framework
- So which one are we actually talking about?
  - Processing Framework - a plugin for QGIS

What is QGIS?

“free and open-source cross-platform desktop geographic information system (GIS) application that supports viewing, editing, and analysis of geospatial data” -Wikipedia

- FOSS4G (Free and Open Source Software For Geospatial)
- GNU GPL (General Public License)
- A software that can display, compose, edit, export and analyze geospatial data and maps
- Supports raster and vector layers
- Supports multiple data sources:
    - File formats like GeoPackage, ESRI Shapefiles, DXF
    - Web Services like WMS and WFS
    - Databases like PostGIS, etc

QGIS and Processing Timeline

2002	2004	2007	2012
Conception of QGIS	**Launch of SEXTANTE**	**QGIS 0.9.0**	**Processing Plugin**
Development and written with Qt toolkit and C++, by Gary Sherman	Processing started as SEXTANTE, a library of geospatial analysis algorithms	QGIS 0.9.0 introduced the Python API that allowed developers to add new functionalities using Python	SEXTANTE became a QGIS core plugin and renamed to Processing

Early QGIS pre-Processing

- Lacked comprehensive framework for spatial analysis
- Geographic Resources Analysis Support System (GRASS GIS) plugin had many redundancies and was cumbersome and errorsome
- In general, the geoprocessing tools in core QGIS were:
    - Not homogenous and inconsistent
    - No code modularity and reusability
    - Isolation of tools

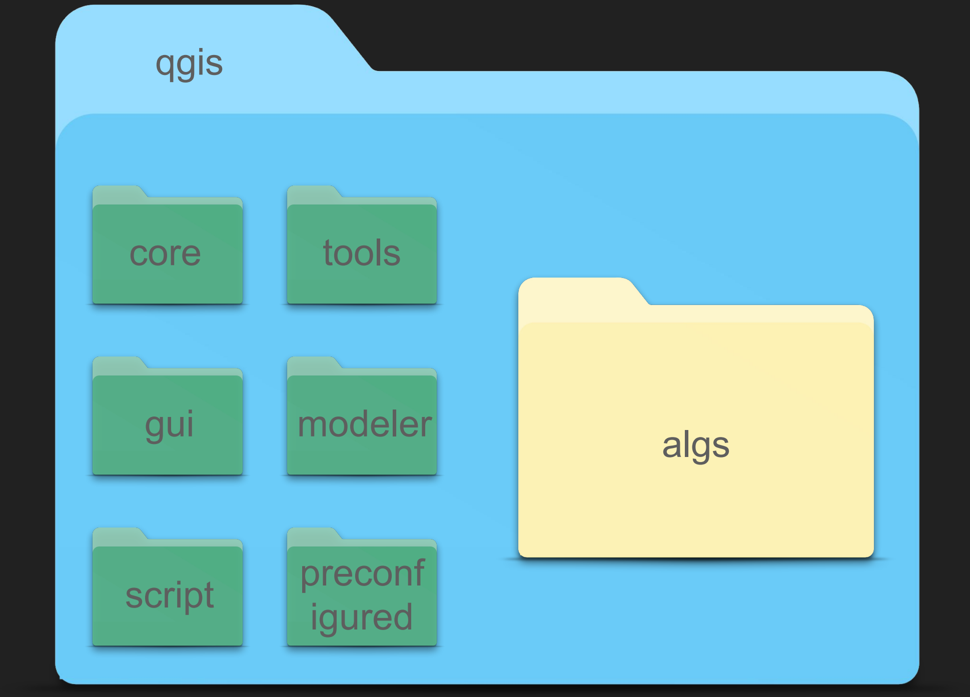
Processing Framework - Proper Introduction

- Built-in Python plugin
- Connected to QGIS with Python API (PyQt)
- Contains Toolbox binaries for R, Orfeo (Raster), SAGA GIS, GRASS GIS
- “Middleman” between these algorithms and tools and QGIS client, making them easy and convenient to use

Processing Framework - 4 Main Goals

- Efficiency
  - Efficient integration of libraries and binaries to make them easy and convenient to use
- Modularity
  - Promote consistency and reduce duplication of commonly used code blocks
- Flexibility
  - Implemented algorithms can be reused in any other graphical tools included, without additional work or conversion
- Automatic GUI generation
  - Processing helps generate GUI based on algorithm description so developers don’t have to

Processing Framework - Architecture

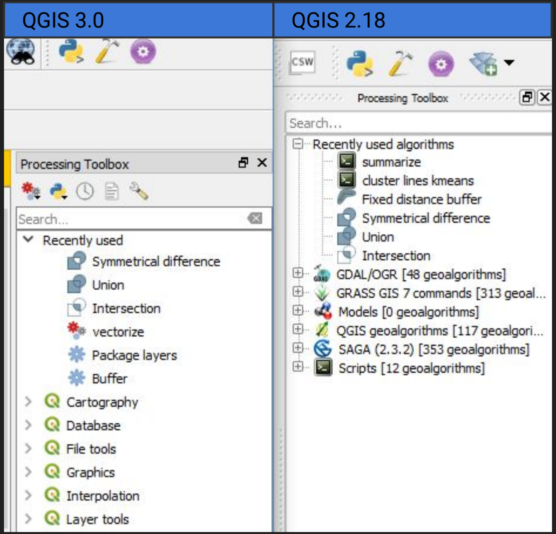


Processing Framework - How it Works

1. Plugin loads
2. *ProcessingPlugin* instance initializes *Processing* class
3. *ProcessingConfig* and *ProcessingLog* initialize
4. *AlgorithmProviders* loads
5. Each *AlgorithmProvider* contains list of *GeoAlgorithms*
6. These contain logic and specifications for the algorithms to run (param and output types)
7. Files that contain such specifications and logic of backend binaries can be found in the subfolders of *algs*

Processing Plugin Features

- Recently revamped!
- Toolbox (main element of GUI)
  - Execute a single algorithm once or a batch process of it
- Graphic Modeler
  - GUI where several algorithms can be combined into a workflow
  - Creates essentially a single process with several subprocesses
- History Manager
  - Easily reproduce previously used algorithms and features
- Batch Processing Interface
  - Execute batch processes
  - Automate execution of single algorithm with multiple datasets
- Scripts
  - Write user defined standalone scripts
  - These scripts can be implemented to run via Processing Framework



Advantages of Processing

- Obviously, workflow automation
- Documentation
- Integration of algorithms
- Can be called from command line and within Pythonic scripts
- Automatic GUI generation

Limitations of Processing

- Inflexible inputs and outputs
  - Optional parameters/outputs are not supported yet
- No room for interactivity
  - Doesn’t accept user inputs
- Reduced performance when conversions of outputs
  - Results parsed as inputs in a chain algorithm still requires conversion

Credits and References

- Processing: A Python Framework for the Seamless Integration of Geoprocessing Tools in QGIS by Anita Graser
    - In-depth development history on Processing Framework
- Processing GitHub repository by Victor Olaya (developer of Processing)
- QGIS 2.18 Documentation
    - Contains a lot of resources and documentations
    - Links to tutorials and textbooks like the PyQGIS Cookbook, QGIS Developers Guide
- QGIS Tutorials by Ujaval Gandhi
    - Helpful step by step tutorials on many aspects of QGIS
- Vast amount of resources, forums and an active and helpful community online
- Special thanks to helpful developers like Anita Graser and other users on GIS Stack Exchange for answering my questions
- And of course, the wonderful people at Geometa Lab, HSR

So… where to now?

