

Introdução ao pyQGIS



Módulo 2

Dados

Esta publicação e os dados usados estão disponíveis em:

<https://gdatasystems.com/repo/pyQGISCP1.pdf>

<https://gdatasystems.com.br/learn.php>

VISITE:

<https://gdatasystems.com>

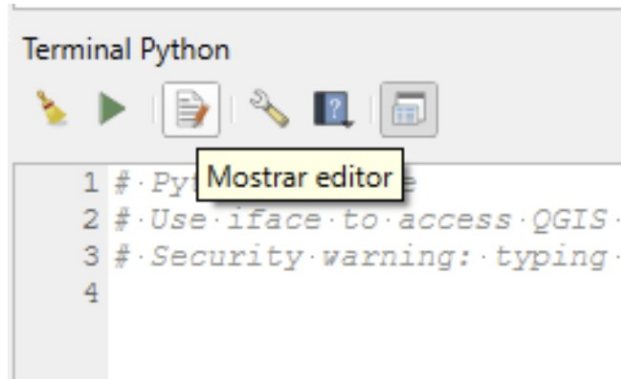
Licença : CC BY 4.0

<https://creativecommons.org/licenses/by/4.0/>

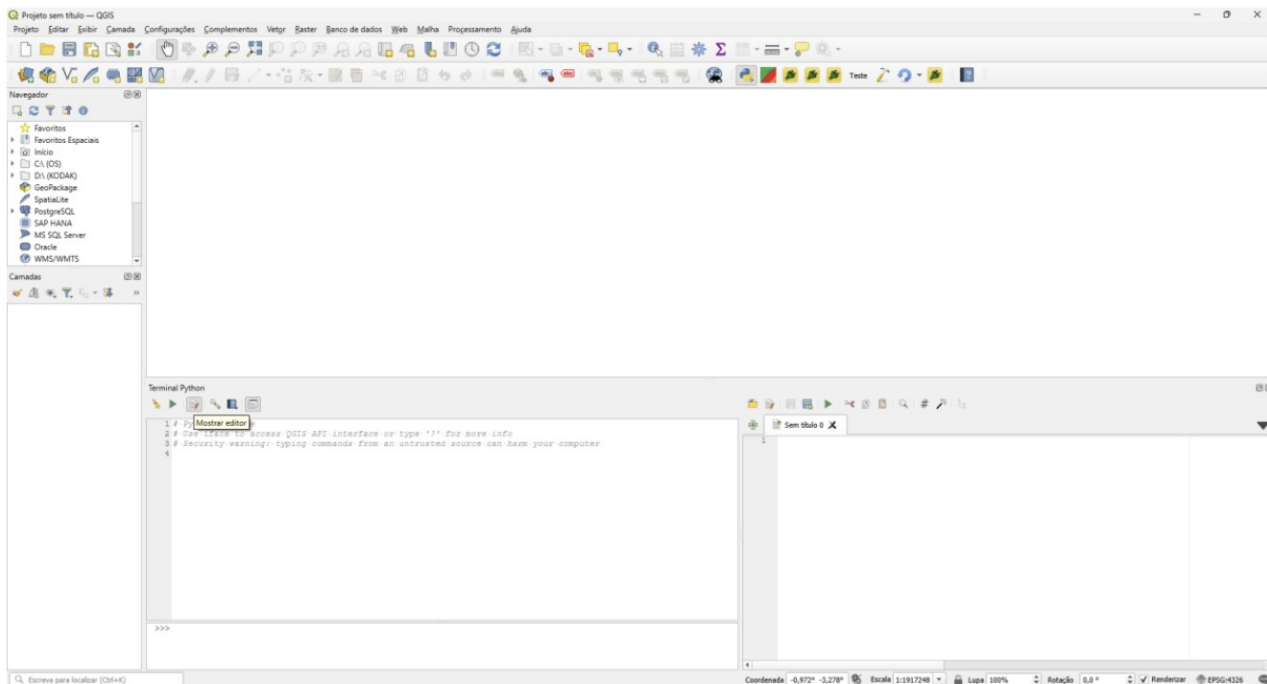
Módulo 2 – Usando pyQGIS: Introdução às primeiras classes

Preparando o Terreno

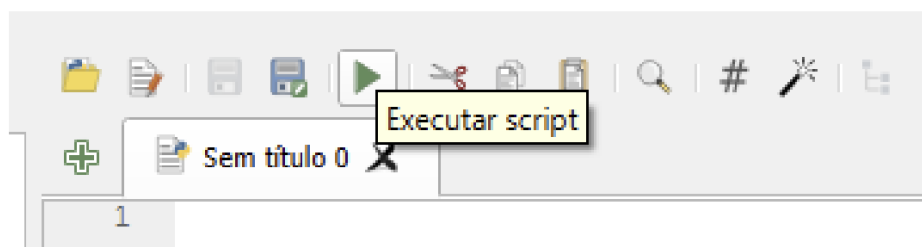
Vamos abrir o QGIS e deixar tudo preparado para podermos operar o ambiente usando o terminal python (CTRL+ALT+P) e clique no botão Mostrar editor:



Veremos a tela desta forma.



Estamos prontos para trabalhar. No editor na parte inferior direita escreveremos o código e executaremos com:



Classes do pyQGIS

Vamos introduzir as classes do PyQGIS na medida que avançamos nos pontos cobertos.

Uma classe em python é a definição de um tipo de objeto e de métodos (funções) associados a este objeto. Por exemplo um projeto, uma camada raster, uma camada vetorial, etc.

A biblioteca PyQGIS é bastante extensa com diversas classes. Vamos aqui cobrir algumas classes mais básicas e essenciais para a partir deste ponto termos uma boa base para desenvolver scripts mais complexos.

Classe Projeto (QgsProject) - Criar e ler um Projeto

Um projeto armazena um conjunto de informações sobre camadas, estilos, layouts, anotações etc. Como se trata de uma classe singleton, criamos um objeto usando o método `QgsProject.instance()`. Vamos mostrar como criar um projeto vazio chamado `meu_projeto.qgs` usando o método `write()`.

```
projeto= QgsProject.instance()
# ajuste caminho para o arquivo a ser criado no seu sistema
projeto.write('C:/Users/User/Desktop/dash/meu_projeto.qgs')
print(projeto.fileName())
C:/Users/User/Desktop/dash /meu_projeto.qgs
```

Agora vamos sair do QGIS e entrar novamente para carregarmos o projeto que criamos usando python.

```
projeto=QgsProject.instance()
projeto.read(' C:/Users/User/Desktop/dash/meu_projeto.qgs')
print(projeto.fileName())
C:/Users/User/Desktop/dash/meu_projeto.qgs
```

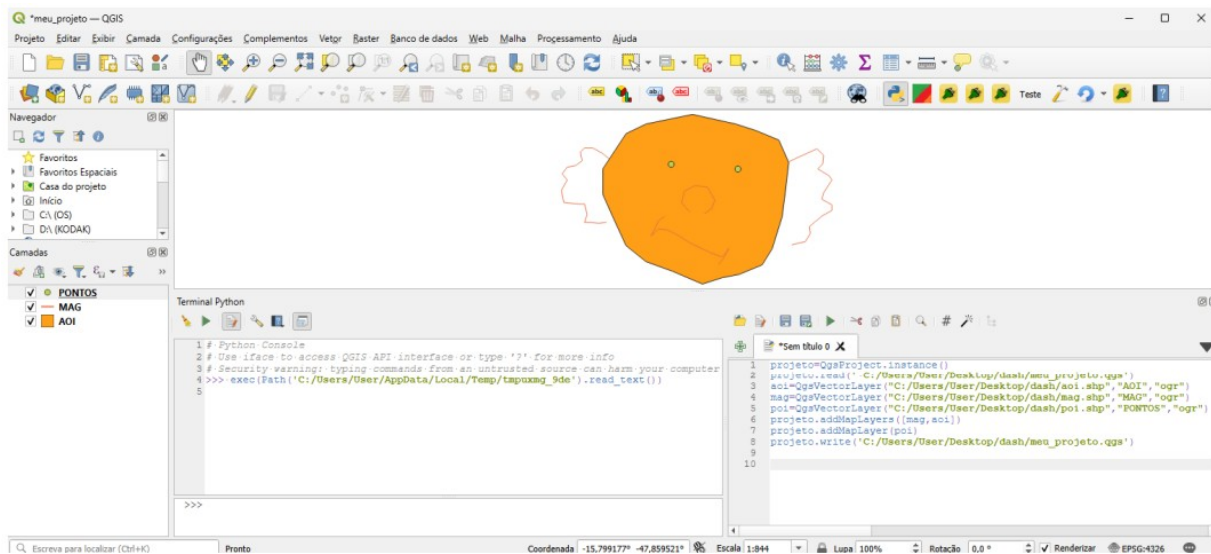
Na medida que formos vendo as outras classes, vamos ver outros métodos associados à classe Projeto.

Classe Vetor (QgsVectorLayer) – Adicionando Camada Vetorial

Um objeto do tipo camada vetorial é usado para carregarmos e interagirmos com camadas do tipo vetor. Vamos carregar o nosso projeto e adicionar nele três camadas vetor criadas anteriormente usando os métodos de projeto `addMapLayer` e `addMapLayers`. Criamos um objeto de classe camada vetorial usando o método `QgsVectorLayer()` passando o caminho para o arquivo vetorial, o identificador que nossa camada terá, e a biblioteca a ser usada (ogr nesse caso). Por último salvamos o nosso projeto com o método `write()`.

```
projeto=QgsProject.instance()
projeto.read(' C:/Users/User/Desktop/dash/meu_projeto.qgs')
aoi=QgsVectorLayer("C:/Users/User/Desktop/dash/aoi.shp", "AOI", "ogr")
mag=QgsVectorLayer("C:/Users/User/Desktop/dash/mag.shp", "MAG", "ogr")
poi=QgsVectorLayer("C:/Users/User/Desktop/dash/poi.shp", "PONTOS", "ogr")
projeto.addMapLayers([mag, aoi])
projeto.addMapLayer(poi)
projeto.write('C:/Users/User/Desktop/dash/meu_projeto.qgs')
```

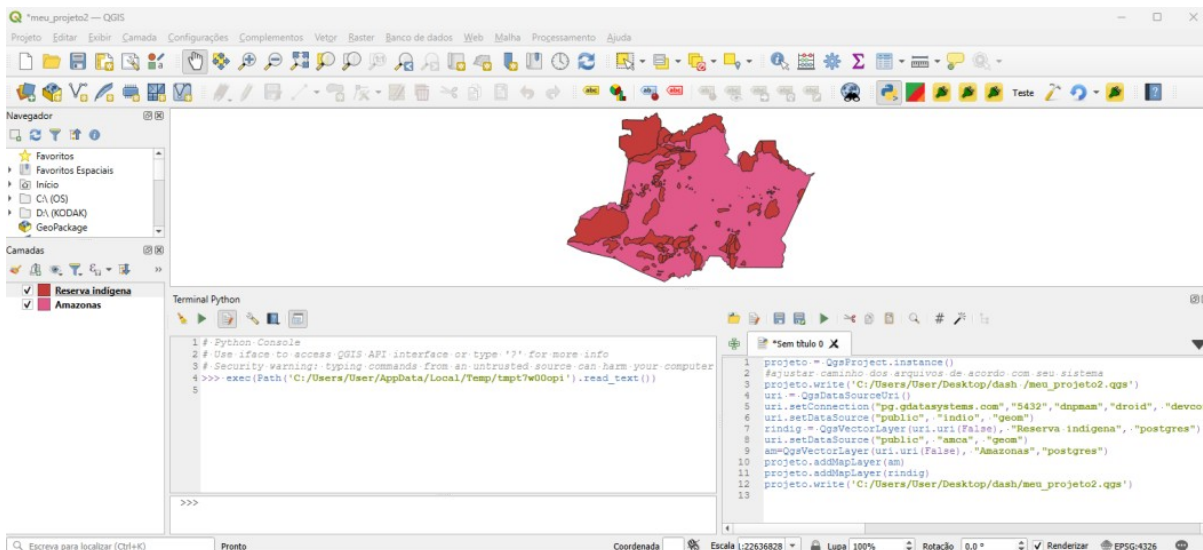
Algo similar ao apresentado abaixo aparecerá:



Agora vamos fazer uso de uma outra classe para podermos carregar uma camada vetorial localizada em um banco de dados Postgis remoto. A classe é a `QgsDataSourceUri` e usaremos os métodos `setConnection()` e `setDataSource()` para extrairmos uma tabela espacial vetorial. Criaremos um projeto novo, adicionaremos uma camada local e uma camada remota Postgis e por último vamos gravar o projeto.

