Começando a Programar com o Python no QGIS

André Silva

GEOeduc

CAPÍTULO 5:
Automatização de Processos

CAPÍTULO 5: Automatização de Processos

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 um grande número de 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:

- Comparação com rotinas de processamento com visualização gráfica
- Scripts de criação de layout
- Trabalhando com imagens raster no PyQGIS

Rotinas com uso de scripts Python X Rotinas com o uso de visualização gráfica

Por que é interessante usar rotinas com script Python? Como já falamos durante o curso, os casos de uso da linguagem são variados, possui suporte a diversas bibliotecas para geotecnologia, com diversas classes e métodos diferentes para promover scripts para automatização de tarefas, suporte aos principais bancos de dados existentes e uma vesta coleção de módulos que podem ser embutios nos programas desenvolvidos.

A linguagem traz em si a simplicidade na maneira em que ela é escrita, facilitando assim o seu aprendizado e sendo muito utilizada em ambientes de estudo de programação, além de trazer o suporte a orientação a objetos, em outras palavras suportar programação e abstração de elementos do mundo real.

Uma das abordagens mais interessantes da linguagem é a utilização para automação de tarefas, onde os scripts desenvolvidos se encarregam das tarefas repetitivas, proporcionando aos analistas e profissionais GIS disponibilidade para atividades que demandem um acompanhamento mais direto.

Veja abaixo uma tabela explicativa mostrando as vantagens e desvantagens da operações que podem ser realizadas com o QGIS:

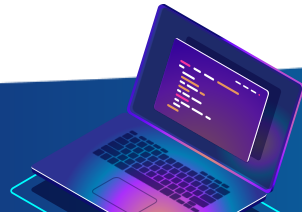
Uso de Scripts de programação (PYQGIS)		**Uso de interface gráfica (QGIS)**	
Vantagens	**Desvantagens**	**Vantagens**	**Desvantagens**
Rápido processamento	Um tempo um pouco maior para aprendizagem da sintaxe	Menor tempo para aprendizagem	Processamento mais demorado
Menor consumo de memória de máquina	É visualmente menos interativo.	Rotinas que demandam apenas o uso de cliques e não de escrita	Maior consumo da memória de máquina

Execução em apenas 1 clique		É visualmente mais interativo.	Execução com várias sequências e cliques
Reuso do script			Tarefas são manualmente repetitivas
Facilidade de compartilhamento com outras pessoas			Dificuldade de compartilhamento de processos
Desenvolvimento do pensamento lógico			Tarefas são realizadas sempre com a presença humana
Possibilidade de trabalhar com agendamentos de execução			Impossibilidade de trabalhar com agendamentos de execução
Automatizar tarefas repetitivas			
Melhor aproveitamento do tempo para outras atividades			Prejudica a gestão do uso do tempo.

Trabalhando com dados raster

O QGIS usa a biblioteca GDAL para ler e gravar formatos de dados raster, incluindo ArcInfo Binary Grid, ArcInfo ASCII Grid, GeoTIFF, ERDAS IMAGINE e muitos outros.

Os dados raster no GIS são matrizes de células discretas que representam recursos acima ou abaixo da superfície terrestre. Cada célula na grade raster é de idêntico tamanho e as células comumente são retangulares. Os conjuntos de dados raster típicos incluem dados de sensoriamento remoto, também fotografia aérea ou imagens de satélite e dados modelados, conforme uma matriz de elevação. Diferentemente dos dados vetoriais, os dados rasterizados



habitualmente não têm um registro de banco de dados agregado para cada célula. Eles são geocodificados pela resolução de pixels e pela coordenada x / y de um pixel de quina da camada raster. Isso permite que o QGIS posicione os dados corretamente na tela do mapa.

O QGIS utiliza informações de georreferência anexo da camada raster (por exemplo, GeoTiff) ou em um arquivo mundial adequado para expor os dados corretamente.

Dica: No link a seguir você pode ver mais de 100 formatos raster que podem ser trabalhados na biblioteca GDAL: http://www.gdal.org/formats_list.html

Raster em Python no QGIS

Arquivos raster são compostos por bandas, e esta composição pode ser realizada por uma ou mais bandas, isto geralmente depende do sensor e também da finalidade da imagem a ser coletada.