Workflow Introduction - Autobahn Construction

- On the GitHub repository is a workflow that we would be attempting to automate today
- Based on Task 6 of the course *Introduction to GIS and Digital Cartography* by Class Leiner, University of Kassel, 2010
- Adapted by Prof. Keller in 2017 for teaching Vector Analysis
- Translated from German into English, updated for QGIS 3.0, and adapted for this workshop by me
- Translated workflow in English can be found in the GitHub repository, along with adapted problem sets and tasks for this workshop

Autobahn Construction - Introduction

- Geospatial analysis
  - On the effects of building a highway on nearby habitat types
- Working with buffers, intersections and other Geoprocessing tools
- Automation of workflow with Processing Framework
  - Graphic Modeler
  - Scripting with Python Console
  - Using Processing algorithms
  - Creating a Processing Script
  - Interactive scripts

Workflow Datasets

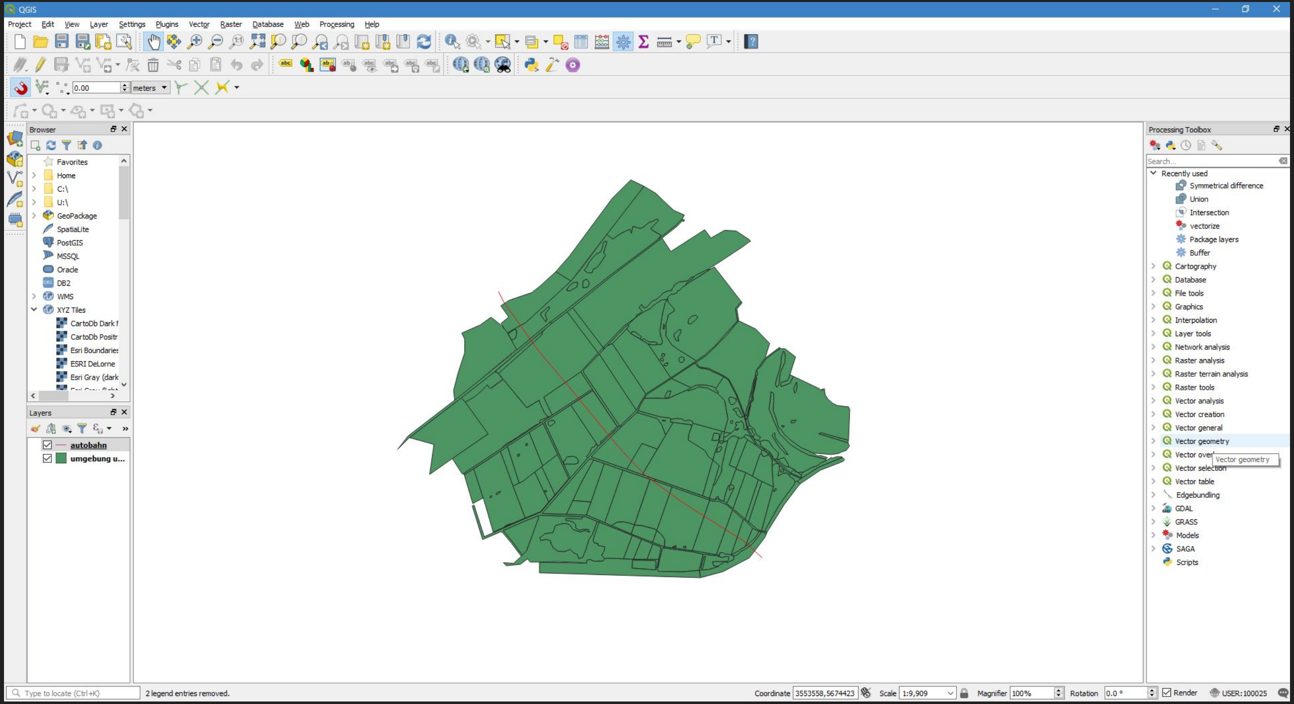
1. Umgebung.gpkg (= environment)
    - Environment polygonal vector layer
    - Attribute table that shows attributes like the type, usage, etc of habitats
2. Autobahn.gpkg (= expressway)
    - Line vector layer that shows the proposed area of autobahn to be constructed
    - Does not include the physical space the actual autobahn will take

Workflow Steps

1. Load the GeoPackage layers as vector layers on to QGIS
2. Represent the physical space of the proposed Autobahn with diameter 20m
3. The impact that the autobahn causes is bigger than its physical space. Represent these impact areas, of 100m and 300m
4. Unify the 3 impact areas to create one aggregated impact area with 3 different impact zones
5. Show the area in the environment that is actually in the impact zone
6. Show the impacted habitats that are actually endangered or protected by laws
7. Style the results and make it readable and easier to analyze

Step 1: Loading the Vector Layers

Load autobahn.gpkg and umgebung.gpkg on to QGIS

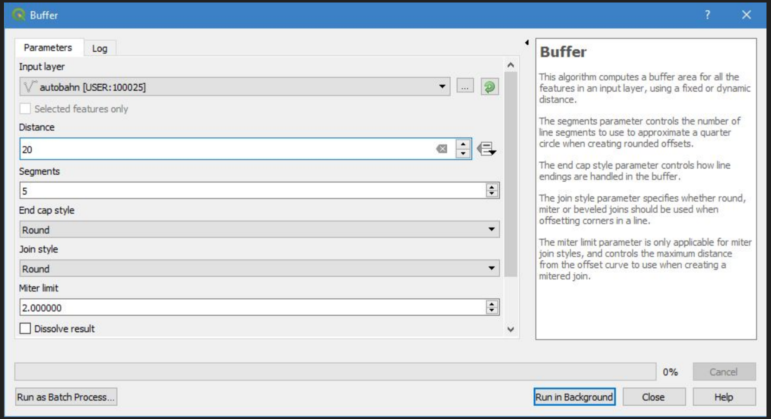


2 legend entries removed.