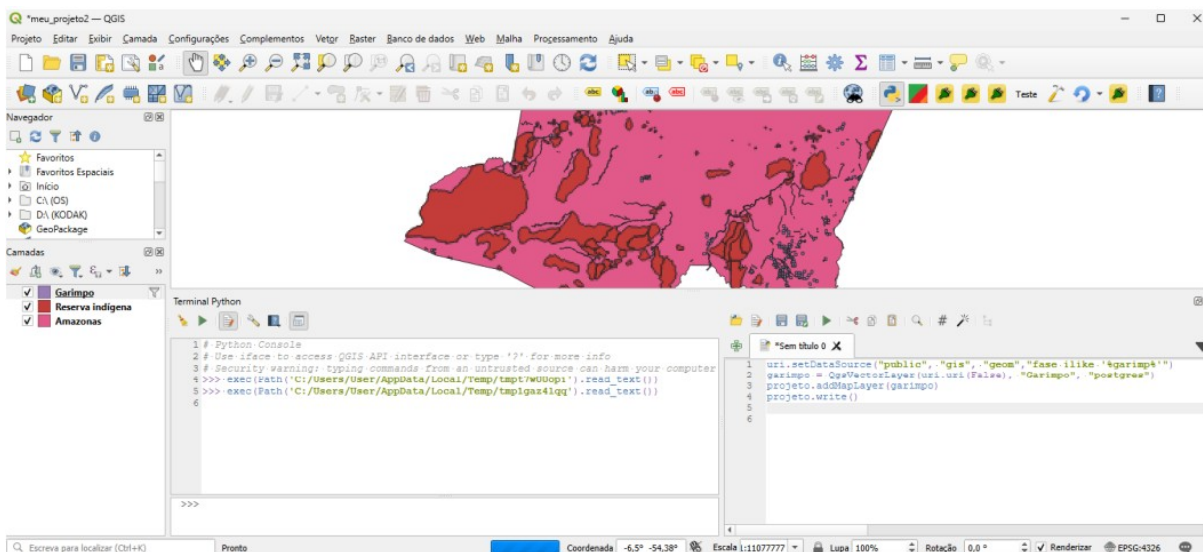
```
projeto = QgsProject.instance()
#ajustar caminho dos arquivos de acordo com seu sistema
projeto.write('C:/Users/User/Desktop/dash/meu_projeto2.qgs')
uri = QgsDataSourceUri()
uri.setConnection("pg.gdatasystems.com", "5432", "dnpmam", "pyqgis", "python")
uri.setDataSource("public", "indio", "geom")
rindig = QgsVectorLayer(uri.uri(False), "Reserva indígena", "postgres")
uri.setDataSource("public", "amca", "geom")
am=QgsVectorLayer(uri.uri(False), "Amazonas", "postgres")
projeto.addMapLayer(am)
projeto.addMapLayer(rindig)
projeto.write('C:/Users/User/Desktop/dash/meu_projeto2.qgs')
```

O método `setConnection()` tem como parâmetros o endereço do servidor (IP ou DNS), a porta (geralmente 5432), o banco de dados, o usuário e a senha. O método `setDataSource()` tem como parâmetros o esquema da tabela, o nome da tabela e a coluna com o elemento geométrico espacial. Um projeto conforme o ilustrado abaixo aparecerá.



Alternativamente podemos adicionar uma cláusula SQL WHERE como o quarto argumento. Vamos ver um exemplo onde extraímos uma camada vetorial somente os requerimentos de garimpo e permissão de lavra garimpeira dos requerimentos do estado do Amazonas e adicionamos ela no nosso projeto já criado acima.

```
uri.setDataSource("public", "gis", "geom", "fase ilike '%garimp%'")
garimpo = QgsVectorLayer(uri.uri(False), "Garimpo", "postgres")
projeto.addMapLayer(garimpo)
projeto.write()
```



Antes de movermos para o próximo tópico vamos dar uma olhada em alguns métodos da Classe Projeto (QgisProject) relacionados a classe Camadas Vetoriais (QgsVectorLayer).

count retorna o número de camadas válidas do projeto.

```
projeto.count()
3
```

mapLayers retorna um mapa das camadas existentes do projeto.

```
projeto.mapLayers()  
{'Amazonas_741f8310_bd3a_4b38_bc38_0a1147ea2769': <QgsVectorLayer: 'Amazonas'  
(postgres)>, 'Garimpo_d47cdf32_a1fb_4c37_8145_7127996261f2': <QgsVectorLayer:  
'Garimpo'  
(postgres)>, 'Reserva_ind_gena_362f9fab_b918_42e4_94d9_e92fb105bead':  
<QgsVectorLayer:  
'Reserva indígena' (postgres)>}
```

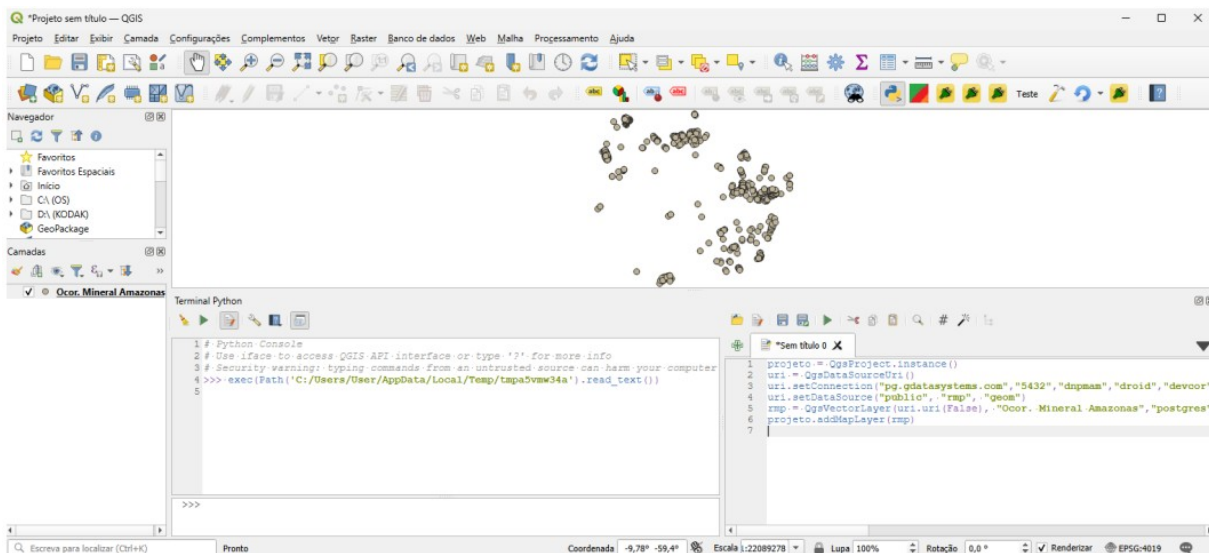
Interagindo com informações de objetos da classe Vector

Podemos obter diversas informações sobre objetos vetoriais tais como, projeções, extensão, número de elementos, valores e nomes dos campos de atributos (colunas) e até criar um metadata da camada (com a informação existente).

Vamos primeiro carregar um dado do tipo ponto de um banco Postgis remoto.

```
projeto = QgsProject.instance()  
uri = QgsDataSourceUri()  
uri.setConnection("pg.gdatasystems.com", "5432", "dnpmam", "pyqgis", "python")  
uri.setDataSource("public", "rmp", "geom")  
rmp = QgsVectorLayer(uri.uri(False), "Ocor. Mineral Amazonas", "postgres")  
projeto.addMapLayer(rmp)
```

A camada será carregada conforme a ilustração mostrada abaixo.



Vamos agora acessar as informações mais usadas de maneira geral. Outras informações existem no objeto camada, veja a documentação para mais detalhes.

O sistema de referência de coordenadas CRS (Coordinate Reference System).

O método `crs()` retorna o sistema de referência de coordenada original do objeto camada que o invoca.

```
crs=rmp.crs()  
print(crs.description())  
SIRGAS 2000
```

Com o método `extent()` de um objeto camada podemos obter os valores máximos e mínimos das coordenadas em X (Easting ou Longitude) e Y (Northing ou Latitude). O método retorna um objeto do tipo retângulo com diversos parâmetros além de X e Y máximos e mínimos tais como `area`, `width`, `height`, `center`, `invert`, etc. Veja a documentação para mais informações.

```
extensão=rmp.extent()
min_x=extensão.xMinimum()
max_x=extensão.xMaximum()
min_y=extensão.yMinimum()
max_y=extensão.yMaximum()
print(min_x,min_y,max_x,max_y)
-70.1 -9.5386 -56.7497 2.2139
```

O método `featureCount()` retorna quantos itens o objeto camada possui.

```
num_elementos=rmp.featureCount()
print("número de elementos: ", num_elementos)
número de elementos: 624
```

Com o método `fields()` obtemos a informação sobre todos os campos de atributos tais como `nome`, `tipo` etc.

```
for field in rmp.fields():
    print (field.name(),field.typeName())
codigo_obj text
TOPONIMIA text
latitude float8
longitude float8
SUBST_PRIN text
subst_sec text
abrev text
STATUS_ECO text
grau_de_im text
metodo_geo text
erro_metod text
data_cad text
classe_util text
tipologia text
classe_gen text
modelo_dep text
assoc_geoq text
rocha_enca text
rocha_hosp text
textura_mi text
tipos_alte text
extrmin_x_ text
assoc_mine text
origem text
municipio text
uf text
```

O método `htmlMetadata()` gera um metadata da camada no formato html que pode ser copiado para um novo arquivo e visualizado em um navegador da web;

```
metadata=rmp.htmlMetadata()
print (metadata)
```

```
<html><head></head>
<body>
<h1>General</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Name</td><td>Ocor. Mineral Amazonas</td></tr>
<tr><td class="highlight">Source</td><td>dbname='dnpmam' host=pg.gdatasystems.com port=5432 user='pyqgis' key='tid'
checkPrimaryKeyUnicity='1' table="public"."rmp" (geom)</td></tr>
<tr><td class="highlight">Provider</td><td>postgres</td></tr>
<tr><td class="highlight">Layer ID</td><td>Ocor_Mineral_Amazonas_677b0b60_ca81_4521_aa55_a7416fe41d4f</td></tr>
</table>
<br><br><h1>Information from provider</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Storage</td><td>PostgreSQL database with PostGIS extension</td></tr>
<tr><td class="highlight">Encoding</td><td></td></tr>
<tr><td class="highlight">Geometry</td><td>Point (Point)</td></tr>
<tr><td class="highlight">Extent</td><td>-70.0999999999999943,-9.5386000003087776 : -56.7497000000000043,2.2139000000729410</td></tr>
<tr><td class="highlight">Feature count</td><td>624</td></tr>
</table>
<br><br><h1>Coordinate Reference System (CRS)</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Name</td><td>EPSG:4674 - SIRGAS 2000</td></tr>
<tr><td class="highlight">Units</td><td>Geographic (uses latitude and longitude for coordinates)</td></tr>
<tr><td class="highlight">Type</td><td>Geographic (2D)</td></tr>
<tr><td class="highlight">Method</td><td>Lat/long (Geodetic alias)</td></tr>
<tr><td class="highlight">Celestial Body</td><td>Earth</td></tr>
<tr><td class="highlight">Reference</td><td>Static (relies on a datum which is plate-fixed)</td></tr>
</table>
<br><br>
<h1>Identification</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Identifier</td><td></td></tr>
<tr><td class="highlight">Parent Identifier</td><td></td></tr>
<tr><td class="highlight">Title</td><td></td></tr>
<tr><td class="highlight">Type</td><td>dataset</td></tr>
<tr><td class="highlight">Language</td><td></td></tr>
<tr><td class="highlight">Abstract</td><td></td></tr>
<tr><td class="highlight">Categories</td><td></td></tr>
<tr><td class="highlight">Keywords</td><td></td></tr>
</table>
<br><br>
<h1>Extent</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">CRS</td><td>EPSG:4674 - SIRGAS 2000 - Geographic</td></tr>
<tr><td class="highlight">Spatial Extent</td><td></td></tr>
<tr><td class="highlight">Temporal Extent</td><td></td></tr>
</table>
<br><br>
<h1>Access</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Fees</td><td></td></tr>
<tr><td class="highlight">Licenses</td><td></td></tr>
<tr><td class="highlight">Rights</td><td></td></tr>
<tr><td class="highlight">Constraints</td><td></td></tr>
</table>
<br><br>
<h1>Fields</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Count</td><td>26</td></tr>
</table>
<br><table width="100%" class="tabular-view">
<tr><th>Field</th><th>Type</th><th>Length</th><th>Precision</th><th>Comment</th></tr>
<tr><td>CODIGO_OBJ</td><td>float8</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>TOPONIMIA</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>LATITUDE</td><td>float8</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>LONGITUDE</td><td>float8</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>SUBST_PRIN</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>SUBST_SEC</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>ABREV</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>STATUS_ECO</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>GRAU_DE_IM</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>METODO_GEO</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>ERRO_METOD</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>DATA_CAD</td><td>timestamp</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>CLASSE_UTI</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>TIPOLOGIA</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>CLASSE_GEN</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>MODELO_DEP</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>ASSOC_GEOQ</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>ROCHA_ENCA</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>ROCHA_HOSP</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>TEXTURA_MI</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>TIPOS_ALTE</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>EXTRMIN_X</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>ASSOC_MINE</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>ORIGEM</td><td>float8</td><td>-1</td><td>0</td><td></td></tr>
<tr><td>MUNICIPIO</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>UF</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
</table>
<br><br><h1>Contacts</h1>
<hr>
<p>No contact yet.</p><br><br>
```

```
<h1>Links</h1>
<hr>
<p>No links yet.</p>
<br><br>
<h1>History</h1>
<hr>
<p>No history yet.</p>
<br><br>
</body>
</html>
```

Essa informação em HTML acima apareceria em um navegador assim:

General

Name	Ocor. Mineral Amazonas
Source	dbname='dnpmam' host=pg.gdatasystems.com port=5432 user='pyqgis' key='tid' checkPrimaryKeyUnicity='1' table="public"."rmp" (geom)
Provider	postgres
Layer ID	Ocor__Mineral_Amazonas_677b0b60_ca81_4521_aa55_a7416fe41d4f

Information from provider

Storage	PostgreSQL database with PostGIS extension
Encoding	
Geometry	Point (Point)
Extent	-70.099999999999943,-9.5386000003087776 : -56.7497000000000043,2.2139000000729410
Feature count	624

Coordinate Reference System (CRS)

Name	EPSG:4674 - SIRGAS 2000
Units	Geographic (uses latitude and longitude for coordinates)
Type	Geographic (2D)
Method	Lat/long (Geodetic alias)
Celestial Body	Earth
Reference	Static (relies on a datum which is plate-fixed)

Identification

Identifier	
Parent Identifier	
Title	
Type	dataset
Language	
Abstract	
Categories	
Keywords	

Extent

CRS	EPSG:4674 - SIRGAS 2000 - Geographic
Spatial Extent	
Temporal Extent	

Access

Fees
Licenses
Rights
Constraints

Fields

Count	26					
	Field	Type	Length	Precision	Comment	
	CODIGO_OBJ	float8	-1	0		
	TOPONIMIA	text	-1	0		
	LATITUDE	float8	-1	0		
	LONGITUDE	float8	-1	0		
	SUBST_PRIN	text	-1	0		
	SUBST_SEC	text	-1	0		
	ABREV	text	-1	0		
	STATUS_ECO	text	-1	0		
	GRAU_DE_IM	text	-1	0		
	METODO_GEO	text	-1	0		
	ERRO_METOD	text	-1	0		
	DATA_CAD	timestampz	-1	0		
	CLASSE_UTI	text	-1	0		
	TIPOLOGIA	text	-1	0		
	CLASSE_GEN	text	-1	0		
	MODELO_DEP	text	-1	0		
	ASSOC_GEOQ	text	-1	0		
	ROCHA_ENCA	text	-1	0		
	ROCHA_HOSP	text	-1	0		
	TEXTURA_MI	text	-1	0		
	TIPOS_ALTE	text	-1	0		
	EXTRMIN_X_	text	-1	0		
	ASSOC_MINE	text	-1	0		
	ORIGEM	float8	-1	0		
	MUNICIPIO	text	-1	0		
	UF	text	-1	0		

Contacts

No contact yet.

