

Introduction to pyQGIS



Module 2

Data

This publication and the data are available at:

<https://gdatasystems.com/repo/pyQGIS1.pdf>

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

VISIT:

<https://gdatasystems.com>

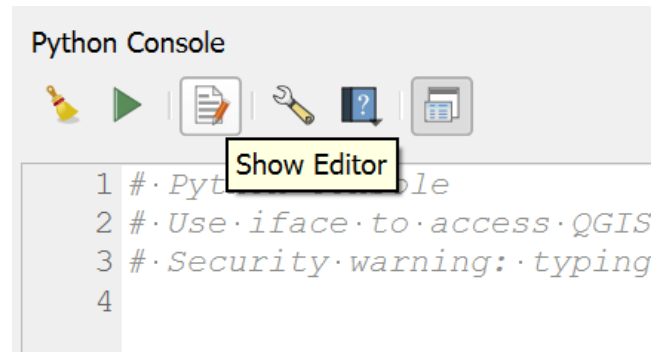
License : CC BY 4.0

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

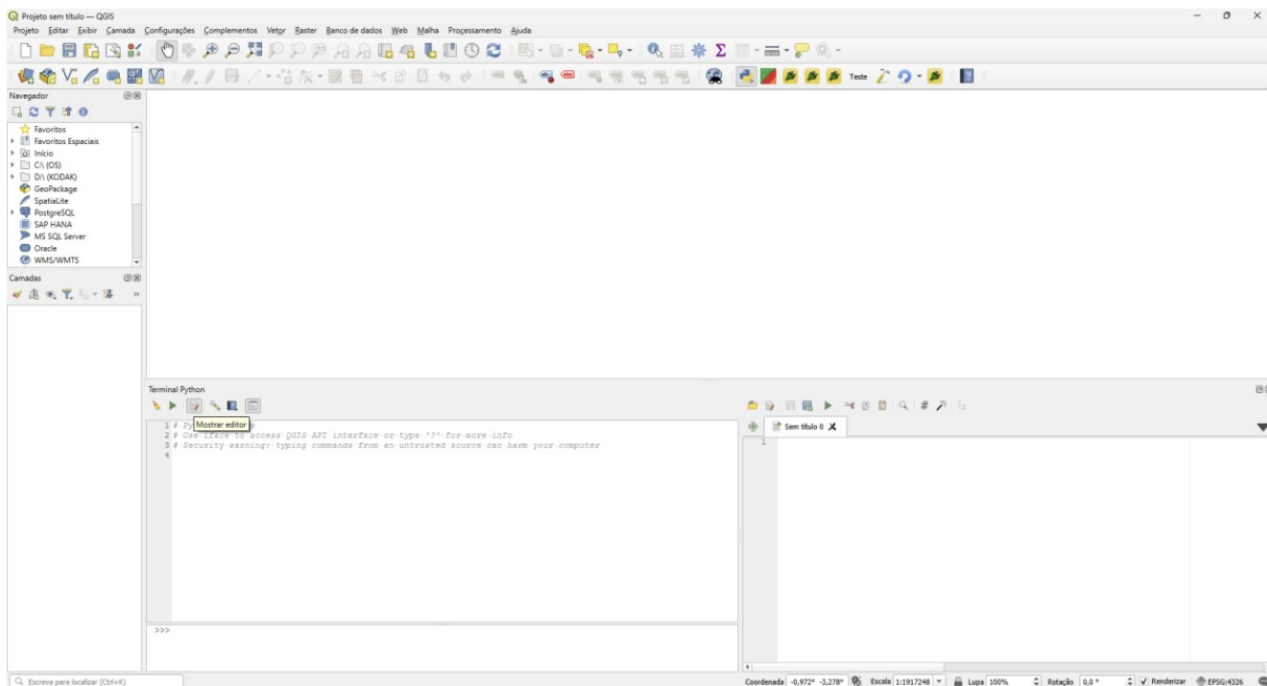
Module 2 – Using pyQGIS: Introduction to the first classes

Preparing the Ground

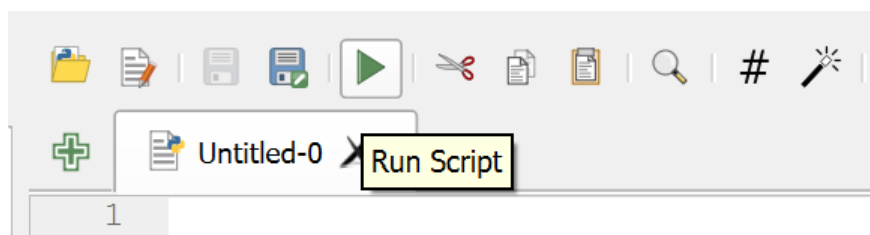
Let's open QGIS and get everything ready to work within the environment using the Python terminal (CTRL+ALT+P), then click the "Show editor" button:



We will see the screen like this.