Rasters com uma única banda representam tipicamente variáveis contínuas (por exemplo, elevação) ou variáveis discretas (por exemplo, uso da terra). Em alguns casos, uma camada raster vem com uma paleta e os valores raster referem-se às cores armazenadas na paleta.

O código a seguir assume que rasterlayer é um objeto QgsRasterLayer.

```
rasterlayer = QgsProject.instance().mapLayersByName('srtm')[0]
print(rasterlayer.width(), rasterlayer.height())
```

```
919 619
```

```
# obtém a extensão do layer
print(rlayer.extent())
```

```
<QgsRectangle: 20.06856808199999875 -34.27001076999999896, 20.83945284300000012 -
33.75077500700000144>
```

```
 # verifica a contagem total das banda do raster
print(rlayer.bandCount())
```

```
1
```

Renderizadores: Quando uma camada raster é inserida, ela obtém um renderizador padrão com base em seu tipo.

Para realizar a consulta do renderizador atual:

```
print(rlayer.renderer())

<qgis._core.QgsSingleBandGrayRenderer object at 0x7f471c1da8a0>

print(rlayer.renderer().type())

singlebandgray
```

Para definir um renderizador, use o método setRenderer de QgsRasterLayer. Há várias classes de renderizador (derivadas de QgsRasterRenderer):

- QgsMultiBandColorRenderer
- QgsPalettedRasterRenderer
- QgsSingleBandColorDataRenderer
- QgsSingleBandGrayRenderer
- QgsSingleBandPseudoColorRenderer

Os arquivos raster com uma única banda podem ser desenhadas nas cores cinza (valores baixos = preto, valores altos = branco) ou com um algoritmo de pseudo-cor que atribui cores aos valores. Rasters com uma única banda podem ser desenhados usando a paleta de cores.

Veja o exemplo, queremos uma camada raster com cores que variam de verde a amarelo (correspondendo a valores de pixel de 0 a 255). No primeiro estágio, prepararemos um objeto QgsRasterShader e configuraremos sua função shader:

```
fcn = QgsColorRampShader()

fcn.setColorRampType(QgsColorRampShader.Interpolated)

lst = [ QgsColorRampShader.ColorRampItem(0, QColor(0,255,0)),
        QgsColorRampShader.ColorRampItem(255, QColor(255,255,0)) ]

fcn.setColorRampItemList(lst)

shader = QgsRasterShader()

shader.setRasterShaderFunction(fcn)
```

E você poderia se perguntar e como seria com raster com múltiplas bandas?

O QGIS mapeia as três primeiras bandas para vermelho, verde e azul para criar uma imagem colorida (esse é o estilo de desenho MultiBandColor). Em alguns casos, você pode substituir essas configurações. O código a seguir troca a faixa vermelha (1) e verde banda (2):

```
rlayer_multi = QgsProject.instance().mapLayersByName('multiband')[0]

rlayer_multi.renderer().setGreenBand(1)

rlayer_multi.renderer().setRedBand(2)
```

No caso de apenas uma banda ser necessária para a visualização da camada raster, o desenho de banda única pode ser escolhido, tanto em níveis de cinza quanto em falsa cor.
É necessário usar triggerRepaint para atualizar o mapa e ver o resultado:

```
rlayer_multi.triggerRepaint()
```

Por fim vamos falar um pouco de consulta de valores.

Os valores de um raster podem ser consultados usando o método de amostra da classe QgsRasterDataProvider. Você precisa especificar um QgsPointXY e o número da banda da camada raster que deseja consultar. O método retorna uma tupla com o valor e True ou False, dependendo dos resultados:

```
val, res = rlayer.dataProvider().sample(QgsPointXY(20.50, -34), 1)
```

Outro método que pode ser usado para consultar valores de um raster é usar o método de identificação que retorna um objeto QgsRasterIdentifyResult.

```
ident = rlayer.dataProvider().identify(QgsPointXY(20.5, -34), QgsRaster.IdentifyFormatValue)

if ident.isValid():
    print(ident.results())
```

```
{1: 323.0}
```

Nesse caso, é retornado pelo método results, um dicionário com índices de banda como chaves e valores de banda como valores. Por exemplo, algo como {1: 323.0}.