Links

No links yet.

History

No history yet.

Com o método `getFeatures()` carregamos todos os dados da camada, onde cada item é armazenado como uma lista. O código abaixo imprime cada um dos itens em formato de lista.

```

elementos=rmp.getFeatures()
for e in elementos:
    attr=e.attributes()
print (attr)
['25368', 'APUI', -7.7972, -58.8558, 'Calcário', NULL, 'cc', 'Não explotado',
'Depósito', 'Levantamento em Carta 1:250.000', '250 a 1.000 m', '2001/11/26',
'Insumos para agricultura', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, '7', 'APUI', 'AM']
...
...
['46734', 'RIO MANICORÉ', -6.11, -61.5669, 'Argila', NULL, 'arg', '(Não
determinado)', 'Depósito', 'GPS Manual pré 25/05/2000', '50 a 200 m', '2006/11/24',
'Material de uso na construção civil', NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, '2', 'MANICORE', 'AM']

```

Criando objeto vetorial

Vamos agora ver os passos para criarmos objetos vetoriais usando python no Qgis. Vamos criar objetos do tipo ponto, linha e polígono para ilustrar o processo.

Ponto

Primeiro definimos o objeto ponto com CRS 4326 (WGS84) com o nome Cidades na memória. Nesse objeto usamos um dataProvider para criar os campos de atributos do objeto vetorial ponto com três atributos (nome, população e IDH) e adicionamos eles no objeto ponto (vponto).

```

vponto = QgsVectorLayer("Point?crs=EPSG:4326", "Cidades","memory")
dPr = vponto.dataProvider()
dPr.addAttributes([QgsField("nome", QVariant.String),QgsField("populacao",
    QVariant.Int), QgsField("idh", QVariant.Double)])
vponto.updateFields()

```

Uma vez criado o objeto ponto e seus campos de atributo vamos adicionar dados nele usando um objeto feature (elemento). Definimos a geometria que será um ponto nesse caso com coordenadas x e y e adicionaremos os atributos deste ponto.

```

elem = QgsFeature()
elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-59.9936,-3.0925)))
elem.setAttributes(["Manaus", 2182763, 0.737])
dPr.addFeature(elem)
elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-60.6253,-3.2872)))
elem.setAttributes(["Manacapuru ", 97377, 0.614])
dPr.addFeature(elem)
elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-60.1883,-3.2756)))
elem.setAttributes(["Iranduba ", 48296, 0.613])
dPr.addFeature(elem)
elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-59.7014,-2.6968)))
elem.setAttributes(["Rio Preto Da Eva ", 33347, 0.611])
dPr.addFeature(elem)

```

Para finalizar vamos configurar a aparência do símbolo mostrado no mapa como estrelas de cor laranjas e de tamanho 8. Atualizamos a extensão do mapa e adicionamos nosso objeto no mapa.

```

symbol = QgsMarkerSymbol.createSimple({'name': 'star',

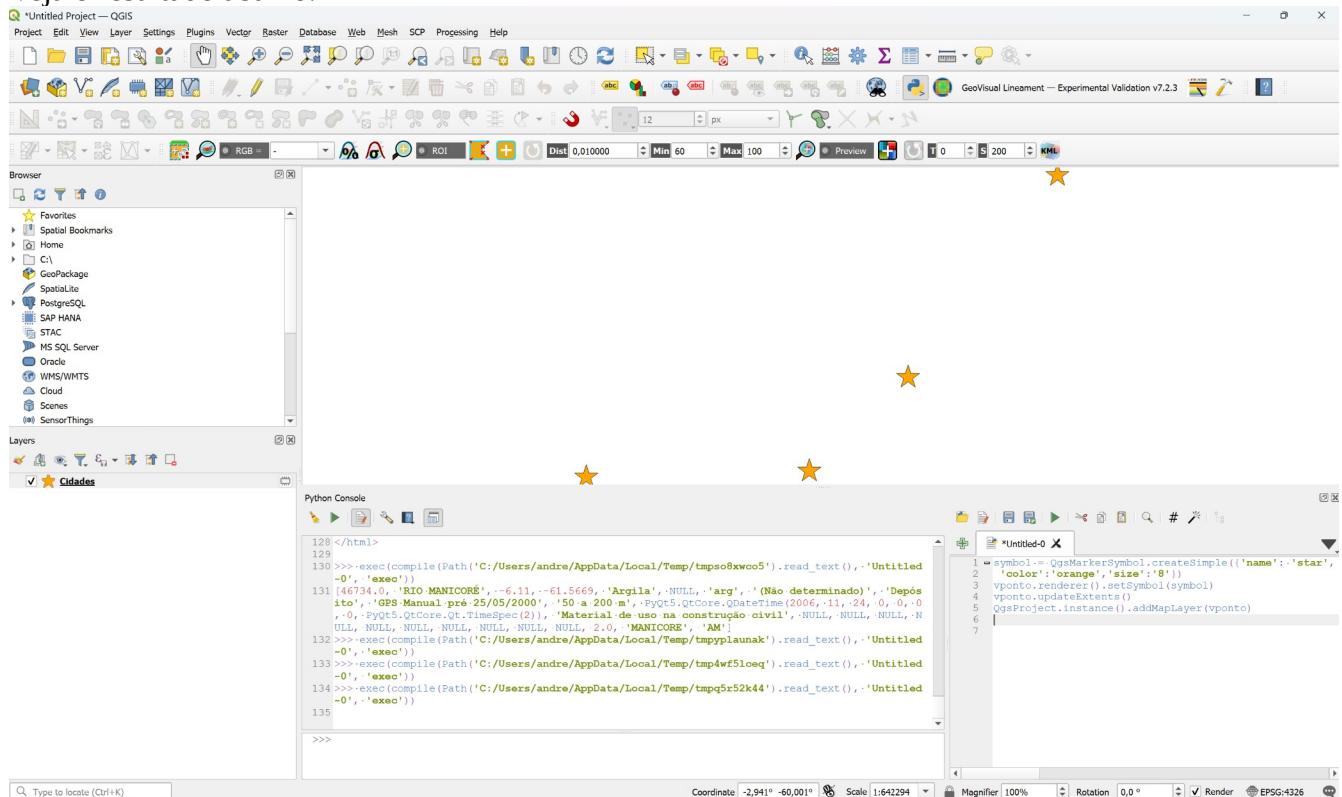
```

```

'color':'orange','size':'8'})
vponto.renderer().setSymbol(symbol)
vponto.updateExtents()
QgsProject.instance().addMapLayer(vponto)

```

Veja o resultado abaixo.



Finalizamos mostrando como escrever o nosso objeto ponto em um arquivo (checar o caminho em seu sistema).

```

QgsVectorFileWriter.writeAsVectorFormat(vponto, 'C:/Users/User/Desktop/
p/dash/cidaIDH.shp', 'utf-8', driverName='ESRI Shapefile')

```

Linha

De forma semelhante ao que fizemos com pontos, criar linhas usando python/Qgis é só uma questão de usarmos uma série (lista) de pontos para cada elemento criado.

Definimos o objeto linha (linestring) com CRS 4326 (WGS84) com o nome Vias na memória. Criamos o dataProvider para adicionar os campos de atributos do objeto vetorial linestring com dois atributos (nome e número) e adicionamos eles no objeto linestring (vlinha).

```

vlinha = QgsVectorLayer("Linestring?crs=EPSG:4326", "Vias", "memory")
dPr = vlinha.dataProvider()
dPr.addAttributes([QgsField("nome", QVariant.String), QgsField("número", QVariant.Int)])
vlinha.updateFields()

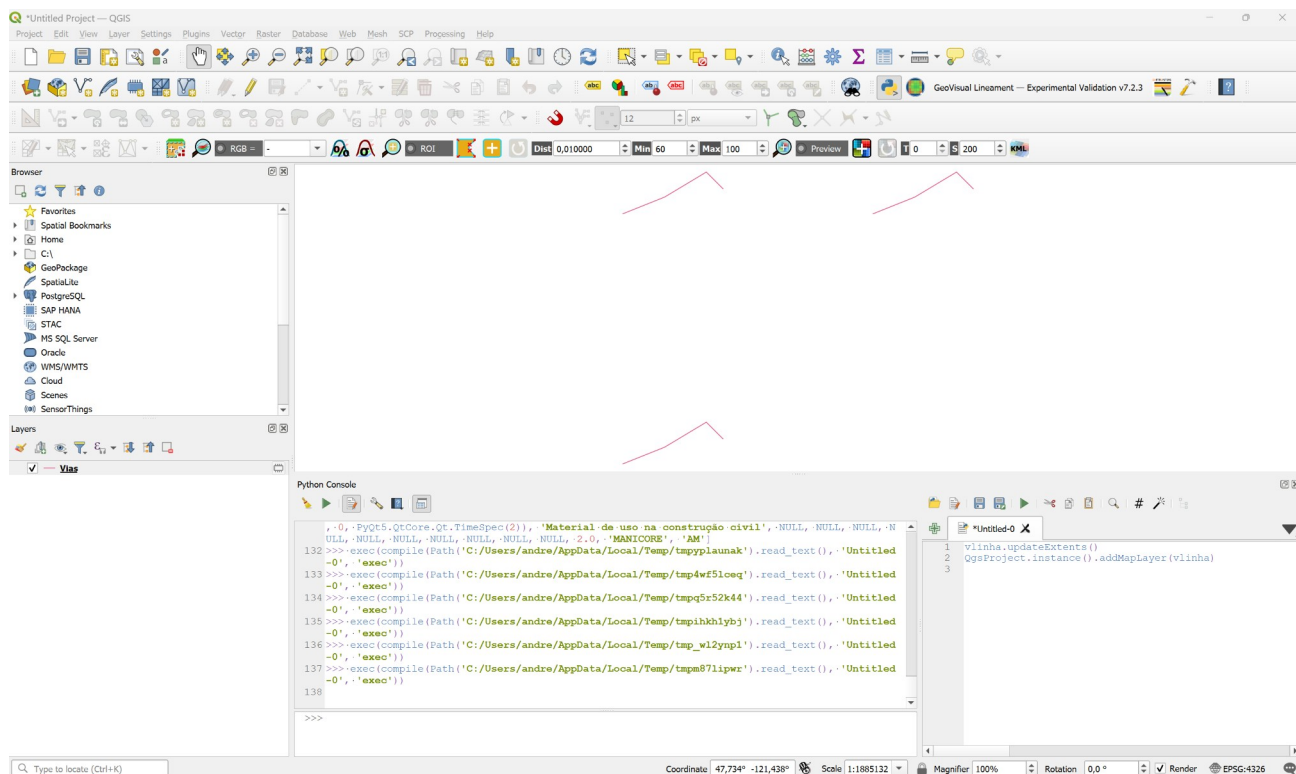
```

Adicionamos as linhas usando um objeto feature (elemento). Mas antes criamos a lista de pontos que farão parte de cada um dos elementos do tipo linha e inserimos os atributos de cada elemento. Vamos inserir três linhas.

```
elem = QgsFeature()
pontos = [QgsPoint(-124,48.4), QgsPoint(-123.5,48.6), QgsPoint(-123,48.9), QgsPoint(-122.8,48.7)]
elem.setGeometry(QgsGeometry.fromPolyline(pontos))
elem.setAttributes(["Rota ", 1])
dPr.addFeature(elem)
pontos = [QgsPoint(-121,48.4), QgsPoint(-120.5,48.6), QgsPoint(-120,48.9), QgsPoint(-119.8,48.7)]
elem.setGeometry(QgsGeometry.fromPolyline(pontos))
elem.setAttributes(["Rota ", 2])
dPr.addFeature(elem)
pontos = [QgsPoint(-124,45.4), QgsPoint(-123.5,45.6), QgsPoint(-123,45.9), QgsPoint(-122.8,45.7)]
elem.setGeometry(QgsGeometry.fromPolyline(pontos))
elem.setAttributes(["Rota ", 3])
dPr.addFeature(elem)
```

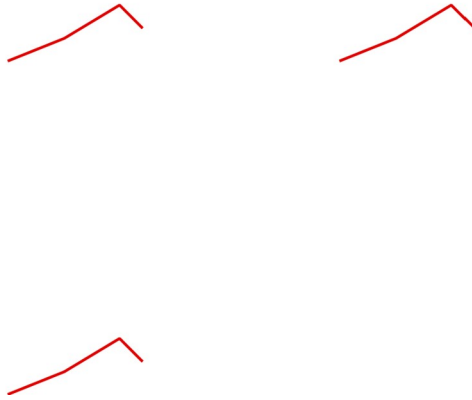
Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
vlinha.updateExtents()
QgsProject.instance().addMapLayer(vlinha)
```



Podemos mudar a aparência de nosso objeto vetorial linestring usando:

```
vlinha.renderer().symbol().setWidth(0.7)
vlinha.renderer().symbol().setColor(QColor(224,0,0))
vlinha.triggerRepaint()
```



Para finalizar, vamos gravar o recém-criado vetor num arquivo do tipo shapefile.