We are ready to work. In the editor at the bottom right, we will write the code and execute it using:



pyQGIS Classes

We will introduce PyQGIS classes as we progress through the topics covered.

In Python, a class is the definition of an object type and the methods (functions) associated with that object—for example, a project, a raster layer, a vector layer, etc.

The PyQGIS library is quite extensive, containing a wide variety of classes. Here, we will cover some of the most fundamental and essential classes, providing a solid foundation for developing more complex scripts.

Project Class (QgsProject) - Creating and Reading a Project

A project stores a collection of information regarding layers, styles, layouts, annotations, etc. Since it is a singleton class, we create an object using the `QgsProject.instance()` method. We will demonstrate how to create an empty project named `meu_projeto.qgs` using the `write()` method.

```
projeto= QgsProject.instance()
# adjust the path for the file to be created on your system
projeto.write('C:/Users/User/Desktop/dash/meu_projeto.qgs')
print(projeto.fileName())
C:/Users/User/Desktop/dash /meu_projeto.qgs
```

Now, let's exit QGIS and reopen it so we can load the project we created using 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
```

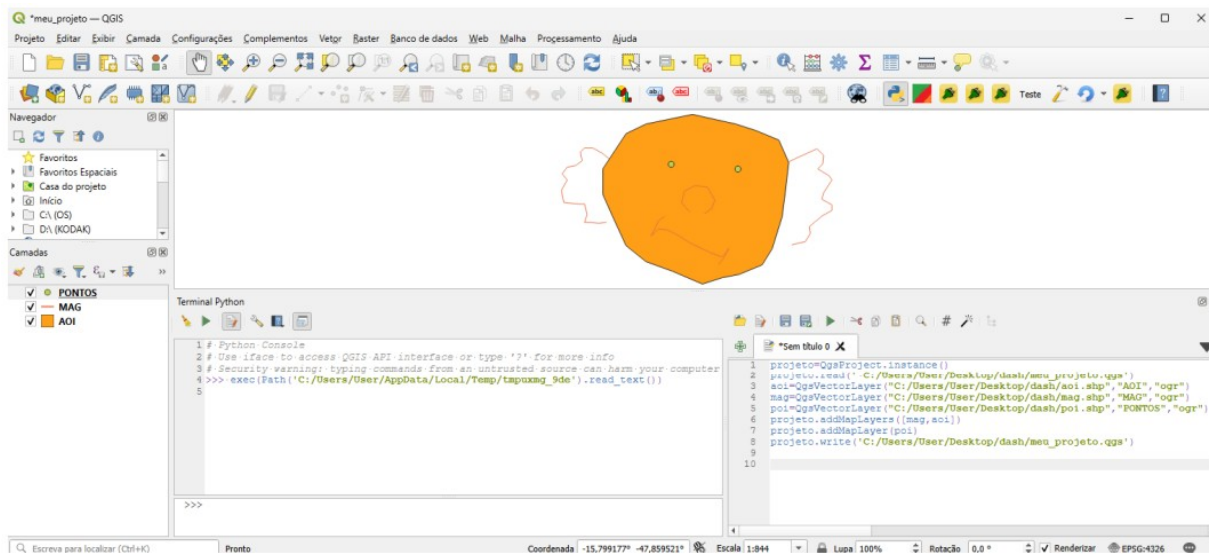
As we look at the other classes, we will see other methods associated with the `Projeto` class.

Vector Class (QgsVectorLayer) – Adding a vectorial layer

A vector layer object is used to load and interact with vector-type layers. We will load our project and add to it three vector layers—created previously—using the project methods `addMapLayer` and `addMapLayers`. We create a vector layer object using the `QgsVectorLayer()` class constructor, passing the path to the vector file, the identifier to be assigned to the layer, and the library to be used (in this case, `ogr`). Finally, we save our project using the `write()` method.

