Começando a Programar com o Python no QGIS

André Silva

GEOeduc

CAPÍTULO 4:

Execução de Ferramentas

CAPÍTULO 4: Execução de Ferramentas

A caixa de ferramentas de processamento no QGIS contém uma coleção cada vez maior de ferramentas de geoprocessamento. A caixa de ferramentas fornece uma interface fácil de processamento em lote para executar qualquer algoritmo em muitas entradas. Mas há casos em que você precisa incorporar um pouco da lógica personalizada no processamento em lote. Como todos os algoritmos de processamento podem ser executados programaticamente por meio da API do Python, você pode executá-los pelo console do Python. Neste capítulo iremos aprender a executar um algoritmo de processamento por meio do Python Console para executar uma tarefa de geoprocessamento personalizada em apenas algumas linhas de código.

Objetivos do capítulo:

- Introdução a processamento de ferramentas
- Execução de ferramentas vetoriais
- Reprojeção de camadas vetoriais

Introdução a Processamento de Ferramentas

O console permite que usuários aumentem seu rendimento e executem operações complexas que jamais podem ser executadas usando qualquer um dos outros elementos da GUI da estrutura de processamento. Modelos que usam vários algoritmos podem ser definidos usando a interface da linha de comandos e operações adicionais, conforme loops e sentenças condicionais, podem ser adicionadas para compor fluxos de trabalho mais interativos e dinâmicos.

Não há um console de processamento no QGIS, mas todos os comandos de processamento estão disponíveis no console Python interno do QGIS. Isso significa que você pode inserir esses comandos no script e conectar algoritmos de processamento a todos os outros recursos (incluindo métodos da API QGIS) disponíveis até então.

O código que você pode elaborar no console do Python, mesmo que jamais chame qualquer método de processamento particular, pode ser convertido em um novo algoritmo

que você pode invocar posteriormente na caixa de ferramentas, no modelador gráfico ou em algum outro componente, igualmente conforme você com qualquer outro algoritmo. Na realidade, alguns algoritmos que você pode descobrir na caixa de ferramentas são scripts simples.

Implementando linhas de comando para execução de ferramentas de geoprocessamento

Como já mencionado anteriormente é necessário importar as funções de processamento, como por exemplo: *import processing*.

Agora, há basicamente somente um acontecimento (curioso) que você pode realizar com isso no console: rodar um algoritmo. Isso é concluído usando o método *run*, que leva a denominação do algoritmo para executar com o seu primeiro parâmetro e, em seguida, um número variável de parâmetros adicionais, dependendo dos requisitos do algoritmo. Logo, a primeira coisa que você precisa compreender é o nome do algoritmo a ser executado. Esse não é o nome que você vê na caixa de ferramentas, mas uma denominação restrita da linha de comando.

Após de conhecer o nome da linha de comandos do algoritmo, a próxima coisa a elaborar é estabelecer a sintaxe correta para executá-lo. Isso significa em compreender quais parâmetros são necessários ao invocar o método run ().

Existe um método para relatar um algoritmo em detalhes, que pode ser usado para adquirir uma lista dos parâmetros que um algoritmo requer e das saídas que ele irá produzir. Para alcançar essas informações, você pode adotar o método algoritmoHelp (id_of_the_algorithm). Use o ID do algoritmo, jamais o nome descritivo terminado.

Invocando o método com native: buffer como parâmetro (qgis: buffer é um apelido para native: buffer também funcionará), você obtém a subsequente exposição:

```
>>> processing.algorithmHelp("native:buffer")
Buffer (native: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 round
ed 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 l
ine.

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.


----------------
Input parameters
----------------

INPUT: Camada de entrada

        Parameter type: QgsProcessingParameterFeatureSource

        Accepted data types:
                - str: layer ID
                - str: layer name
                - str: layer source
                - QgsProcessingFeatureSourceDefinition
                - QgsProperty
                - QgsVectorLayer
```

```
33
34 DISTANCE: Distância
35
36         Parameter type: QgsProcessingParameterDistance
37
38         Accepted data types:
39                 - int
40                 - float
41                 - QgsProperty
42
43 SEGMENTS: Segmentos
44
45         Parameter type: QgsProcessingParameterNumber
46
47         Accepted data types:
48                 - int
49                 - float
50                 - QgsProperty
51
52 END_CAP_STYLE: Estilo da cobertura do fim
53
54         Parameter type: QgsProcessingParameterEnum
55
56         Available values:
57                 - 0: Arredondado
58                 - 1: Plano
59                 - 2: Quadrado
60
61         Accepted data types:
62                 - int
63                 - str: as string representation of int, e.g. '1'
64                 - QgsProperty
65
66 JOIN_STYLE: Estilo da união
```

```
        Parameter type: QgsProcessingParameterEnum

        Available values:
                - 0: Arredondado
                - 1: Pontiagudo
                - 2: Chanfrado

        Accepted data types:
                - int
                - str: as string representation of int, e.g. '1'
                - QgsProperty

MITER_LIMIT: Limite do mitre

        Parameter type: QgsProcessingParameterNumber

        Accepted data types:
                - int
                - float
                - QgsProperty

DISSOLVE: Dissolver Resultado

        Parameter type: QgsProcessingParameterBoolean

        Accepted data types:
                - bool
                - int
                - str
                - QgsProperty

OUTPUT: Buffered

        Parameter type: QgsProcessingParameterFeatureSink

        Accepted data types:
                - str: destination vector file, e.g. 'd:/test.shp'
                - str: 'memory:' to store result in temporary memory layer
                - str: using vector provider ID prefix and destination URI, e.g. 'postgres:...' to store result in PostGIS table
                - QgsProcessingOutputLayerDefinition
                - QgsProperty

----------------
Outputs
----------------

OUTPUT:  <QgsProcessingOutputVectorLayer>
        Buffered
```