```
QgsVectorFileWriter.writeAsVectorFormat(vlinha, 'C:/Users/User/Desktop/dash/vias.shp', 'utf-8', driverName='ESRI Shapefile')
```

Polígono

Criamos polígonos usando python/Qgis de forma idêntica à forma que criamos linhas só que nesse caso o último ponto se liga ao primeiro ponto informado. Definimos o objeto polígono com CRS 4326 (WGS84) com o nome Fazendas na memória.

Criamos o dataProvider para adicionar os campos de atributos do objeto vetorial polígono com dois atributos (nome e número) e adicionamos eles no objeto polígono (vpgon).

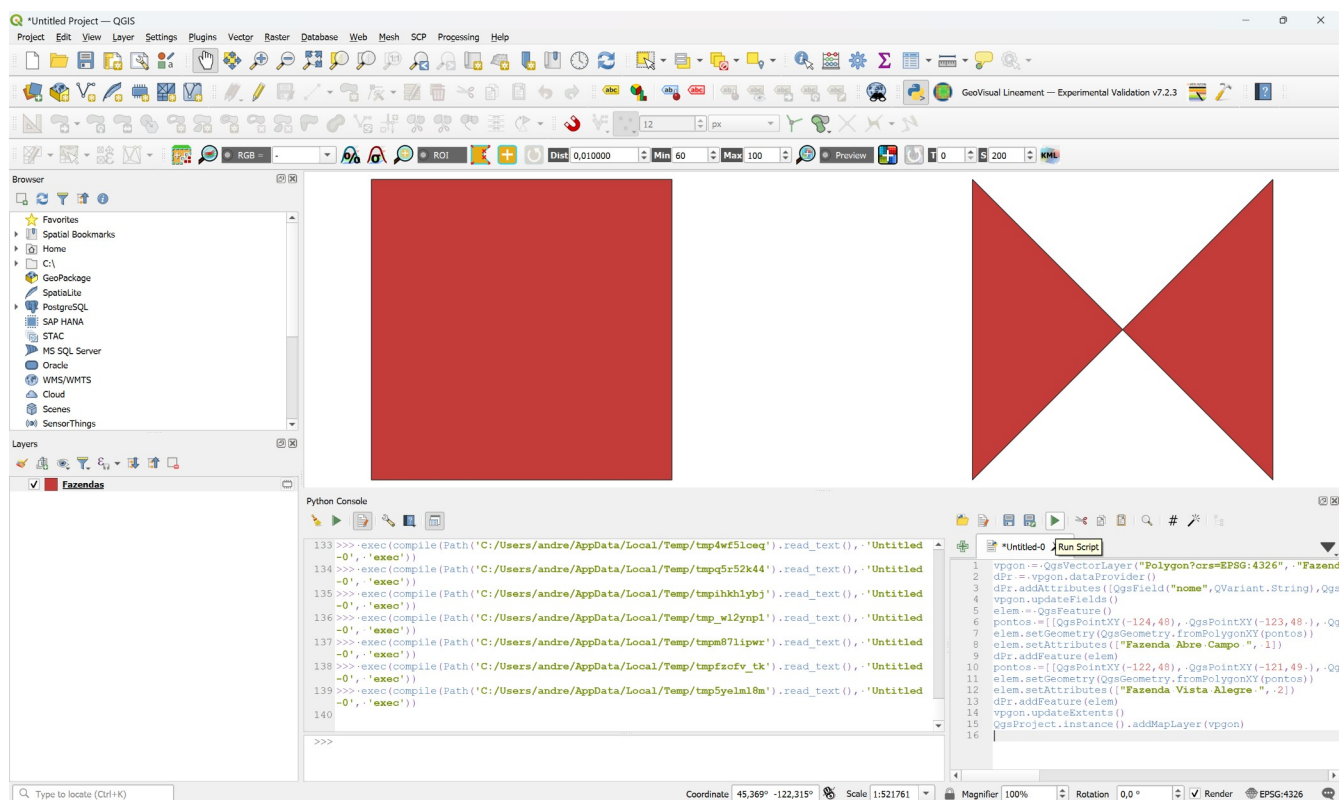
```
vpgon = QgsVectorLayer("Polygon?crs=EPSG:4326", "Fazendas", "memory")
dPr = vpgon.dataProvider()
dPr.addAttributes([QgsField("nome", QVariant.String), QgsField("número",
QVariant.Int)])
vpgon.updateFields()
```

Adicionamos os polígonos usando um objeto feature (elemento). Mas antes criamos a lista de pontos que farão parte de cada um dos elementos do tipo polígono (em colchete duplo) e inserimos os atributos de cada elemento. Vamos inserir dois polígonos.

```
elem = QgsFeature()
pontos = [[QgsPointXY(-124,48), QgsPointXY(-123,48), QgsPointXY(-123,49),
QgsPointXY(-124,49)]]
elem.setGeometry(QgsGeometry.fromPolygonXY(pontos))
elem.setAttributes(["Fazenda Abre Campo ", 1])
dPr.addFeature(elem)
pontos = [[QgsPointXY(-122,48), QgsPointXY(-121,49), QgsPointXY(-121,48),
QgsPointXY(-122,49)]]
elem.setGeometry(QgsGeometry.fromPolygonXY(pontos))
elem.setAttributes(["Fazenda Vista Alegre ", 2])
dPr.addFeature(elem)
```

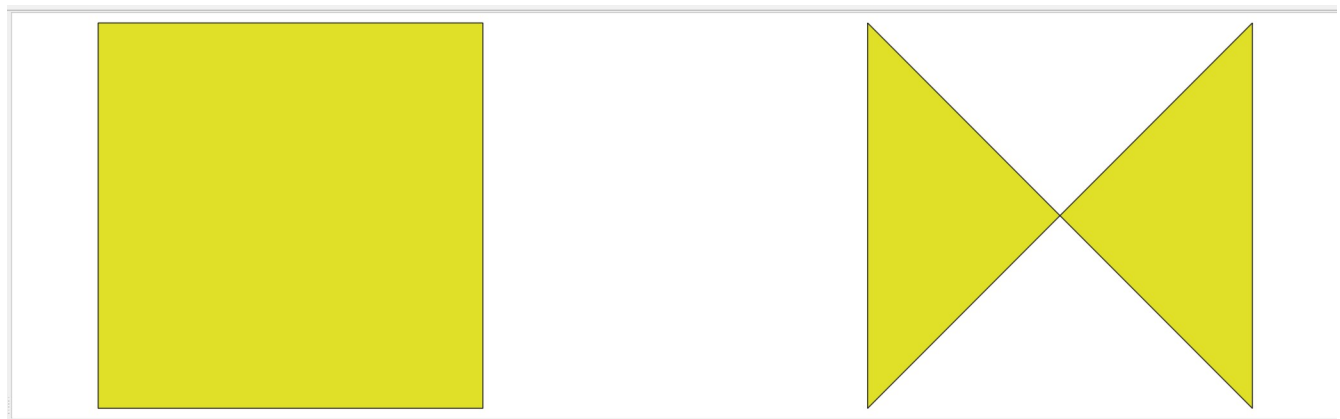
Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
vpgon.updateExtents()
QgsProject.instance().addMapLayer(vpgon)
```



Podemos mudar a aparência de nosso objeto vetorial polígono usando:

```
vpgon.renderer().symbol().setColor(QColor(224,224,40))
vpgon.triggerRepaint()
```



Gravamos o vetor recém-criado em um arquivo do tipo shapefile usando.

```
QgsVectorFileWriter.writeAsVectorFormat(vpgon, 'C:/Users/User/Desktop
/dash/fazendas.shp', 'utf-8', driverName='ESRI Shapefile')
```

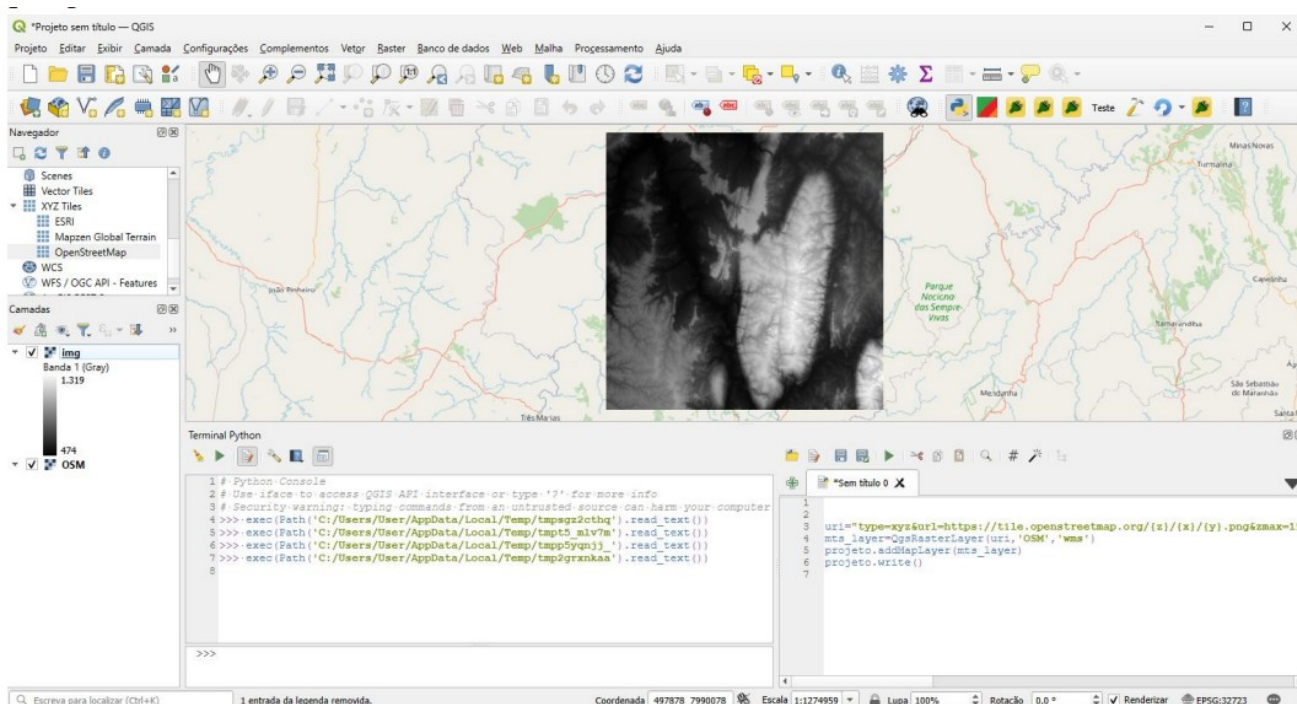
Classe Raster (QgsRasterLayer)

Similar à forma que adicionamos camadas vetoriais, podemos adicionar imagens raster no nosso projeto usando objeto da a classe raster. Vamos criar um projeto e adicionar uma imagem raster nele.

```
projeto=QgsProject.instance()
camadaR =QgsRasterLayer("C:/Users/user/desktop/dash/dem2.tif","img")
projeto.addMapLayer(camadaR)
projeto.write('C:/Users/user/desktop/dash/meu_projeto3.qgs')
```

Podemos também adicionar dados do tipo raster usando provedores do tipo TMS (TileMapService) ou WMS (WebMapService).

```
uri="type=xyz&url=https://tile.openstreetmap.org/{z}/{x}/{y}.png&zmax=19&zmin=0"
mts_layer=QgsRasterLayer(uri,'OSM','wms')
projeto.addMapLayer(mts_layer)
projeto.write()
```

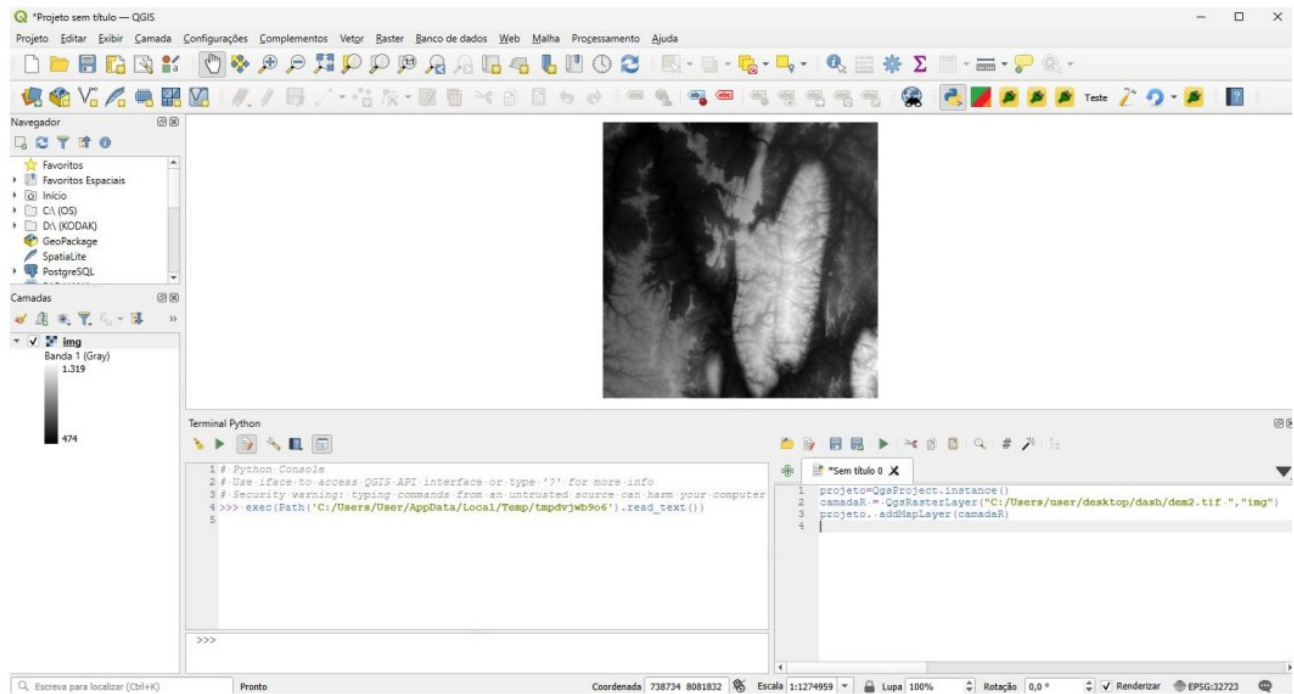


Interagindo com informações de objetos da classe Raster

Podemos extrair informações relevantes de um objeto raster também tais como dimensões, resoluções, número de bandas, valor de um pixel, etc. Vamos carregar uma imagem inicialmente.

```
projeto=QgsProject.instance()
camadaR = QgsRasterLayer("C:/Users/user/desktop/dash/dem2.tif ","img")
projeto.addMapLayer(camadaR)
```

Isso abrirá a imagem como mostrado abaixo:



Informações de dimensão do objeto Raster

Podemos acessar informações de parâmetros dimensionais de uma imagem raster usando métodos específicos para o tal.

```
camadaR.width(), camadaR.height() #largura e altura
(3649, 3650)
camadaR.extent() #extensão da imagem na unidade da coordenada
<QgsRectangle: 500000 7990240, 609780 8100040>
camadaR.crs().description() #sistema de referência
'WGS 84 / UTM zone 23S'
camadaR.rasterUnitsPerPixelX() #resolução em X
30.08495478213209
camadaR.rasterUnitsPerPixelY() #resolução em Y
30.08219178082192
```

Essas importantes informações sobre o raster poderão ser usadas para análises espaciais futuras. Existem outras formas de usar os métodos para obtermos a mesma informação.