```
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')
```

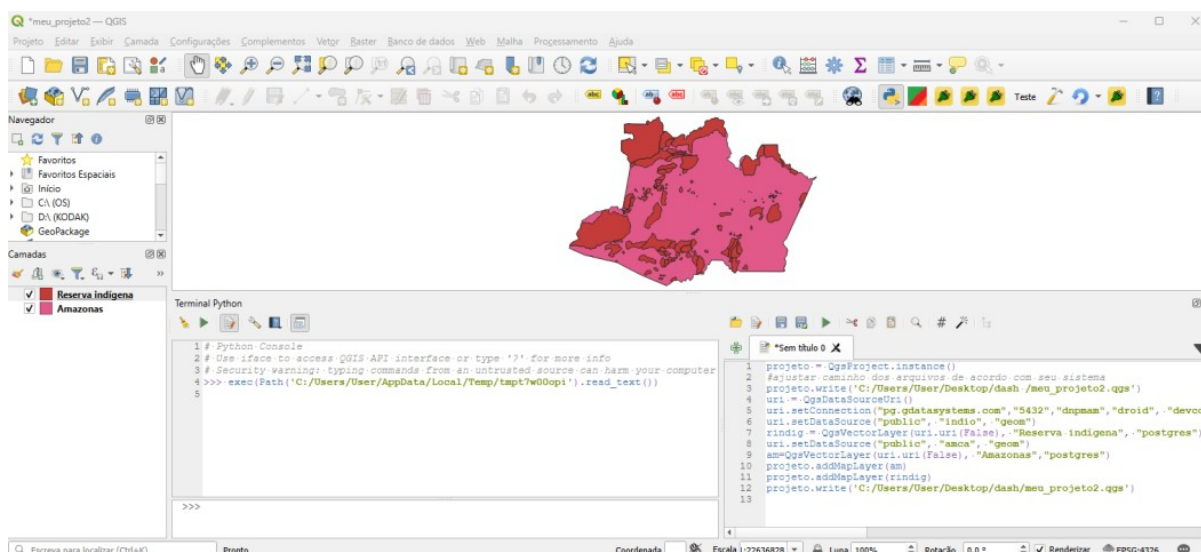
Something similar to what is shown below will appear:



Now, we will use another class to load a vector layer located in a remote PostGIS database. This class is `QgsDataSourceUri`, and we will employ its `setConnection()` and `setDataSource()` methods to retrieve a spatial vector table. We will create a new project, add both a local layer and a remote PostGIS layer, and finally, save the project.

```
projeto = QgsProject.instance()
#adjust the path for the file to be created on your system
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')
```

The `setConnection()` method takes the following parameters: server address (IP or DNS), port (typically 5432), database, username, and password. The `setDataSource()` method takes the following parameters: table schema, table name, and the column containing the spatial geometric element. A project, as illustrated below, will appear.



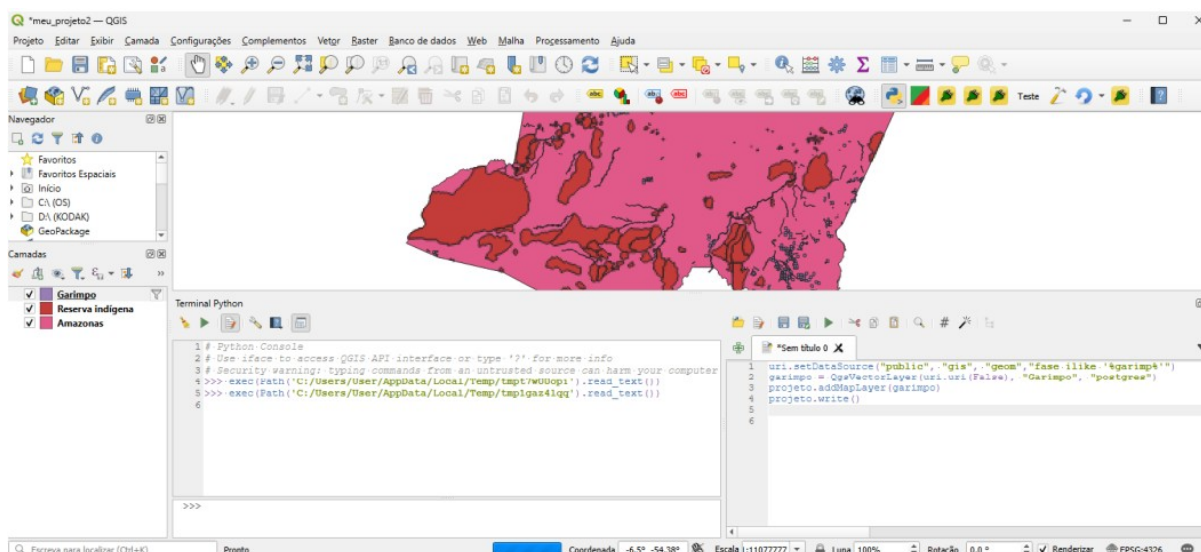
Alternatively, we can add a SQL WHERE clause as the fourth argument.

Let's look at an example where we extract—from a vector layer containing applications for small-scale mining (garimpo) and mining permits in the state of Amazonas—only those specific types of applications, and then add this result to the project we created above.