Coordinate 3553558,5674423 Scale 1:9,909 Magnifier 100% Rotation 0.0 ° ✓ Render USER:100025

Step 2: Representing the Autobahn

Create a buffer of 20m on the *autobahn* vector layer


Buffer

Parameters | Log

Input layer: autobahn [USER:100025]

Selected features only

Distance: 20

Segments: 5

End cap style: Round

Join style: Round

Miter limit: 2.000000

Dissolve result

Buffer

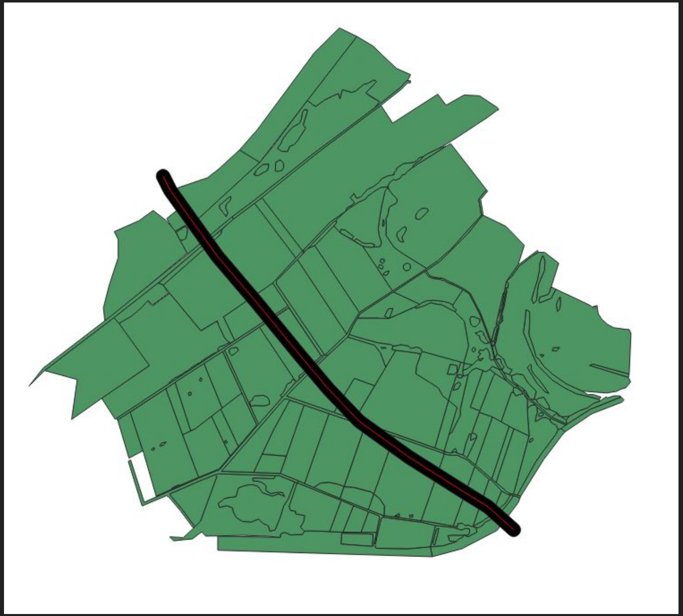
This algorithm computes a buffer area for all the features in an input layer, using a fixed or dynamic distance.

The segments parameter controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.

The end cap style parameter controls how line endings are handled in the buffer.

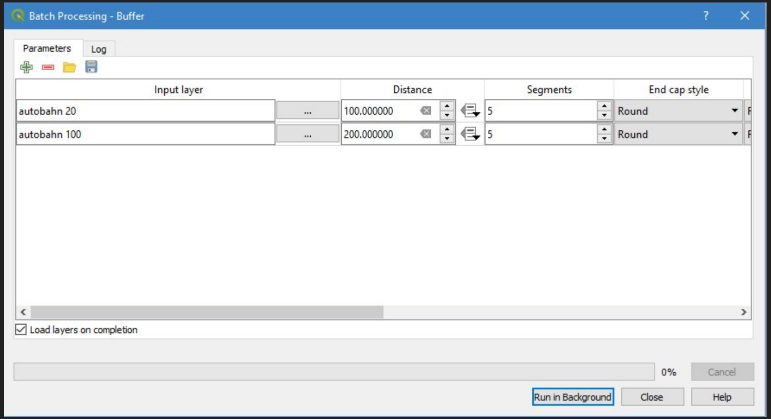
The join style parameter specifies whether round, miter or beveled joins should be used when offsetting corners in a line.

The miter limit parameter is only applicable for miter join styles, and controls the maximum distance from the offset curve to use when creating a mitered join.



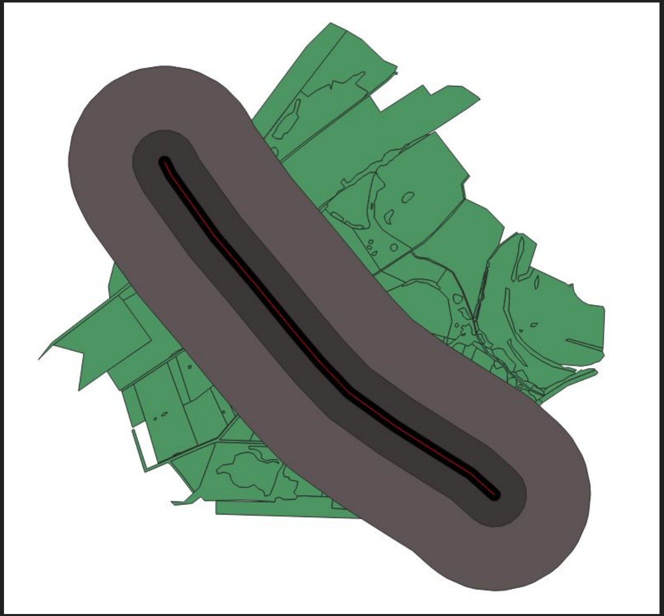
Step 3: Representing the Impact Areas

Create 2 more buffers from previous buffers for 100m and 300m



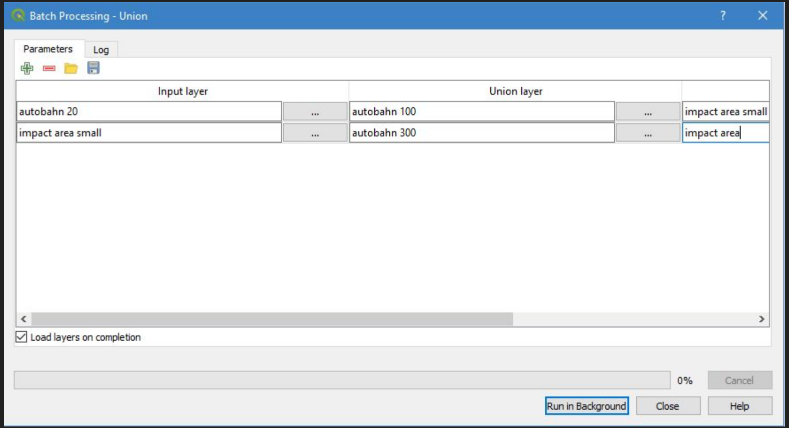
Batch Processing - Buffer

Input layer	Distance	Segments	End cap style
autobahn 20	100.000000	5	Round
autobahn 100	200.000000	5	Round



Step 4: Aggregating the Impact Areas

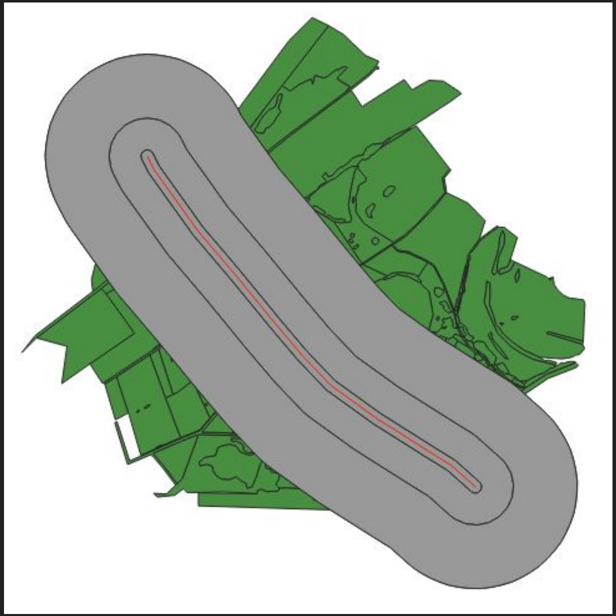
Perform a Union on the 3 buffers to create an overall Impact Area