EXERCÍCIO 1: CARREGANDO E SIMBOLIZANDO RASTERS

Essa prática tem o intuito de mostrar algumas práticas e procedimentos com dados raster, proporcionando ao usuário o entendimento do conceito dos valores de pixels de uma imagem. Comumente na interface do QGIS, existem diversas ferramentas e procedimentos que possibilitam o trabalho com o imagens raster, agora nos exercícios abaixo, iremos abordar como criar uma escala de cores para melhor representação de pixels de um arquivo raster, possibilitando assim uma melhor análise. Usaremos também uma biblioteca GDAL, conhecida como uma biblioteca poderosa para o trabalho com imagens raster dentro do QGIS e por fim como criar um layout de impressão totalmente automatizado fazendo uso de scripts python.

Passo 1 – Adicione a camada raster ao seu projeto

1) Abra o QGIS, mais precisamente o console Python e crie uma variável chamada *fn,* e adicione a ele o valor do raster em questão que iremos trabalhar MDE.tif.

2) Carregue o raster para dentro da interface do QGIS, mais uma vez usaremos o método *iface,* adicionando o nome que você deseja usar para a camada adicionada.

```
rlayer = iface.addRasterLayer(fn, 'layer name')
```

Passo 2 – Altere a simbologia do raster

1) Vamos adquirir as estatísticas da banda do raster para podermos acessar adequadamente uma graduação de cores. O número 1 é banda do raster através da qual iremos obter as estatísticas. Insira o trecho abaixo:

```
stats = rlayer.dataProvider().bandStatistics(1, QgsRasterBandStats.All)
```

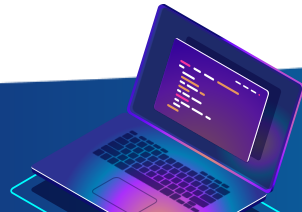
2) Vamos garantir o acesso aos valores mínimos e máximo dos pixels.

```
min = stats.minimumValue
max = stats.maximumValue
```

Passo 3 – Criação de uma escala de cores

A classe QgsColorRampShader definirá a nova escala de cores criada para simbolizar o raster. Primeiro, criaremos e esvaziaremos o sombreamento de cores. Também precisamos definir o tipo de rampa de cores que queremos usar. As opções são Interpoladas, Discretas ou Exatas.

- Interpolado estica cores em uma faixa de valores
- Discreto fornece todos os valores em um intervalo da mesma cor
- Exato fornece a cada valor de pixel exclusivo cores únicas



1) Abaixo usaremos os valores interpolados, pois o dados que estamos usando, representam elevações e são contínuos.

```
fnc = QgsColorRampShader()
fnc.setColorRampType(QgsColorRampShader.Interpolated)
```

2) Agora vamos definir o esquema de cores para o método QgsColorRampShader. Criaremos uma escala de cores muito simples que interpola de verde (valores baixos) para amarelo (valores altos).

```
lst = [QgsColorRampShader.ColorRampItem(min, QColor(0,255,0)),\
 QgsColorRampShader.ColorRampItem(max, QColor(255,255,0))]
fnc.setColorRampItemList(lst)
```