Podemos calcular a resolução em X usando o código abaixo em vez de usar o método `rasterUnitsPerPixelX()`:

```
(camadaR.extent().xMaximum() - camadaR.extent().xMinimum()) / camadaR.width()
30.08495478213209
```

Podemos também visualizar informações usando o método `htmlMetadata()`.

```
camadaR.htmlMetadata()
```

Texto HTML gerado acima visualizado no navegador.

General

Name	img
Source	C:/Users/andre/Desktop/pyqgis/dem2.tif
Provider	gdal
Layer ID	img_3293a840_5dc7_4287_b5f3_447604468edb

Information from provider

Extent	500000.0000000000000000,7990240.0000000000000000 : 609780.0000000000000000,8100040.0000000000000000
Width	3649
Height	3650
Data type	Int16 - Sixteen bit signed integer
GDAL Driver Description	GTiff
GDAL Driver Metadata	GeoTIFF
Dataset Description	C:/Users/andre/Desktop/pyqgis/dem2.tif
Compression	- STATISTICS_APPROXIMATE=YES - STATISTICS_MINIMUM=474 - STATISTICS_MAXIMUM=1319 - STATISTICS_MEAN=739.82844614963 - STATISTICS_STDDEV=201.15310812442
Band 1	STATISTICS_VALID_PERCENT=100
	- Scale: 1 - Offset: 0
More information	- TIFFTAG_XRESOLUTION=1 - TIFFTAG_YRESOLUTION=1 - AREA_OR_POINT=Area
Dimensions	X: 3649 Y: 3650 Bands: 1
Origin	500000.0000000000000000,8100040.0000000000000000
Pixel Size	30.0849547821320904,-30.08219178082191902

Coordinate Reference System (CRS)

Name	EPSG:32723 - WGS 84 / UTM zone 23S
Units	meters
Type	Projected
Method	Universal Transverse Mercator (UTM)
Celestial Body	Earth
Accuracy	Based on <i>World Geodetic System 1984 ensemble</i> (EPSG:6326), which has a limited accuracy of at best 2 meters .
Reference	Dynamic (relies on a datum which is not plate-fixed)

Identification

Identifier	
Parent Identifier	
Title	
Type	dataset
Language	
Abstract	

Categories
Keywords

Extent

CRS EPSG:32723 - WGS 84 / UTM zone 23S - Projected
Spatial Extent
Temporal Extent

Access

Fees
Licenses
Rights
Constraints

Bands

Band count 1

Number	Band	NoData	Min	Max
1	Band 1	n/a	474.0000000000	1319.0000000000

Contacts

No contact yet.

References

No links yet.

History

No history yet.

Custom properties

identify/format	Value
-----------------	-------

Raster com uma banda de valores

Vamos ver agora métodos para raster de uma banda. Os métodos abaixo informam o número de bandas e o tipo da imagem raster. 0 para cinza ou não definido de banda única, 1 para paletado de banda única e 2 para multibanda.

```
camadaR.bandCount()
1
camadaR.rasterType()
<RasterLayerType.GrayOrUndefined: 0>
```

A função `dataProvider()` funciona como uma interface entre o objeto raster os seus dados individuais, seu método `sample()` toma dois valores, um objeto ponto (coordenadas XZ) e o número da banda. Se a coordenada for dentro da imagem e a banda existir o resultado será um tuple com o valor do pixel e se o dado é verdadeiro ou não.

```
valor=camadaR.dataProvider().sample(QgsPointXY(687567, 7460876),1)
valor
(nan, False)
valor2=camadaR.dataProvider().sample(QgsPointXY(547802,8043049), 1)
valor2
(980.0, True)
```

A rampa de cor assinalada ao objeto raster pode ser checada usando o método `type()` do método `renderer()`. O tipo `singlebandgray` é o padrão inicial.

```
camadaR.renderer().type()
'singlebandgray'
```

Podemos alterar via python a rampa de cores, o processo é mostrado abaixo. O processo envolve na criação de um objeto do tipo `ColorRampShader` e definimos a rampa de cor de preenchimento como sendo do tipo interpolado.

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

Criamos agora uma lista com as cores representando os dois valores extremos do raster (0 e 2046 que serão interpolados entre azul e amarelo. Em seguida adicionamos esta lista como item do `ColorRampShader` criado acima.

```
lista = [ QgsColorRampShader.ColorRampItem(0, QColor(0,0,255)),
QgsColorRampShader.ColorRampItem(2046, QColor(255,255,0)) ]
fcn.setColorRampItemList(lista)
```

O próximo passo é criarmos o `RasterShader` (preenchedor de cor) e associarmos o `RampShader` a ele.

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

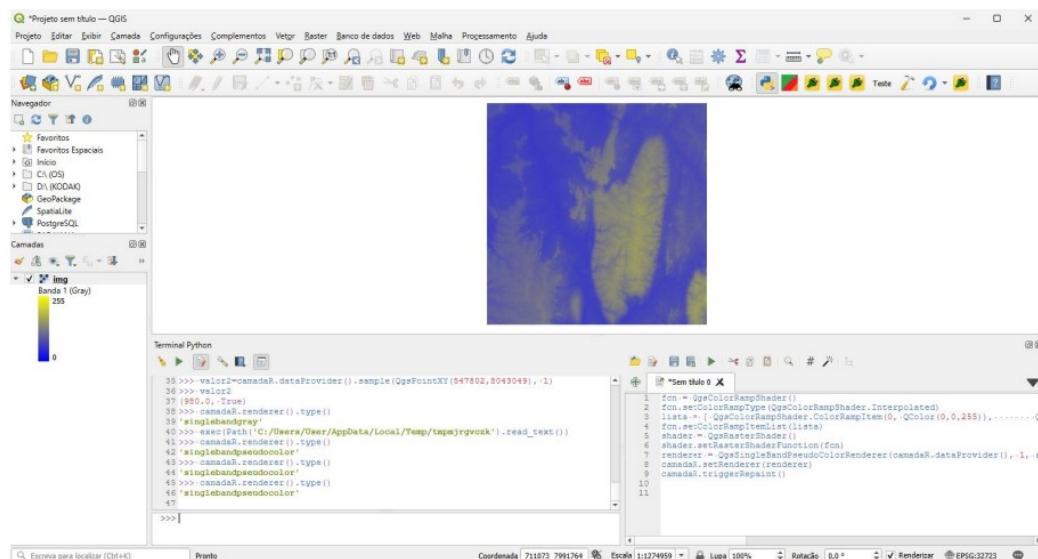
Finalmente criamos o objeto renderizador de cor com: dados do objeto raster, banda 1 e shader acima. Em seguida aplicamos este ao objeto raster e chamamos a repintura do objeto.

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

Se chamarmos o tipo novamente podemos ver a mudança.

```
camadaR.renderer().type()
'singlebandpseudocolor'
```

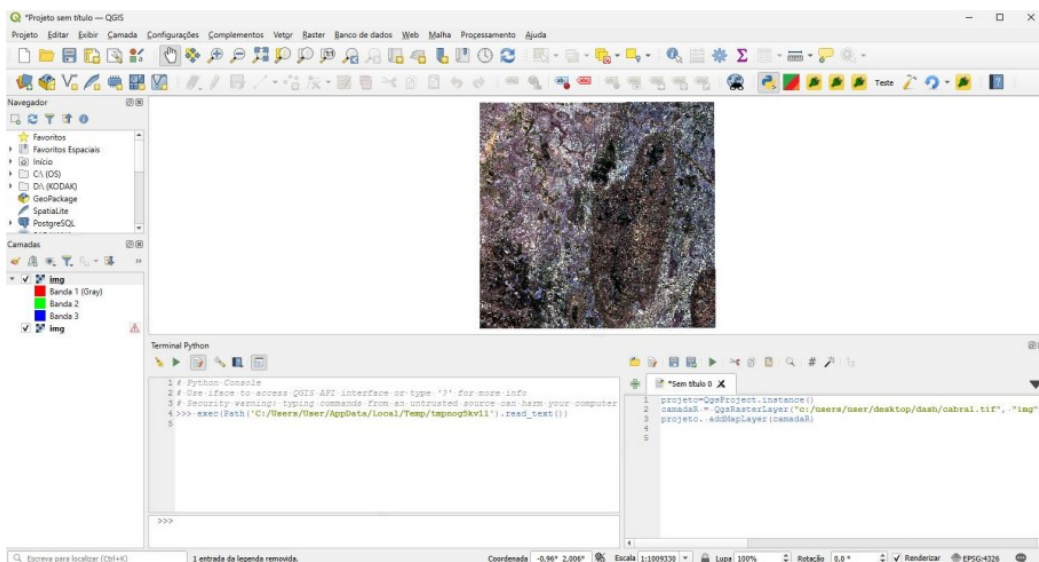
O resultado é mostrado na imagem abaixo.



Raster com mais de uma banda de valores

Vamos ver agora métodos para raster de mais de uma banda. Vamos trabalhar um pouco agora com imagem raster de 3 bandas. Carregamos o raster de forma similar e vamos extrair algumas de suas informações.

```
projeto=QgsProject.instance()
camadaR = QgsRasterLayer("c:/users/user/desktop/dash/cabral.tif","img")
projeto.addMapLayer(camadaR)
```



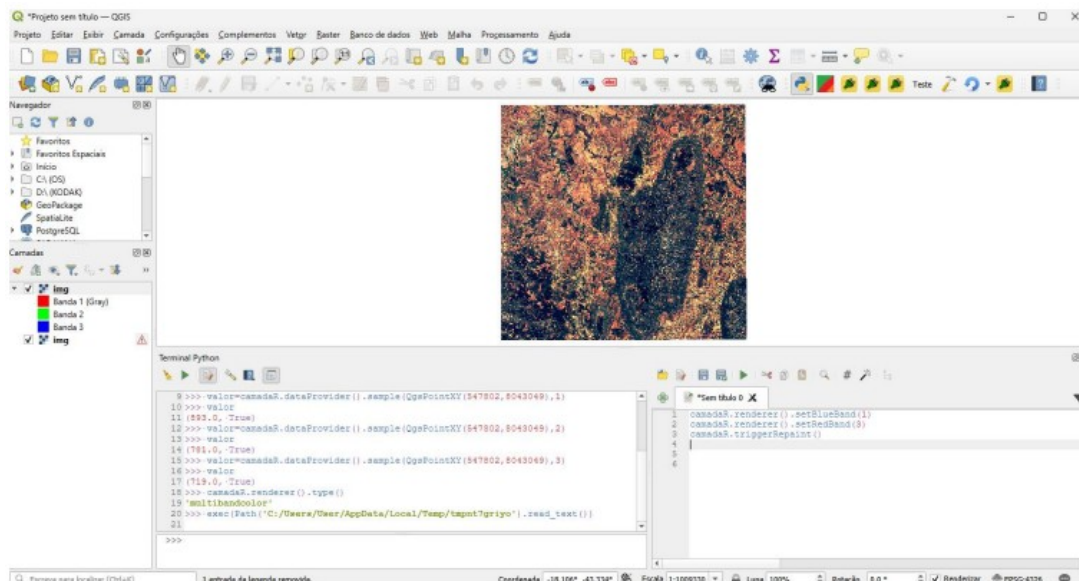
Podemos ver algumas das informações usando:

```
camadaR.bandCount() # número de bandas
3
camadaR.rasterType() # 2 para multi banda
<RasterLayerType.MultiBand: 2>
# valor do pixel na banda 1
valor=camadaR.dataProvider().sample(QgsPointXY(547802,8043049),1)
valor
(893.0, True)
# valor do pixel na banda 2
valor=camadaR.dataProvider().sample(QgsPointXY(547802,8043049),2)
valor
(781.0, True)
# valor do pixel na banda 3
valor=camadaR.dataProvider().sample(QgsPointXY(547802,8043049),3)
valor
(719.0, True)
camadaR.renderer().type()
'multibandcolor'
```

Vamos ver abaixo como modificar a imagem para que a banda 1 fique no canal azul (B) e a banda 3 fique no canal Vermelho (R).