Batch Processing - Union

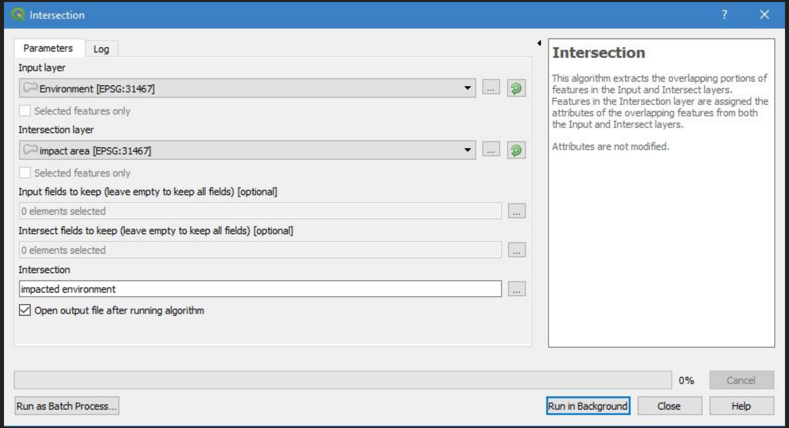
Input layer		Union layer		
autobahn 20	...	autobahn 100	...	impact area small
impact area small	...	autobahn 300	...	impact area

☑ Load layers on completion



Step 5: Show the Impacted Habitats

Perform a Intersection on the Impact Area and Environment


Intersection

Parameters | Log

Input layer: Environment [EPSG:31467]

Selected features only

Intersection layer: impact area [EPSG:31467]

Selected features only

Input fields to keep (leave empty to keep all fields) [optional]: 0 elements selected

Intersect fields to keep (leave empty to keep all fields) [optional]: 0 elements selected

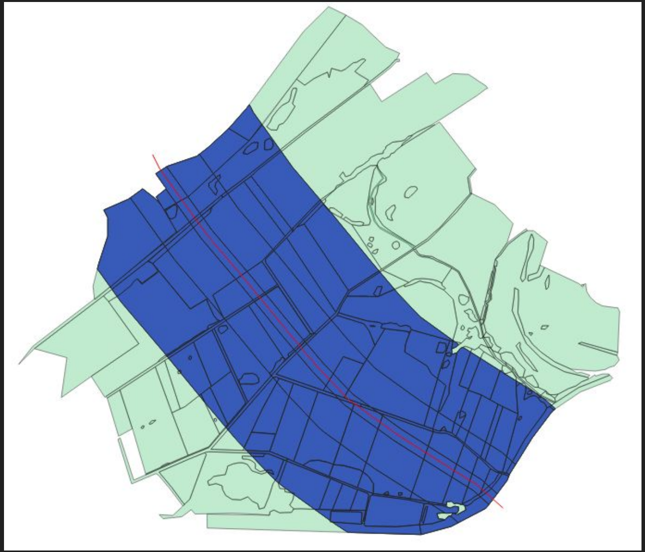
Intersection: impacted environment

☑ Open output file after running algorithm

Intersection

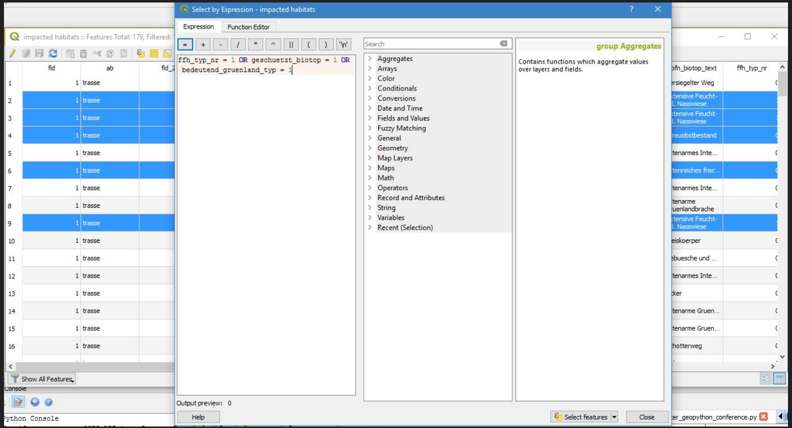
This algorithm extracts the overlapping portions of features in the Input and Intersect layers. Features in the Intersection layer are assigned the attributes of the overlapping features from both the Input and Intersect layers.

Attributes are not modified.



Step 6: Show the Protected Habitats

Query the features on the intersected habitats



Select by Expression - impacted habitats

impacted habitats :: Features Total: 179, Filtered:

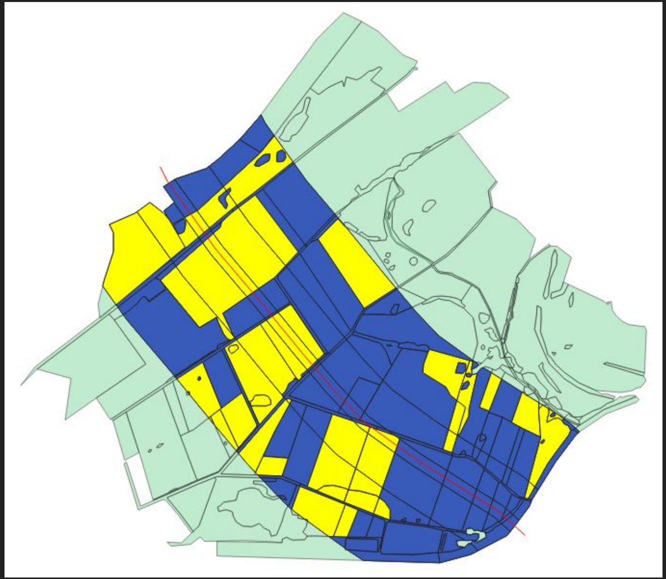
```
ffh_typ_nr = 1 OR geschuetzt_biotop = 1 OR
 bedeutend_gruenland_typ = 1
```

group Aggregates

Contains functions which aggregate values over layers and fields.