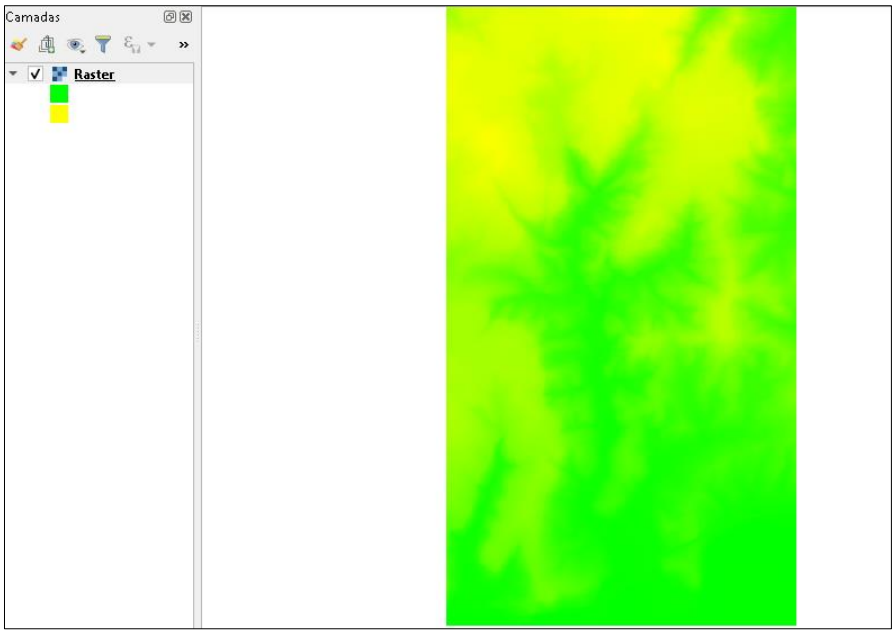
3) Atribuiremos a escala de cores a um QgsRasterShader para que possa ser usada para simbolizar uma camada raster.

```
shader = QgsRasterShader()
shader.setRasterShaderFunction(fnc)
```

4) Para concluirmos, necessitamos aplicar a simbologia que foi criada na camada raster. Antes de tudo criaremos um renderizador o shader raster, em seguida será adicionado o renderizador a camada raster.

```
renderer = QgsSingleBandPseudoColorRenderer(rlayer.dataProvider(), 1, shader)
rlayer.setRenderer(renderer)
```

5) Verifique se o resultado abaixo é similar ao da imagem abaixo:



Importante: Os códigos usados nos exercícios, foram testados exclusivamente na versão 3.X do QGIS, infelizmente o uso em versões anteriores, vai demandar alteração nos métodos e classes usadas.

EXERCÍCIO 2: OBTENÇÃO DE DADOS RASTER COM GDAL

O PyQGIS possui alguns métodos para acessar valores únicos em um raster, mas a melhor maneira de acessar dados raster é usando a Geospatial Data Abstraction Library, ou GDAL. O GDAL vem pré-instalado no intérprete QGIS Python, portanto, se você acessar o intérprete Python no QGIS (Plugins> Console Python), não precisará instalar nenhum pacote.

Passo 1 – Adicione a camada raster ao seu projeto

1) Primeiramente importe a biblioteca GDAL.

```
from osgeo import gdal
```

2) Agora iremos obter informações para a camada com a qual iremos trabalhar. Abaixo usaremos um trecho que irá obter a informação do primeiro layer de aba Table of Contents chamada MDE.tif.

```
layers = QgsProject.instance().mapLayersByName('layer name')
layer = layers[0]
```

3) O trecho abaixo, nos mostrará como obter o nome da camada e abrir este arquivo com um datasource GDAL.

```
ds = gdal.Open(layer.dataProvider().dataSourceUri())
```

4) Agora nós podemos selecionar uma band raster e fazer a leitura dos dados desta banda como uma matriz numérica 2D.

```
dem_arr = ds.GetRasterBand(1).ReadAsArray()
```

5) Por fim acesse os dados da matriz com a indexação numpy. O código abaixo imprime o valor da banda raster na primeira linha e na primeira coluna.

```
print(dem_arr[0][0])
```

EXERCÍCIO 3: TRABALHANDO COM LAYOUT DE MAPAS COM PYTHON

Junto com a API PyQGIS, você pode adotar o Python no QGIS para automatizar a geração de layouts de mapa. Há duas razões pelas quais isso é vantajoso. Primeiro, após de redigir o código inicial para caracterizar o mapa, você pode modificar a simbologia ou as camadas do mapa e replicar exatamente o layout. Segundo, após desenvolver o script, você pode desenvolver um mapa com o clique de um botão. Você só precisa desenvolver um layout uma vez e não precisa se preocupar em elaborar alterações nele acidentalmente.