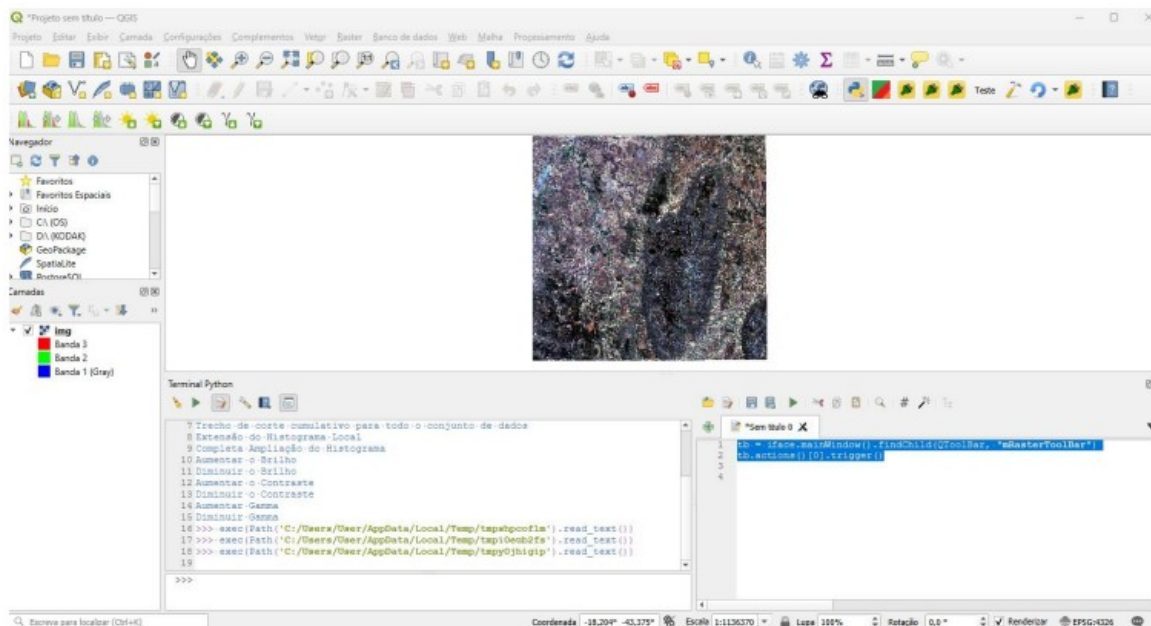
```
camadaR.renderer().setBlueBand(1)
camadaR.renderer().setRedBand(3)
camadaR.triggerRepaint()
```

Note que o histograma da imagem não foi apropriadamente ajustado porque ainda usa os valores de máximo e mínimo das bandas anteriores.



Para ajustar execute:

```
tb = iface.mainWindow().findChild(QToolBar, "mRasterToolBar")
tb.actions()[0].trigger()
```



Criando objeto raster

Imagens raster também podem ser criadas via script de forma bem eficiente usando uma lista de dados pontuais com um determinado valor. Vamos aqui criar um raster mostrando a temperatura média de uma área com base em informações pontuais de vários locais. O arquivo CSV tfinal.csv tem os dados com coordenadas, e respectivos valores. Vamos carregar a informação em um objeto do tipo QgsInterpolator camada de dados (layerData).

```
uri="file:///c:/users/user/desktop/dash/tfinal.csv"
type=csv&xField=LONGITUDE&yField=LATITUDE&crs=epsg:4326"
camada = QgsVectorLayer(uri, 'Converte', "delimitedtext")
c_data = QgsInterpolator.LayerData()
c_data.source = camada
c_data.zCoordInterpolation = False
c_data.interpolationAttribute = 6
c_data.sourceType = QgsInterpolator.SourcePoints
```

Executaremos a interpolação usando o inverso da distância ponderada (IDW) ao quadrado (coeficiente 2).

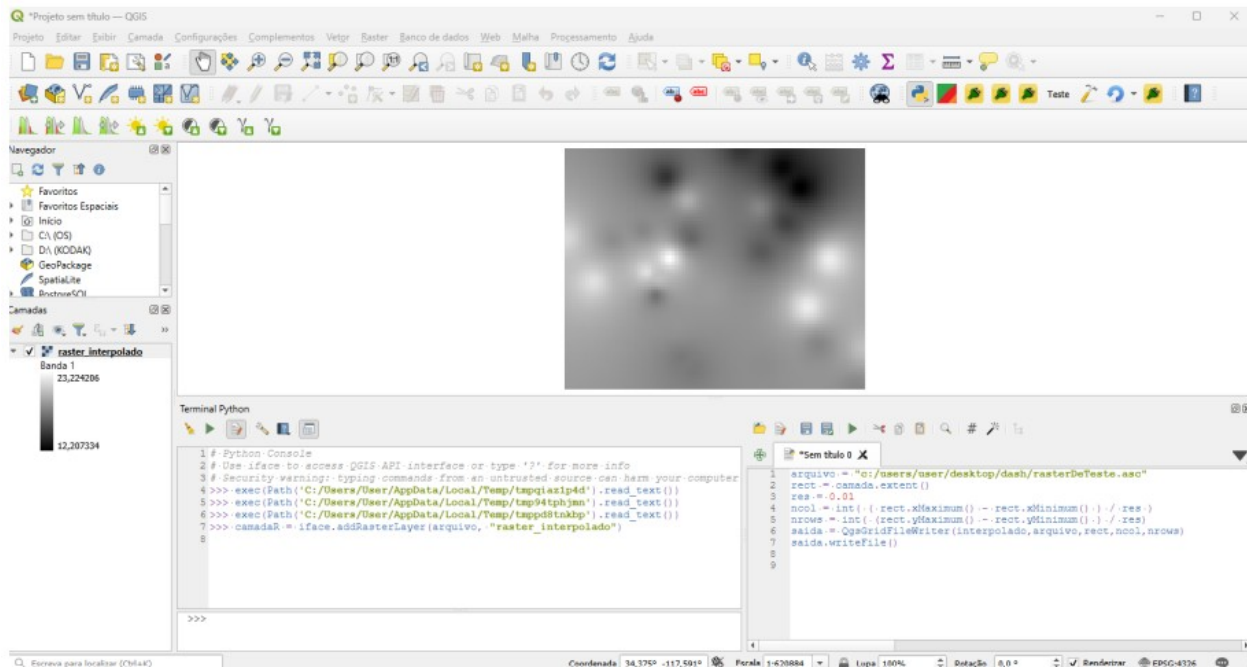
```
interpolado = QgsIDWInterpolator([c_data])
interpolado.setDistanceCoefficient(2)
```

Agora definimos qual arquivo será criado e os parâmetros do grid a ser usado.

```
Arquivo = "c:/users/user/desktop/dash/rasterDeTeste.asc"
rect = camada.extent()
res = 0.01
ncol = int( ( rect.xMaximum() - rect.xMinimum() ) / res )
nrows = int( ( rect.yMaximum() - rect.yMinimum() ) / res)
saida = QgsGridFileWriter(interpolado,arquivo,rect,ncol,nrows)
saida.writeFile()
```

Carregamos o arquivo do grid usando.

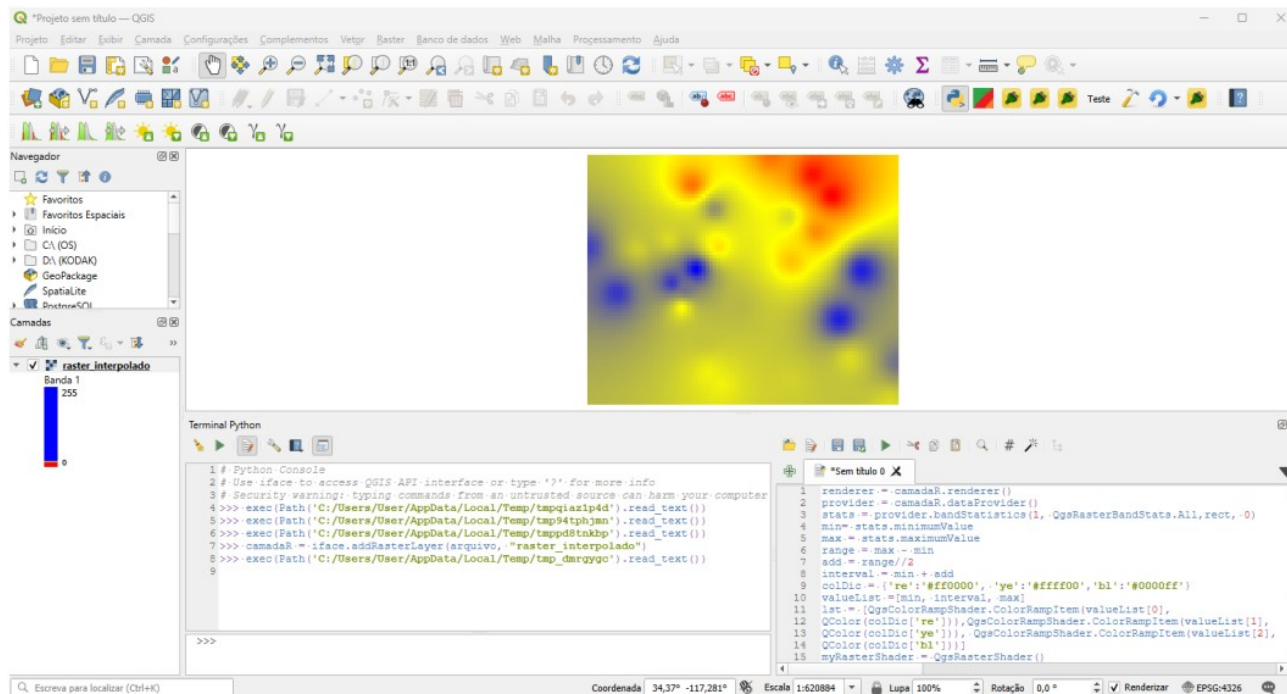
```
camadaR = iface.addRasterLayer(arquivo, "raster_interpolado")
```



Podemos alterar a aparência do raster que criamos usando o código seguinte.

```
renderer = camadaR.renderer()
provider = camadaR.dataProvider()
stats = provider.bandStatistics(1, QgsRasterBandStats.All, rect, 0)
min= stats.minimumValue
max = stats.maximumValue
range = max - min
add = range//2
interval = min + add
colDic = {'re':'#ff0000', 'ye':'#ffff00', 'bl':'#0000ff'}
valueList =[min, interval, max]
lst = [QgsColorRampShader.ColorRampItem(valueList[0],
QColor(colDic['re'])),QgsColorRampShader.ColorRampItem(valueList[1],
QColor(colDic['ye'])),
QgsColorRampShader.ColorRampItem(valueList[2],
QColor(colDic['bl']))]
myRasterShader = QgsRasterShader()
myColorRamp = QgsColorRampShader()
myColorRamp.setColorRampItemList(lst)
myRasterShader.setRasterShaderFunction(myColorRamp)
myPseudoRenderer
=QgsSingleBandPseudoColorRenderer(camadaR.dataProvider(),
camadaR.type(), myRasterShader)
camadaR.setRenderer(myPseudoRenderer)
camadaR.triggerRepaint()
```

O resultado da rampa de cores criada aplicado no raster será.



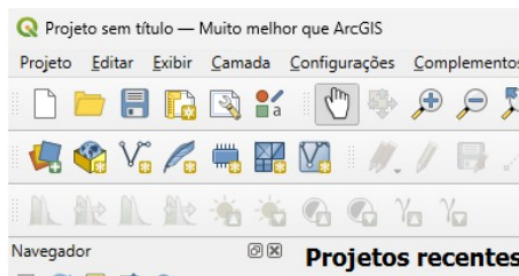
iface – QgisInterface

A classe QgisInterface é iniciada automaticamente quando o QGIS é iniciado e ela é representada pela variável iface (de interface). O iface é o seu programa QGIS e representa a tela principal, os menus, a barra de ferramentas e as demais partes do programa.

No módulo anterior usamos o iface em duas ocasiões, uma para acessar uma ferramenta da barra de ferramenta de raster, e outra para carregar um raster gerado por interpolação. Vamos agora ver outras funções da classe iface.

Mudando o título do nome do programa

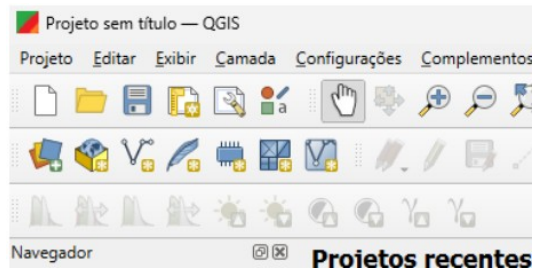
```
titulo = iface.mainWindow().windowTitle()
novo_titulo = titulo.replace('QGIS', 'Muito melhor que ArcGIS')
iface.mainWindow().setWindowTitle(novo_titulo)
```



Mudando o ícone do programa

Mudamos o ícone usando o código abaixo. O `os.path.expanduser('~')` te leva para o pasta raiz de usuário no seu sistema.