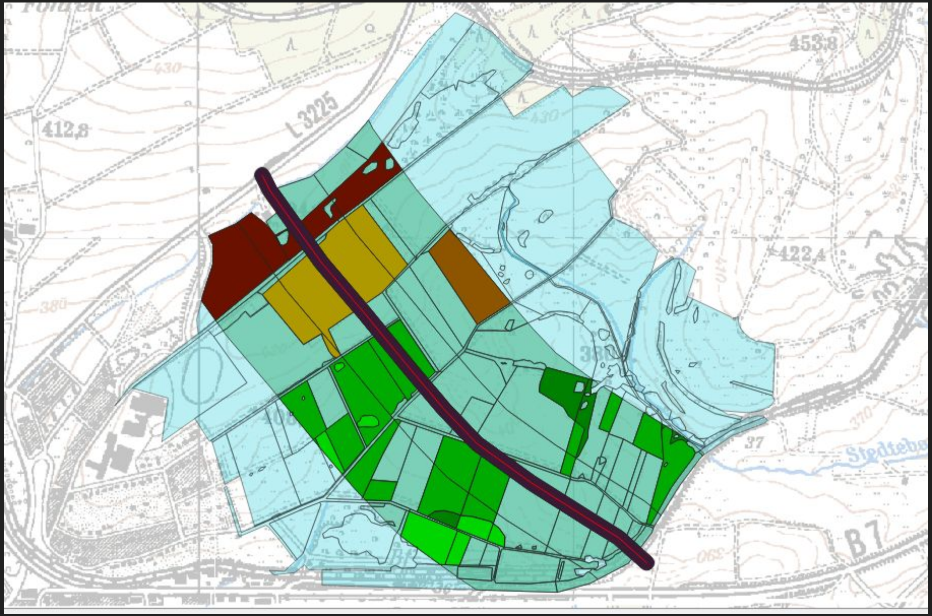
Output preview: 0

	fid	ab	fid_2	ofn_biotop_text	ffh_typ_nr
1	1	trasse		rsiegelter Weg	
2	1	trasse		tensive Feucht- l. Nasswiese	
3	1	trasse		tensive Feucht- l. Nasswiese	
4	1	trasse		reuobstbestand	
5	1	trasse		tenarmes Inte...	
6	1	trasse		tenreiches frisc...	
7	1	trasse		tenarmes Inte...	
8	1	trasse		tenarme uenlandbrache	
9	1	trasse		tensive Feucht- l. Nasswiese	
10	1	trasse		eiskoerper	
11	1	trasse		buesche und ...	
12	1	trasse		tenarmes Inte...	
13	1	trasse		ker	
14	1	trasse		tenarme Gruen...	
15	1	trasse		tenarme Gruen...	
16	1	trasse		hotterweg	



Step 7: Style the Results

Using the available styling functions, make the end results more readable and easier to understand

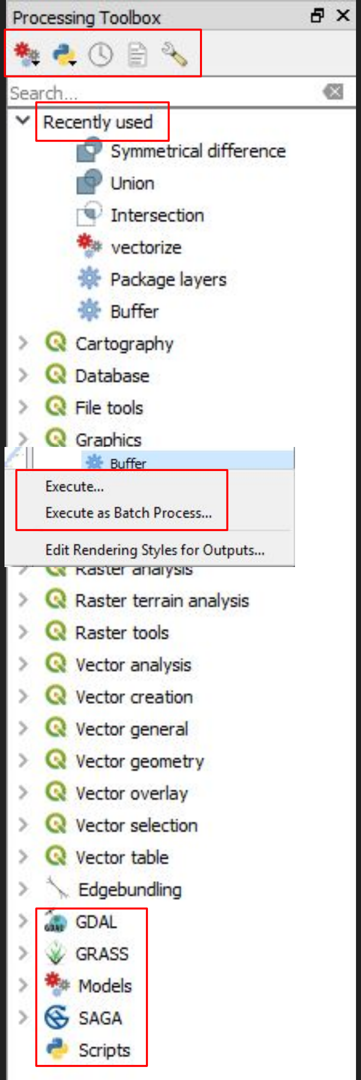


Processing Features

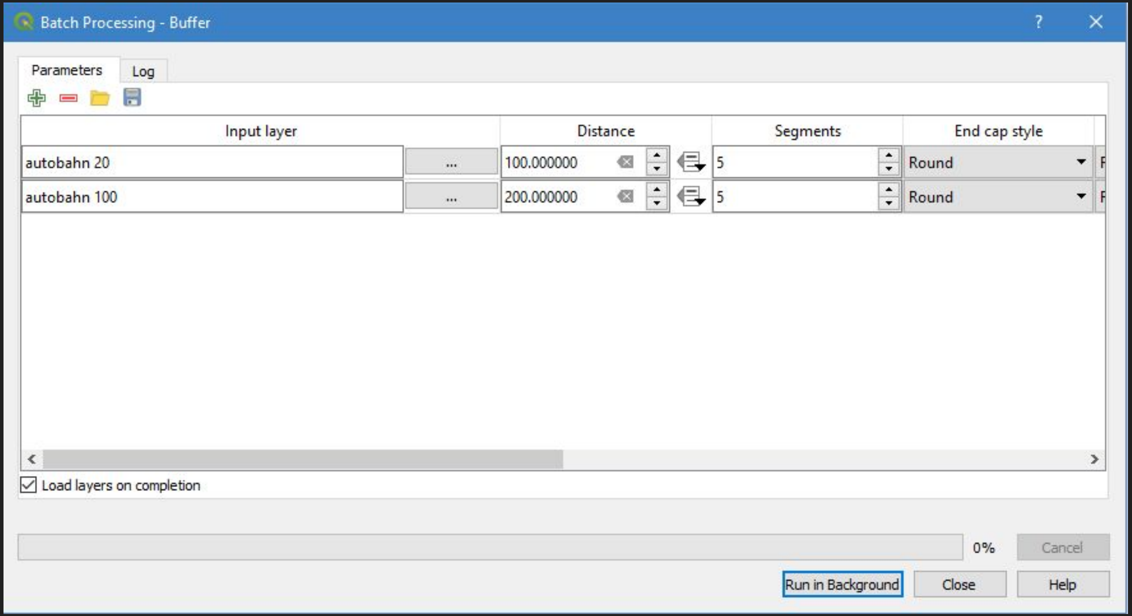
- Toolbox
- Batch Processing Interface
- Graphic Modeler
- History
- Script

Processing Feature - Toolbox

- Main element of the Processing GUI
- Lists all available algorithms grouped by their providers
- Access point to run these algorithms, be it single or batch processes
- Menu buttons at the top for *Graphic Modeler, Scripts, History, Options,* etc