Agora você tem tudo o que precisa para rodar qualquer algoritmo. conforme já mencionamos, os algoritmos podem ser executados usando: run (). Sua sintaxe é a seguinte:

>>> processing.run(name_of_the_algorithm, parameters)

Onde parâmetros é um dicionário de parâmetros que dependem do algoritmo que você deseja executar e é exatamente a lista que o método algoritmHelp () fornece.

Se um parâmetro for opcional e você não quiser usá-lo, jamais o inclua no dicionário.

Se um parâmetro não for especificado, o valor padrão será usado.

EXERCÍCIO 1: SCRIPTS DE PROCESSAMENTO DE DADOS GEOGRÁFICOS

Neste conjunto de exercícios você aprenderá na prática, como criar e executar scripts de geoprocessamento, mais comumente usados no dia a dia dos analistas e desenvolvedores GIS.

Passo 1 – Criação de Buffers

1) Com o Python Console aberto, arraste a camada MG_linhas_transmissao.shp para a área de trabalho do QGIS.

2) Com o Python Console aberto, crie uma variável chamada *layerName* e atribua ao valor dela o nome que está inserida no QGIS

3) Defina o local de saída e o nome do arquivo buffer, atribuindo a uma variável *outFile.*

Obs: não esqueça de colocar .shp no final da nomenclatura do arquivo

4) Defina a distância do buffer, atribuindo o valor 0.001, à variável *bufDist.*

```
bufDist = 0.001
```

5) Nesta etapa use o método QgsProject para ler o layer de interesse, usando também o índice para leitura do layer.

```
layers = QgsProject.instance().mapLayersByName(layerName)
layer = layers[0]
fields = layer.fields()
feats = layer.getFeatures()
```

6) Agora, iremos criar um QgsVectorFileWriter que adicionará as features com buffer a uma nova camada e irá salvar em um local especificado. Os parâmetros, em ordem, são o caminho do arquivo de saída, a codificação do arquivo, os campos da nova camada (QgsFields; nesse caso, idênticos aos campos da camada de entrada), o tipo de geometria (QgsWkbTypes), a definição do sistema de referência de coordenadas (QgsCoordinateReferenceSystem).

```
writer = QgsVectorFileWriter(outFile, 'UTF-8', fields, \
QgsWkbTypes.Polygon, layer.sourceCrs(), 'ESRI Shapefile')
```

7) Depois que o writer é criado, percorra as features na camada de entrada. Para cada feature, a leitura da geometria será feita. Em seguida, use o método buffer para criar a geometria com buffer. O buffer receberá dois argumentos; a distância do buffer e o número de segmentos de linha usados para formar um quarto de círculo (definido como 5). Agora defina a geometria da feature como a geometria do buffer recém-criada. A variável feat agora contém todos os atributos da camada de entrada, mas possui uma geometria diferente (polígono em vez de linha ou ponto). Use o writer para adicionar a feature (addFeature) à nova camada (buffer).

```
for feat in feats:
    geom = feat.geometry()
    buffer = geom.buffer(bufDist, 5)
    feat.setGeometry(buffer)
    writer.addFeature(feat)
```

8) Por fim adicione o layer com o buffer a interface do QGIS e exclua o writer.

```
iface.addVectorLayer(outFile, '', 'ogr')
del(writer)
```

Passo 2 – Recortando Vetores

É muito simples recortar camadas vetoriais com o PyQGIS. Este tutorial demonstrará como usar o Python no QGIS para cortar uma camada de linha com uma camada de polígono.

1) Importe o módulo *processing,* para que posteriormente possa usar métodos de processamento.

```
import processing
```

2) Defina os arquivos de entrada e saída. Defina o arquivo que será usado como cortador, a feição que será cortada e o arquivo de saída resultante. Para isso defina consecutivamente as seguintes variáveis *polyPath*, *linePath, clipPath.*

Obs: Insira em seu QGIS a camada MG_linhas_transmissao.shp e o arquivo Cliper.shp, você irá recortar a linha de transmissão.