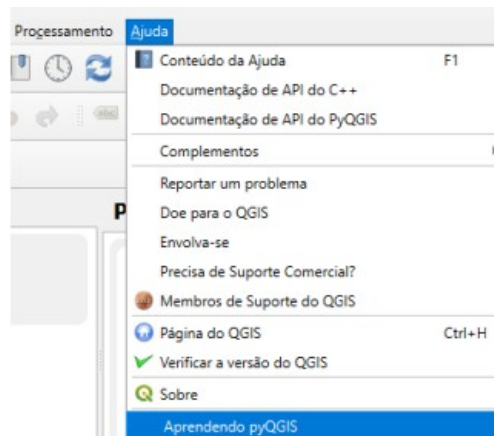
```
import os
icone = 'icon.png'# use um arquivo png 64x64 pixels
dire = os.path.join(os.path.expanduser('~'), 'desktop/dash/')
path = os.path.join(dire, icone)
icone2 = QIcon(path)
iface.mainWindow().setWindowIcon(icone2)
```



Adicionando um novo item de menu ajuda

Com o seguinte código adicionamos um novo item no menu Ajuda que direciona para a nossa página do Aprendendo pyQGIS.

```
import webbrowser
def abreSite():
    webbrowser.open('https://gdatasystems.com/learn.php')
acao = QAction('Aprendendo pyQGIS ')
acao.triggered.connect(abreSite)
iface.helpMenu().addSeparator()
iface.helpMenu().addAction(acao)
```

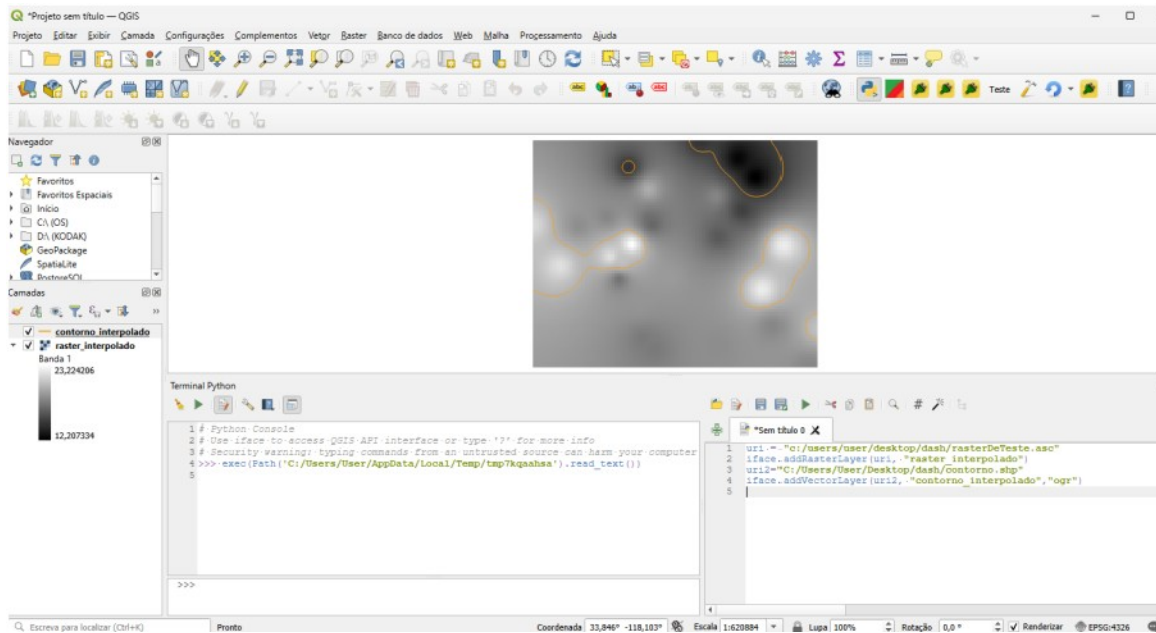


Clicando no item a página vai abrir no navegador.

Adicionando uma nova camada

Como já fizemos no módulo anterior podemos adicionar novas camadas usando o iface.

```
uri = "c:/users/user/desktop/dash/rasterDeTeste.asc"
iface.addRasterLayer(uri, "raster_interpolado")
uri2="C:/Users/User/Desktop/dash/contorno.shp"
iface.addVectorLayer(uri2, "contorno_interpolado", "ogr")
```



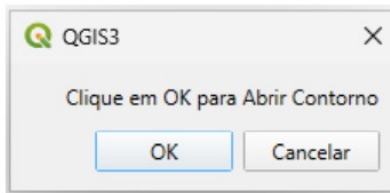
PyQt

Qt é um conjunto de ferramentas ou plataforma open-source de widget (controles do tipo botões, entrada de texto, rótulos, etc) para criar interfaces gráficas de usuário (GUI) e aplicativos multiplataforma. O QGIS é construído sobre esta plataforma. O PyQt é a interface Python do Qt. O PyQt fornece classes e funções de interação com os widgets QT.

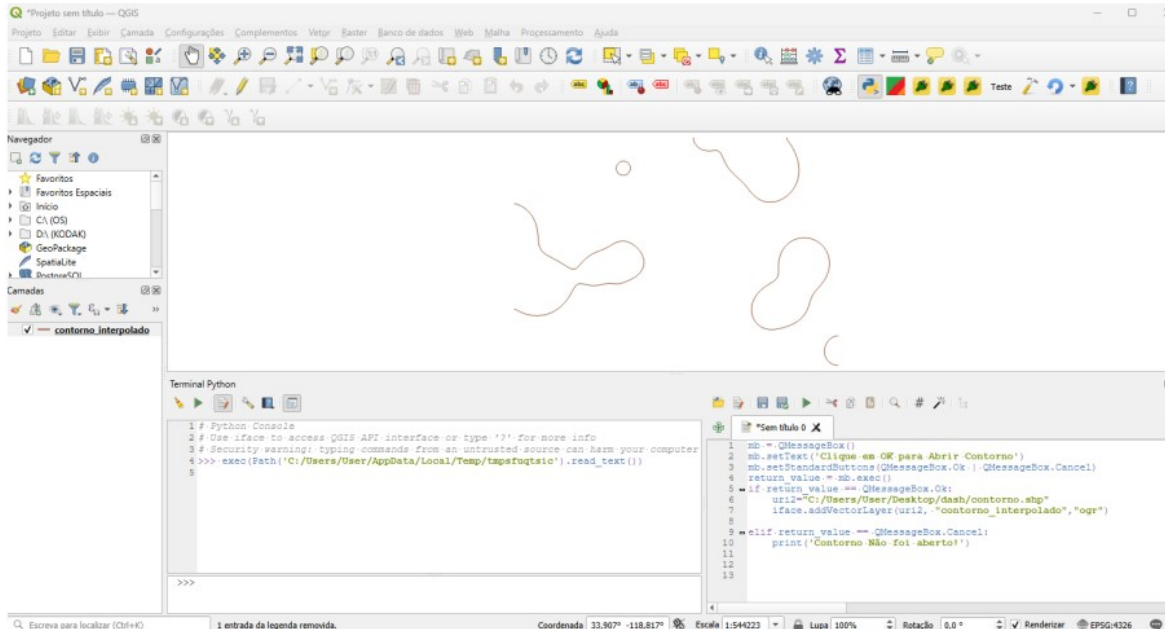
QMessageBox

Um exemplo de como usar a classe Qt diretamente no QGIS.

```
mb = QMessageBox()
mb.setText('Clique em OK para Abrir Contorno')
mb.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
return_value = mb.exec()
if return_value == QMessageBox.Ok:
    uri2="C:/Users/User/Desktop/dash/contorno.shp"
    iface.addVectorLayer(uri2, "contorno_interpolado", "ogr")
elif return_value == QMessageBox.Cancel:
    print('Contorno Não foi aberto!')
```



Ao clicar OK teremos:



Agora somente um exemplo mais elaborado para ilustrar como a interação entre QGIS e QT pode ser feita, mas não se preocupe, temos os Plugins para facilitar tudo. Isso é somente uma demonstração. Para os pligins usaremos o QtDesigner para criar as Interfaces Gráficas visualmente.

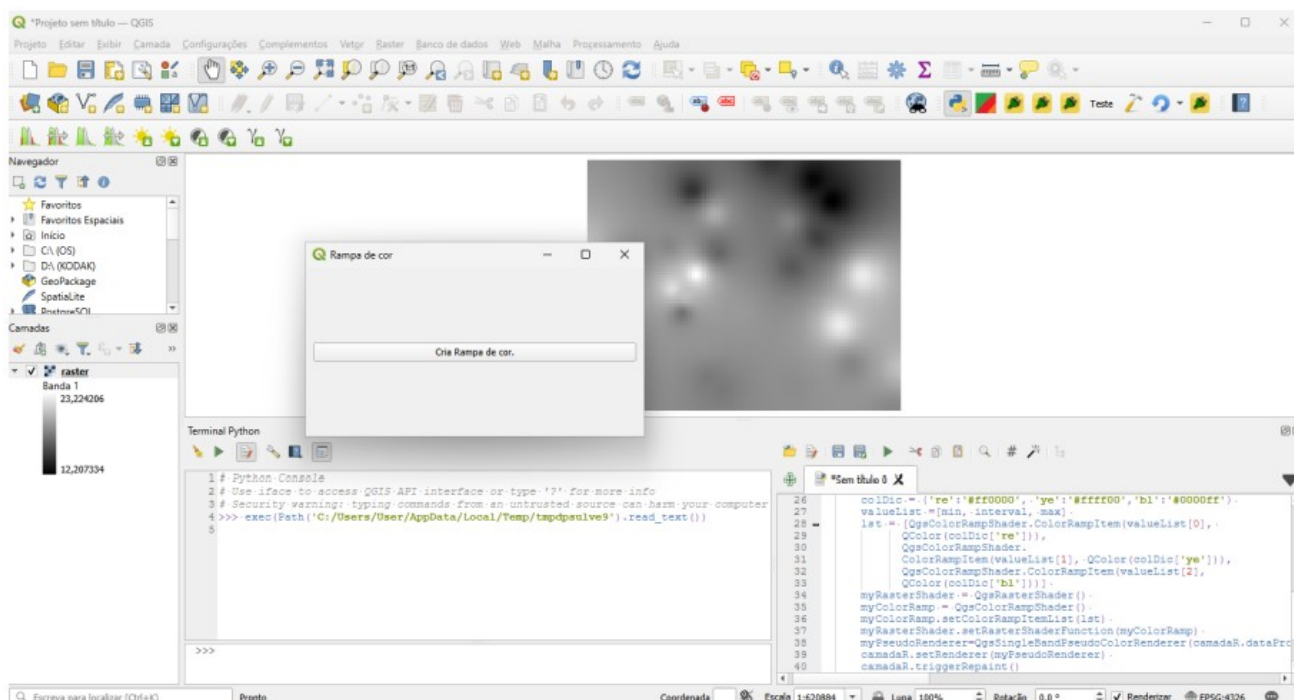
```
class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Rampa de cor")
        self.setGeometry(100, 100, 400, 200)
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        dialog_button = QPushButton("Cria Rampa de cor.")
        dialog_button.clicked.connect(self.rampa)
        layout = QVBoxLayout()
        layout.addWidget(dialog_button)
        central_widget.setLayout(layout)
        def rampa(self):
            camadaR = iface.activeLayer()
            renderer = camadaR.renderer()
            provider = camadaR.dataProvider()
            rect = camadaR.extent()
            stats = provider.bandStatistics(1,
            QgsRasterBandStats.All, rect, 0)
```

```

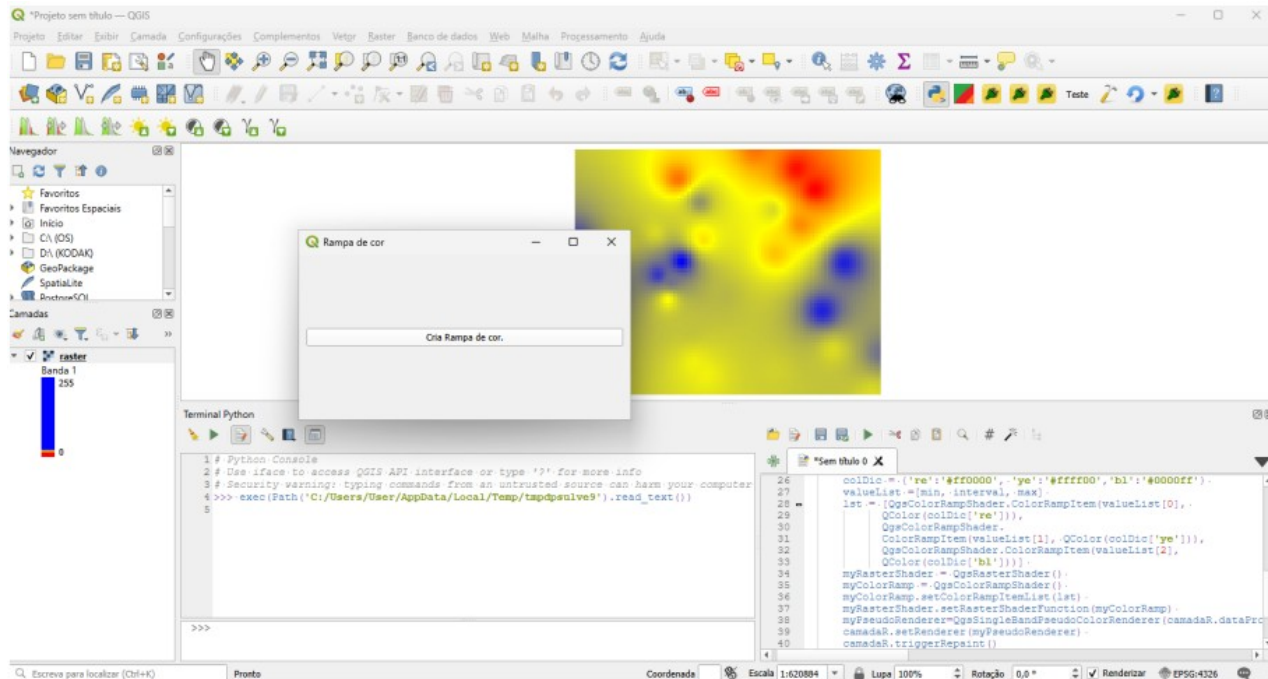
min= stats.minimumValue
max = stats.maximumValue
range = max - min
add = range//2
interval = min + add
colDic = {'re':'#ff0000', 'ye':'#ffff00','bl':'#0000ff'}
valueList =[min, interval, max]
lst = [QgsColorRampShader.ColorRampItem(valueList[0],
QColor(colDic['re'])),
QgsColorRampShader.
ColorRampItem(valueList[1], QColor(colDic['ye'])),
QgsColorRampShader.ColorRampItem(valueList[2],
QColor(colDic['bl']))]
myRasterShader = QgsRasterShader()
myColorRamp = QgsColorRampShader()
myColorRamp.setColorRampItemList(lst)
myRasterShader.setRasterShaderFunction(myColorRamp)
myPseudoRenderer=
QgsSingleBandPseudoColorRenderer(camadaR.dataProvider(),
camadaR.type(), myRasterShader)
camadaR.setRenderer(myPseudoRenderer)
camadaR.triggerRepaint()
arquivo = "c:/users/user/desktop/dash/rasterDeTeste.asc"
iface.addRasterLayer(arquivo, "raster")
window = MyWindow()
window.show()