Processing Feature -
Batch Processing Interface

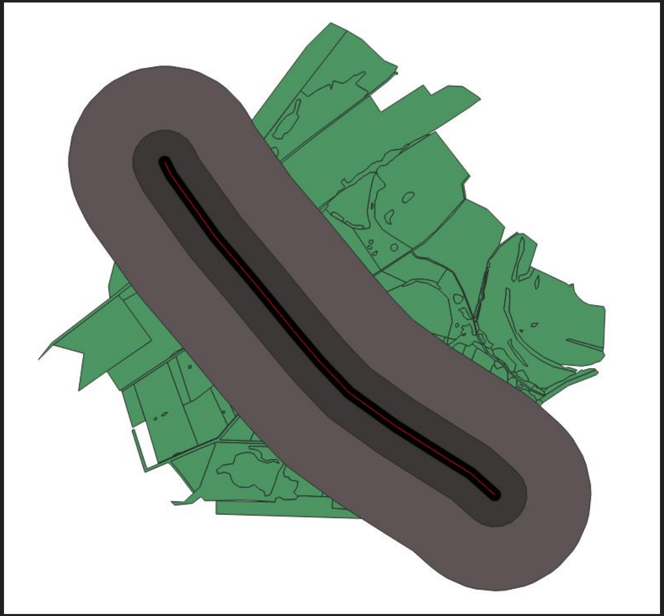

Batch Processing - Buffer

Parameters Log

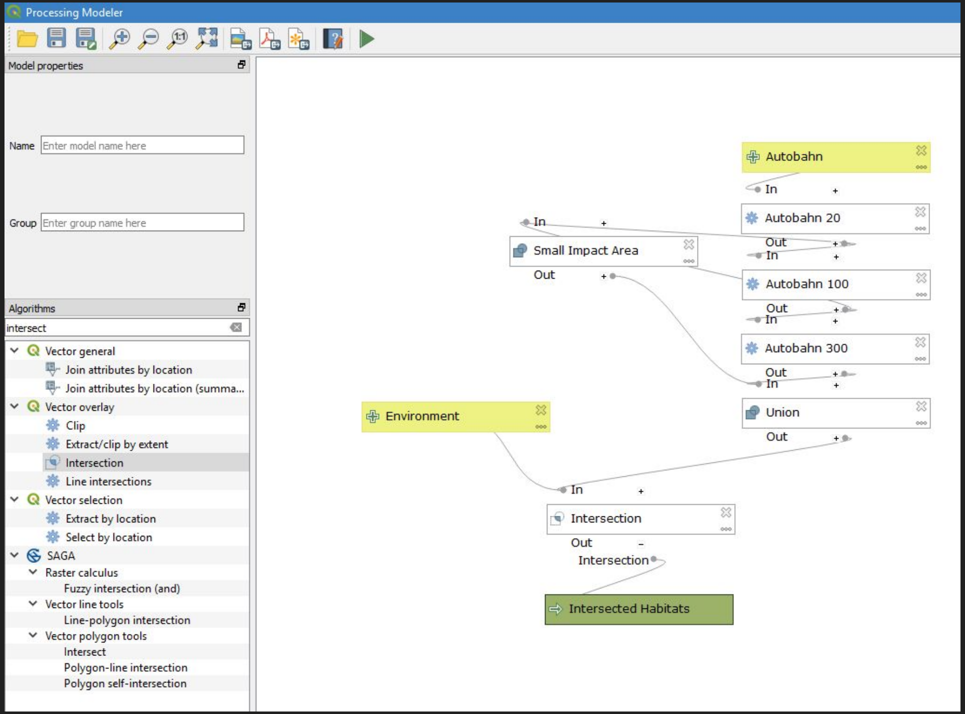
Input layer	Distance	Segments	End cap style
autobahn 20	100.000000	5	Round
autobahn 100	200.000000	5	Round