3) Agora vamos chamar o método processing.run (), que será nossa ferramenta de corte. Deve-se informar o nome da ferramenta de processamento, em nosso caso será a ferramenta de clipe nativa QGIS, e um dicionário que contém os parâmetros de entrada para a ferramenta. Para a ferramenta de clipe, os únicos parâmetros são os caminhos de arquivo que definimos na primeira etapa. Esse bloco de código executará a análise e salvará as features cortados em um novo shapefile.

```
processing.run("native:clip", {'INPUT':linePath,\
'OVERLAY':polyPath,\
'OUTPUT':clipPath})
```

4) Adicione o vetor na interface do Qgis com o comando.

```
iface.addVectorLayer(clipPath, '', 'ogr')
```

Passo 3 – Dissolvendo vetores

O processo de dissolver nada mais é do que agregar diversos polígonos em uma só feição. Agora iremos verificar como isso pode ser feito usando scripts Python.

1) Retorne a camada buffer.shp para a interface do QGIS e defina a variável *infn*, com o valor do caminho do buffer.shp

2) Defina uma variável *outfn*, com o valor de saída do seu arquivo, para o arquivo de saída defina o nome *buffer_dissolve_all.shp*.
Obs: Não esqueça ao declarar o caminho do local do arquivo use a barra (/)

3) Insira o método processing.run() para executar o dissolve, defina a opção “native:dissolve”, para execução da ferramenta. Defina os parâmetros de entrada e saída como sendo os definidos em passos anteriores.

processing.run(“native:dissolve”, {‘INPUT’:infn, ‘OUTPUT’:outfn})

4) Adicione a camada resultante na interface do Qgis.

iface.addVectorLayer(outfn, ‘’, ‘ogr’)

Passo 4 – Reprojeção vetorial

Complementando nosso aprendizado sobre execução de ferramentas de geoprocessamento, iremos verificar como realizar a reprojeção de uma camada vetorial, fazendo o uso de um script em python. Não se preocupe este procedimento também é muito simples, e usaremos métodos e classes já conhecidos.

1) Novamente adicione a camada buffer adicionada ao QGIS, abra seu editor e faça a importação da biblioteca necessária:
Import processing

2) Ative o layer de interesse para que possa fazer a reprojeção, crie uma variável chamada *lyr*, e adicione a ela o valor do método: iface.activeLayer()

3) Crie um dicionário chamado *parameter* e insira o seguinte valor para entrada (INPUT): lyr, coordenada(TARGET_CRS): ‘EPSG:3814’, para saída defina o local da saída do arquivo juntamente com seu nome (OUTPUT).

parameter = {'INPUT': lyr, 'TARGET_CRS': 'EPSG:3814', 'OUTPUT': memory: Reprojected’}

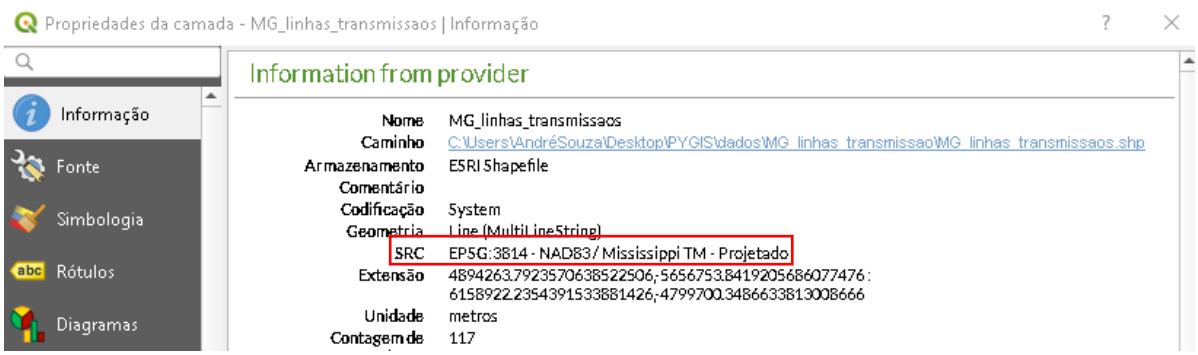
4) Adicione uma variável *result*, e adicione a ela um método de processamento, passando como parâmetro a variável *parameter.*

result = processing.run('native:reprojectlayer', parameter)

5) Por fim adicione uma classe chamada QgsProject para adicionar o layer a interface do QGIS. Não se esqueça de passar como parâmetro o local de gravação do arquivo e também o nome dele.

QgsProject.instance().addMapLayer(result[‘OUTPUT’])

6) Verifique se o arquivo resultante foi criado, e adicione o ao QGIS, depois verifique se houve a reprojeção corretamente.



Parabéns! Chegamos ao final de mais capítulo, no qual você aprendeu sobre execução de ferramentas usando scripts no PYQGIS.