Passo 1 – Adicione a camada raster ao seu projeto

1) Iniciaremos importando a biblioteca qgis.PyQt e também a QtGui.

```
from qgis.PyQt import QtGui
```

2) Vamos agora informar qual layer nos queremos trabalhar e qual posição este layer está no Table of Contents do QGIS.

```
layers = QgsProject.instance().mapLayersByName('stream_order')
layer = layers[0]
```

3) Agora vamos criar um novo layout dentro da interface do QGIS, chamado de *layout 1*, adicione o trecho abaixo em seu código:

```
project = QgsProject.instance()
manager = project.layoutManager()
layoutName = 'Layout1'
layouts_list = manager.printLayouts()
```

4) Vamos agora criar uma interação para verificar se existe algum layout com mesmo nome já adicionado em minha interface do QGIS, se houver alguma interface repetida, a mesma será excluída.

```
for layout in layouts_list:
    if layout.name() == layoutName:
        manager.removeLayout(layout)
layout = QgsPrintLayout(project)
layout.initializeDefaults()
layout.setName(layoutName)
manager.addLayout(layout)
```

5) Não menos importante, iremos criar itens dentro do nosso layout de mapa, como por exemplo escalas, título, legenda e entre outros. Adicione o código abaixo.

```
map = QgsLayoutItemMap(layout)
map.setRect(20, 20, 20, 20)
```

Dica: A cada passo completado, execute o script e vá até Projetos > Layouts > Layouts 1, na interface do Qgis e verifique as alterações.

6) Durante este passo, iremos definir a extensão do mapa que irá aparecer em nosso layout.

```
s = QgsMapSettings()
ms.setLayers([layer]) # set layers to be mapped
rect = QgsRectangle(ms.fullExtent())
rect.scale(1.0)
ms.setExtent(rect)
map.setExtent(rect)
```

```
map.setBackgroundColor(QColor(255, 255, 255, 0))
layout.addLayoutItem(map)

map.attemptMove(QgsLayoutPoint(5, 20, QgsUnitTypes.LayoutMillimeters))
map.attemptResize(QgsLayoutSize(180, 180, QgsUnitTypes.LayoutMillimeters))
```

Dica: Sempre que necessário, faça uma pesquisa para compreender corretamente a funcionalidade de uma determinada classe ou método, abuse dos buscadores da internet para verificar estas funcionalidades.

7) Agora vamos adicionar um item importante em nosso mapa, a barra de escala, definindo tamanho, estilo, fonte e outras caraterísticas pertinentes.

```
scalebar = QgsLayoutItemScaleBar(layout)
scalebar.setStyle('Line Ticks Up')
scalebar.setUnits(QgsUnitTypes.DistanceKilometers)
scalebar.setNumberOfSegments(4)
scalebar.setNumberOfSegmentsLeft(0)
scalebar.setUnitsPerSegment(0.5)
scalebar.setLinkedMap(map)
scalebar.setUnitLabel('km')
scalebar.setFont(QFont('Arial', 14))
scalebar.update()
layout.addLayoutItem(scalebar)
scalebar.attemptMove(QgsLayoutPoint(220, 190, QgsUnitTypes.LayoutMillimeters))
```

8) Defina as caraterísticas do título que terá o seu layout, como por exemplo, tamanho, fonte, qual será a escrita do título.

```
title = QgsLayoutItemLabel(layout)
title.setText("My Title")
```

```
title.setFont(QFont('Arial', 24))
title.adjustSizeToText()
layout.addLayoutItem(title)
title.attemptMove(QgsLayoutPoint(10, 5, QgsUnitTypes.LayoutMillimeters))
```

Dica: Experimente mudar o tamanho da letra, fonte, para que você observe uma melhor experiência em seu layout.

9) Conclua a construção do seu script, promovendo a capacidade de exportação para pdf, sendo assim adicione o trecho abaixo, sem se esquecer de definir o local de saída e o nome do arquivo pdf:

```
layout = manager.layoutByName(layoutName)

exporter = QgsLayoutExporter(layout)

fn = '<local de saída>/<nome do arquivo>.pdf'
```

10) Por fim exporte, usando o método *exporter.exportToPdf().*

```
exporter.exportToPdf(fn, QgsLayoutExporter.PdfExportSettings())
```

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