☑ Load layers on completion

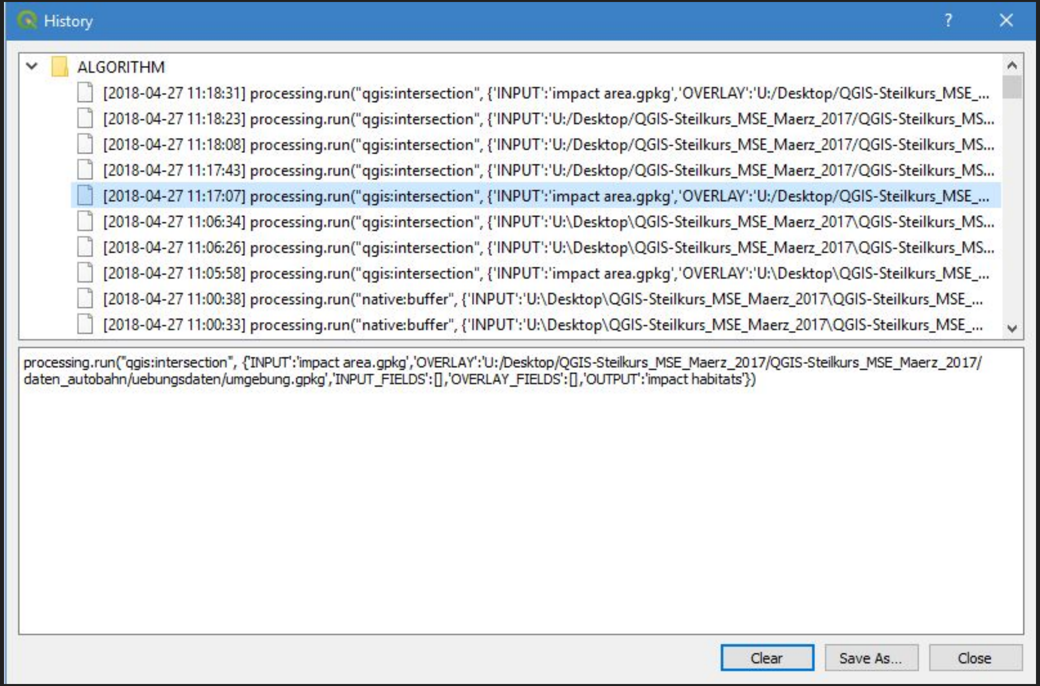
0%



Processing Feature - Graphic Modeler



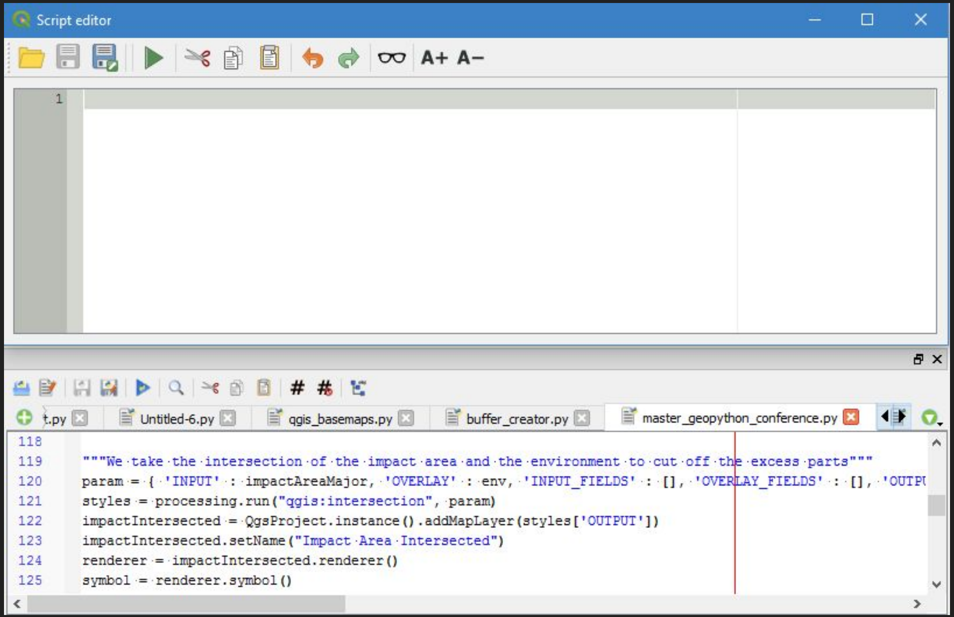
Processing Feature - History Tool



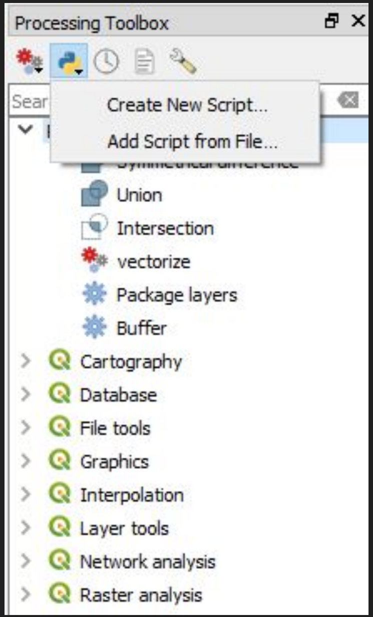
```
[2018-04-27 11:18:31] processing.run("qgis:intersection", {'INPUT':'impact area.gpkg','OVERLAY':'U:/Desktop/QGIS-Steilkurs_MSE_...
[2018-04-27 11:18:23] processing.run("qgis:intersection", {'INPUT':'U:/Desktop/QGIS-Steilkurs_MSE_Maerz_2017/QGIS-Steilkurs_MS...
[2018-04-27 11:18:08] processing.run("qgis:intersection", {'INPUT':'U:/Desktop/QGIS-Steilkurs_MSE_Maerz_2017/QGIS-Steilkurs_MS...
[2018-04-27 11:17:43] processing.run("qgis:intersection", {'INPUT':'U:/Desktop/QGIS-Steilkurs_MSE_Maerz_2017/QGIS-Steilkurs_MS...
[2018-04-27 11:17:07] processing.run("qgis:intersection", {'INPUT':'impact area.gpkg','OVERLAY':'U:/Desktop/QGIS-Steilkurs_MSE_...
[2018-04-27 11:06:34] processing.run("qgis:intersection", {'INPUT':'U:\Desktop\QGIS-Steilkurs_MSE_Maerz_2017\QGIS-Steilkurs_MS...
[2018-04-27 11:06:26] processing.run("qgis:intersection", {'INPUT':'U:\Desktop\QGIS-Steilkurs_MSE_Maerz_2017\QGIS-Steilkurs_MS...
[2018-04-27 11:05:58] processing.run("qgis:intersection", {'INPUT':'impact area.gpkg','OVERLAY':'U:\Desktop\QGIS-Steilkurs_MSE_...
[2018-04-27 11:00:38] processing.run("native:buffer", {'INPUT':'U:\Desktop\QGIS-Steilkurs_MSE_Maerz_2017\QGIS-Steilkurs_MSE_...
[2018-04-27 11:00:33] processing.run("native:buffer", {'INPUT':'U:\Desktop\QGIS-Steilkurs_MSE_Maerz_2017\QGIS-Steilkurs_MSE_...
```

```
processing.run("qgis:intersection", {'INPUT':'impact area.gpkg','OVERLAY':'U:/Desktop/QGIS-Steilkurs_MSE_Maerz_2017/QGIS-Steilkurs_MSE_Maerz_2017/
daten_autobahn/uebungsdaten/umgebung.gpkg','INPUT_FIELDS':[],'OVERLAY_FIELDS':[],'OUTPUT':'impact habitats'})
```

Processing Feature - Scripting Tool



```
"""We take the intersection of the impact area and the environment to cut off the excess parts"""
param = { 'INPUT' : impactAreaMajor, 'OVERLAY' : env, 'INPUT_FIELDS' : [], 'OVERLAY_FIELDS' : [], 'OUTPU
styles = processing.run("qgis:intersection", param)
impactIntersected = QgsProject.instance().addMapLayer(styles['OUTPUT'])
impactIntersected.setName("Impact Area Intersected")
renderer = impactIntersected.renderer()
symbol = renderer.symbol()
```



Processing -
Overview and Conclusion

- Certain geoanalytical tasks involve menial and repetitive steps
- Processing is very nifty as it automates entire processes
  - Example: Your agency wants to build an autobahn whereby the autobahn construction would have the least impact on the environment and protected habitats nearby
  - Solution: Run the script/algorithm through batch process with different inputs and then later access the end results
- We also learnt there are some limitations of Processing:
  - Inflexible parameters/outputs
  - No interactivity
  - Conversion of outputs is expensive

Scripts Demonstration

- Script 1: Script that automates the entire workflow
- Script 2: Interactive script
- Script 3: Flexible parameters (kind of)

Scripting in QGIS - PyQGIS and QGIS 0.9.0

- QGIS 0.9.0 introduced Python to its client
- PyQGIS or Python Console in QGIS client
- Features of PyQGIS:
  - Automatically run Python code when QGIS starts
  - Create custom applications with Python API
  - Run Python code and commands on the Python Console
  - Create and use Python plugins

Hands-on Exercises

Hands-on Exercises - Objectives

- Introduction to using QGIS
  - Loading vector layers
  - Some basic functions and nifty tips and tricks
  - Python Console
- Introduction to Processing in QGIS
  - Processing Toolbox
  - Batch Processing Interface
  - Graphic Modeler
  - Scripts

Getting Started - Prerequisites

- QGIS 3.0
- Connection to workshop GitHub repository (or a cached page)
  - *autobahn.gpkg* and *umgebung.gpkg* datasets
  - Problem tasks
  - Link to page

Task 1 -
Adding GeoPackage as Layers into QGIS

Task 2 -
Adding Buffers to Autobahn Layer

Task 3 -
Performing Union on the Buffer Areas

Task 4 -
Refining Code

Task 5 -
Selecting Features from Queries

Task 6 -
Styling and Cleaning Up

Bonus Task -
Interactive and Independent Script

Conclusion

- Processing Framework and QGIS
  - History
  - Features
  - Advantages, limitations
- Programming in Python
- Interest in QGIS and GIS
- Interest in Python and programming

Questions?