```

Ao executar teremos:



E clicando no botão teremos:



Executando um script pyQGS fora do Qgis

Podemos executar processamentos do QGIS sem iniciar a interface gráfica, usando somente scripts. Para fazer isso temos que criar um script com a seguinte estrutura mínima.

```
from qgis.core import *
# indique onde o programa qgis está localizado no seu computador
QgsApplication.setPrefixPath("/usr", True)
# Crie uma referência à QgsApplication. Usando False como segundo
argumento
# para não ativar a interface gráfica.
qgs = QgsApplication([], False)
# inicie o qgis
qgs.initQgis()
#####
# Escreva o código de processamento aqui###
#####
# finalize o script usando:
qgs.exitQgis()
```

Vamos mostrar como isso funciona criando um script que criará um grid raster a partir de um arquivo texto CSV com alguns pontos. O mesmo procedimento do último exemplo do módulo anterior. Nomeie o arquivo de `criaRaster.py`. Substitua `users/user/desktop/dash` apropriadamente para o seu sistema.

```
from qgis.core import *
from qgis.analysis import *
import os
QgsApplication.setPrefixPath("C:/ProgramFiles/QGIS3.34.1/bin/",
True)
qgs = QgsApplication([], False)
```

```

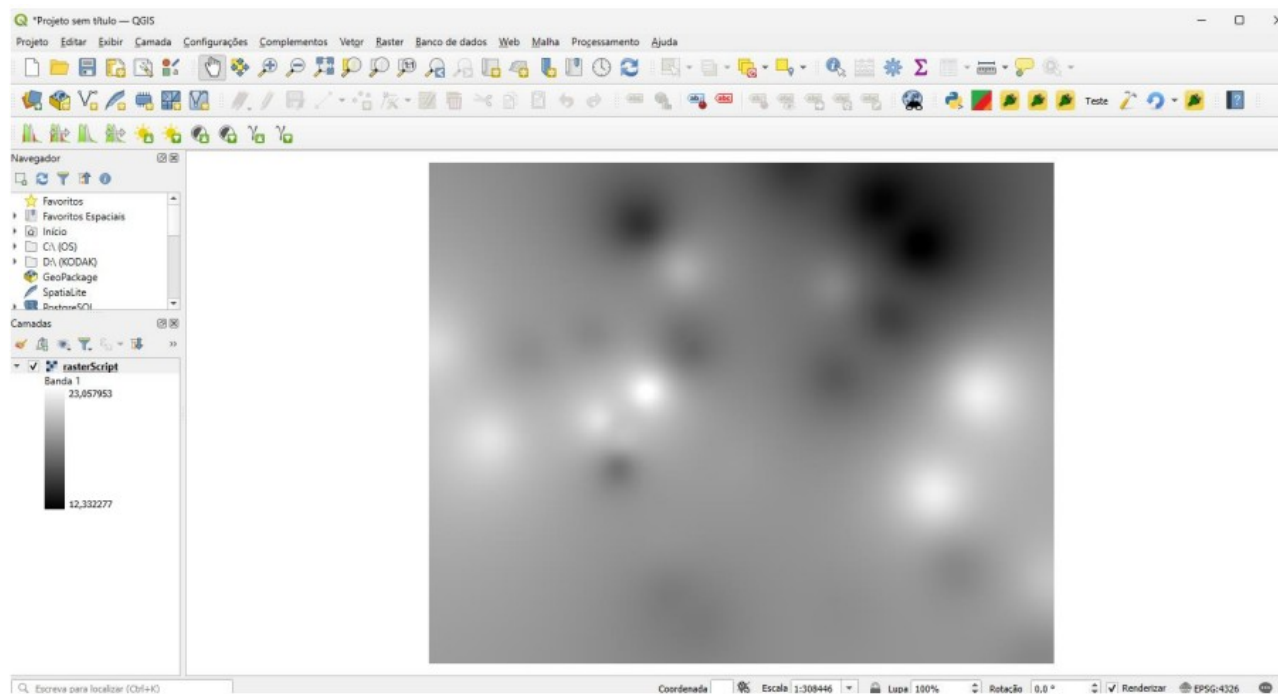
qgs.initQgis()
uri="file:///users/user/desktop/dash/tfinal.csv?type=csv&xField=LONG
ITUDE&yField=LATITUDE&crs=epsg:4326"
camada = QgsVectorLayer(uri, 'Converte', "delimitedtext")
if not camada.isValid():
    print("A camada não carregou apropriadamente!")
else:
    c_data = QgsInterpolator.LayerData()
    c_data.source = camada
    c_data.zCoordInterpolation = False
    c_data.interpolationAttribute = 6
    c_data.sourceType = QgsInterpolator.SourcePoints
    interpolado = QgsIDWInterpolator([c_data])
    interpolado.setDistanceCoefficient(2)
    arquivo = os.path.expanduser('~')+"/desktop/dash/rasterScript.asc"
    rect = camada.extent()
    res = 0.001
    ncol = int( ( rect.xMaximum() - rect.xMinimum() ) / res )
    nrows = int( (rect.yMaximum() - rect.yMinimum() ) / res)
    saida = QgsGridFileWriter(interpolado,arquivo,rect,ncol,nrows)
    saida.writeFile()
qgs.exitQgis()

```

Abrir o shell OSGeo4W, navegar até a pasta do script criaRaster.py e executar usando:

```
python-qgis criaRaster.py
```

Ao abrirmos o raster rasterScript.asc criado pelo script acima no QGIS veremos o seguinte



Colinha do pyQGS

A seguir temos uma cola (cheat sheet) das classes principais para uma rápida referência. Em azul temos as bibliotecas a serem carregadas no caso de usar em Plugins ou em scripts fora do QGIS.

QgsInterface() iface

Aparência

```
from qgis.PyQt.QtWidgets import QApplication
app = QApplication.instance()
app.setStyleSheet(".QWidget {color:blue;background-color:yellow;}")
```

Você pode carregar o stylesheet de um arquivo com

```
from qgis.PyQt.QtWidgets import QApplication
app = QApplication.instance()
with open("testdata/file.qss") as qss_file_content:
app.setStyleSheet(qss_file_content.read())
```

Mudando o ícone e título do programa

```
from qgis.PyQt.QtGui import QIcon
icon = QIcon("/caminho/para/ele/icon.png")
iface.mainWindow().setWindowIcon(icon)
iface.mainWindow().setWindowTitle("ArcGIS só que não....")
```

Barra de Ferramentas

Remove e adiciona barra de ferramenta

```
toolbar = iface.helpToolBar()
parent = toolbar.parentWidget()
parent.removeToolBar(toolbar)
parent.addToolBar(toolbar)
```

Remove ações da barra de ferramenta

```
actions = iface.attributesToolBar().actions()
iface.attributesToolBar().clear()
iface.attributesToolBar().addAction(actions[4])
iface.attributesToolBar().addAction(actions[3])
```

Menus

Remove e adiciona Menu

```
menu = iface.helpMenu()
menubar = menu.parentWidget()
menubar.removeAction(menu.menuAction())
menubar.addAction(menu.menuAction())
```

Canvas (tela do mapa)

Acessando o canvas

```
canvas = iface.mapCanvas()
```

Mudando a cor do canvas

```
from qgis.PyQt.QtCore import Qt
iface.mapCanvas().setCanvasColor(Qt.black)
iface.mapCanvas().refresh()
```

Mudando intervalo de atualização do mapa

```
from qgis.core import QgsSettings
# para 150 milisegundos
QgsSettings().setValue("/qgis/map_update_interval", 150)
```

Camadas

Adicionando uma camada vetor do tipo shapefile

```
camada = iface.addVectorLayer("caminho/aoi.shp", "Camada", "ogr")
if not camada or not camada.isValid():
    print("Falha ao carregar camada!")
```

Adicionando uma camada raster do tipo tif

```
from qgis.core import QgsRasterLayer
arquivoR = "caminho/srtm.tif"
camada = QgsRasterLayer(arquivo, "Raster")
if not camada.isValid():
    print("Falha ao carregar camada!")
```

Obtendo a camada ativa

```
camada = iface.activeLayer()
```

Listando todas as camadas

```
from qgis.core import QgsProject
QgsProject.instance().mapLayers().values()
```

Obtendo o nome das camadas

```
from qgis.core import QgsProject
nomes = [layer.name() for layer in QgsProject.instance().mapLayers().values()]
print("Camadas = {}".format(nomes))
```

Obtendo camada pelo seu nome e fazendo ela ativa

```
from qgis.core import QgsProject
camada = QgsProject.instance().mapLayersByName("NomeCamada")[0]
iface.setActiveLayer(layer)
```

Adicionando um novo elemento

```
from qgis.core import QgsGeometry, QgsPointXY, QgsFeature
pr = camada.dataProvider()
feat = QgsFeature()
feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(10,10)))
pr.addFeatures([feat])
```

Obtendo os elementos

```
for f in camada.getFeatures():
    print (f)
```

Obtendo elementos selecionados

```
for f in camada.selectedFeatures():
    print (f)
```

Obtendo o ID dos elementos selecionados

```
selected_ids = camada.selectedFeatureIds()
print(selected_ids)
```

Criando uma camada na memória com os elementos selecionados

```
from qgis.core import QgsFeatureRequest
camadam=
layer.materialize(QgsFeatureRequest().setFilterFids(layer.selectedFeatureIds()))
QgsProject.instance().addMapLayer(camadam)
```

Obtendo a geometria da camada

```
# Camada tipo Point
for f in camada.getFeatures():
    geom = f.geometry()
print ('%f, %f' % (geom.asPoint().y(), geom.asPoint().x()))
```

Transladando uma geometria

```
from qgis.core import QgsFeature, QgsGeometry
f = QgsFeature()
geom = QgsGeometry.fromWkt("POINT(7 45)")
geom.translate(1, 1)
f.setGeometry(geom)
print(f.geometry())
```

Assinalando o Sistema de referência de coordenadas (CRS) de uma camada

```
from qgis.core import QgsProject, QgsCoordinateReferenceSystem
for camada in QgsProject.instance().mapLayers().values():
    camada.setCrs(QgsCoordinateReferenceSystem('EPSG:4326'))
```

Checando o CRS de uma camada

```
from qgis.core import QgsProject
for camada in QgsProject.instance().mapLayers().values():
    crs = camada.crs().authid()
    camada.setName('{} ({} )'.format(camada.name(), crs))
```

Uma função para ocultar uma coluna de atributo

```
from qgis.core import QgsEditorWidgetSetup
def fieldVisibility (camada,fname):
    setup = QgsEditorWidgetSetup('Hidden', {})
    for i, column in enumerate(layer.fields()):
        if column.name()==fname:
            camada.setEditorWidgetSetup(idx, setup)
            break
    else:
        continue
```

Criando camada na memória a partir de WKT

```
from qgis.core import QgsVectorLayer,QgsFeature,QgsGeometry,QgsProject
camada = QgsVectorLayer('Polygon?crs=epsg:4326', 'MS', 'memory')
pr = camada.dataProvider() f = QgsFeature()
geom = QgsGeometry.fromWkt("POLYGON ((-88.82 34.99,-88.09 34.89,-
88.39 30.34,-89.57 30.18,-89.73 31,-91.63 30.99,-90.87 32.37,-91.23
33.44,-90.93 34.23,-90.30 34.99,-88.82 34.99))")
poly.setGeometry(geom)
pr.addFeatures([f])
camada.updateExtents()
QgsProject.instance().addMapLayers([camada])
```

Lendo as camadas de um GeoPackage de vetores

```
from qgis.core import QgsDataProvider
arquivo = "caminho/seupacote.gpkg"
camada = QgsVectorLayer(arquivo, "teste", "ogr")
subCamadass = camada.dataProvider().subLayers()
for subCamada in subCamadass:
    nome = subCamada.split(QgsDataProvider.SUBLAYER_SEPARATOR)[1]
    uri = "%s|layername=%s" % (arquivo, nome,)
    # Cria a Camada
    subCamadav = QgsVectorLayer(uri, nome, 'ogr')
    # Adiciona a camada ao mapa
    QgsProject.instance().addMapLayer(subCamadav)
```

Adicionando uma camada Tile (Camada XYZ)

```
from qgis.core import QgsRasterLayer, QgsProject
def loadXYZ(url, name):
    rasterLyr = QgsRasterLayer("type=xyz&url=" + url, name, "wms")
    QgsProject.instance().addMapLayer(rasterLyr)
url= 'https://tile.openstreetmap.org/{z}/{x}/{y}.png&zmax=19&zmin=0'
loadXYZ(url, 'OpenStreetMap')
```

Adicionando uma camada de um banco de dados Postgis

```
from qgis.core import QgsVectorLayer, QgsProject, QgsDataSourceUri
uri = QgsDataSourceUri()
uri.setConnection("servidorDNS", "5432", "dbase", "user", "pswrd")
uri.setDataSource("schema", "tablename", "geom")
camada = QgsVectorLayer(uri.uri(False), "Remota", "postgres")
QgsProject.instance().addMapLayer(rasterLyr)
```

Removendo todas as camadas do Projeto

```
QgsProject.instance().removeAllMapLayers()
```

Removendo tudo do projeto

```
QgsProject.instance().clear()
```