Questions?

Thank You!

Task 1.1. -

Manually adding the GeoPackage files into QGIS

1. Run QGIS 3.0
2. Set Project CRS to 31467, DHDN/Gauss-Kruger Zone 3
3. You can do this manually by clicking on the CRS at the bottom-right of the window, or by clicking *Project -> Project Properties -> CRS* or you can press *Ctrl + Shift + P* to open up Project Properties and then clicking CRS
4. From there, change the CRS to *DHDN/Gauss-Kruger zone 3, EPSG: 31467*
5. Once you're done, check that it says 31467 at the bottom of the window on the CRS tab
6. Now on the browser panel, look for GeoPackage, right click it and select New Connection
7. Navigate to the folder you saved umgebung.gpkg in and add it
8. On the browser panel, show the child items of environment/umgebung.gpkg and drag the vector layer onto the map canvas

Task 1.2. -

Creating a Dialog Box to ask for User Input on File to Add

1. On the Menu Toolbar, click *Plugins -> Python Console* or press *Ctrl* + *Alt* + *P* on your keyboard to open up the Python Console
2. You can run Python code on the console to perform various tasks, try creating a file dialog box that asks for user input on the file path
3. ***envPath = QFileDialog.getOpenFileName(QFileDialog(), "Environment Layer Select", "$set_a_default_path$")[0]***
4. The [0] is because the above returns a list, and we only need the first value of it, which is the file path

Task 1.3. -

Adding Vector Layers into QGIS

1. *env = iface.addVectorLayer(envPath, '$pick_a_name$', 'ogr')*
2. If the layer name is saved as something else, you can change it with *env.setName("new_name")*
3. Practice and do the same for the Autobahn layer using the Python console

Task 2.1. -

Introduction and Running the Graphic Modeler

1. To start off, you need to open up your Processing Toolbox, on the menu toolbar, click *Processing -> Toolbox* or press *Ctrl + Alt + T* and see that the Processing Toolbox window now appears on the right side of the QGIS window
2. On the Toolbox's menu toolbar, click *Models -> Create New Model*
3. First, we need to visualize and get an idea of what we want to achieve (this helps us to form pseudo code before creating actual code in the future): Run through the Autobahn layer with a Buffer algorithm to create a new Autobahn layer that is 40m wide in diameter

Task 2.2. -

Create a 20m buffer file for the Autobahn layer using Graphic Modeler

1. Name the Model *Autobahn Buffer* and the Group *vector*
2. On the bottom left, click on Input if it is not already selected, and drag Vector Layer into the blank canvas
3. Name the parameter name *Autobahn* and under Geometry type, select Line as we only want it to exclusively deal with line geometries
4. Drag a Number under Input into the canvas, and name it *Buffer Distance*, you may choose to fill in the other fields
5. On the bottom left, click on Algorithms, and in the search bar, type 'Buffer', and drag the Algorithm called Buffer into the canvas, making sure it is under Vector Geometry and not any other algorithm provider

Task 2.2. -

Create a 20m buffer file for the Autobahn layer using Graphic Modeler

1. In the Input Layer field, select Autobahn from the dropdown menu, in the Distance field, type @bufferdistance, and in the Buffered field, type ***Output Layer Name***
2. On your canvas, you should see that ***Buffer Distance*** and ***Autobahn*** are connected as inputs to ***Buffer*** which gives an output named ***Output Layer Name***
3. On the menu toolbar, click on the green arrow Run Model or press F5 to run the model
4. Under Autobahn, select your Autobahn vector layer from the drop down menu, under Buffer Distance, type in 20 and under Output Layer Name, type *Autobahn 20* and run it
5. Let the Model run and after processing, you should see the output vector on your main QGIS window

Task 2.3. -

Recreating the same function using a standalone script

1. On the main QGIS window, at the Processing Toolbox, search for *Buffer*, this is the algorithm that we utilized in the Modeler
2. Double click on it and you can do essentially the same thing as we did in the modeler, except with a few extra fields that we set to default in the Modeler
3. Enter the fields for ***Input Layer, Distance and Buffered*** and run it
4. At the top of the window, click on Log, you will see a bunch of code, we will be needing this for our script
5. Study the Input parameters and copy its entire line of code
6. On the PyQGIS console, type ***atbn = QgsProject.instance().mapLayersByName('%NAMEOFYOURAUTOBAHN LAYER%')[0]***, this assigns the vector layer of your autobahn to the variable atbn

Task 2.3. -

Recreating the same function using a standalone script

1. On the PyQGIS console, type param = and paste the copied code, and edit some fields, it should look something like this:
   param = { 'INPUT' : atbn, 'DISTANCE' : 20, 'SEGMENTS' : 5, 'END_CAP_STYLE' : 0, 'JOIN_STYLE' : 0, 'MITER_LIMIT' : 2, 'DISSOLVE' : False, 'OUTPUT' : 'memory:' }
2. We changed the file paths or assigned them to variables to make it easier for us because the Python console cannot read Unicode characters like \
3. Now to add the output as a map layer, type:
   algoOutput = processing.run("qgis:buffer", param)
   autobahn20 = QgsProject.instance().addMapLayer(algoOutput['OUTPUT'])
   autobahn20.setName("Autobahn 20")

Task 2.4. -

Creating 2 more buffers

1. On the Processing Toolbox, press the Scripts button and select ***Create New Script***
2. It will open up a text editor where you can write your Pythonic scripts
3. Add the following as the header of your script:
   from qgis.core import QgsProject
   import processing
4. Write a script that does the following:
   1. ***Using Autobahn 20 and 100m as inputs, create a new buffer naemd Autobahn 100***
   2. Using Autobahn 100 and 200m as inputs, create a new buffer named Autobahn 300
   Hint: You might need to assign the variable `autobahn20` in the script, do this with: `autobahn20 =

Task 2
Running a Script (Console)

Task 3
Running a Script (Script Runner)

Task 4
Creating Custom Scripts

Bonus Content

Questions

Contact