```

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

```



Before we move on to the next topic, let's take a look at some methods in the Project class (QgsProject) that relate to the Vector Layers class (QgsVectorLayer).

count returns the number of valid layers in the project.

```

projeto.count()
3

```

mapLayers returns a map of the project's existing layers.

```
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 indigena' (postgres)>}
```

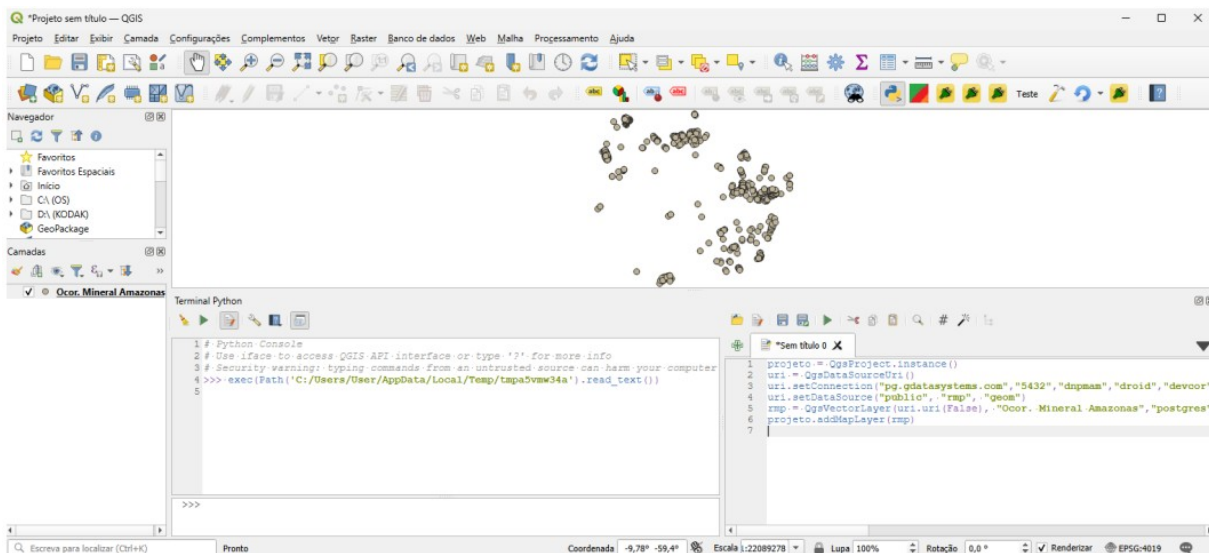
Interacting with information from objects of the Vector class

We can obtain various information about vector objects such as projections, extent, number of elements, and the values and names of attribute fields (columns) and even create layer metadata (using the existing information).

First, let's load point-type data from a remote PostGIS database..

```
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)
```

The layer will be loaded as shown in the illustration below.



Let's now access the information that is most commonly used. Other information is available within the layer object; see the documentation for further details..

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

The `crs()` method returns the original coordinate reference system of the layer object that invokes it.

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

Using the `extent()` method of a layer object, we can obtain the maximum and minimum coordinate values for X (Easting or Longitude) and Y (Northing or Latitude). The method returns a rectangle-type object that includes various parameters—beyond just the maximum and minimum X and Y values—such as area, width, height, center, invert, etc. Refer to the documentation for further information.

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

The `featureCount()` method returns how many items the layer object contains.

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

With the `fields()` method, we obtain information about all attribute fields, such as name, type, 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_uti 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
```

The `htmlMetadata()` method generates layer metadata in HTML format, which can be copied into a new file and viewed in a web browser:

```
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>
```

This HTML information above would appear in a browser like this:

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.

Using the `getFeatures()` method, we load all the data from the layer, with each item stored as a list. The code below prints each of these items in list format.

```

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']

```

Creating a vector object

Let's now look at the steps for creating vector objects using Python in QGIS. We will create point, line, and polygon objects to illustrate the process.

Point

First, we define a point object with CRS 4326 (WGS84), named "Cidades" (Cities), in memory. For this object, we use a `dataProvider` to create attribute fields for the point vector object specifically three attributes (name, population, and HDI) and add them to the point object (`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()

```

Once the point object and its attribute fields have been created, we will add data to it using a feature object. We define the geometry, in this case, a point with x and y coordinates, and then add the attributes for this point.

```

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)

```

To finish, we will configure the appearance of the symbol shown on the map as orange-colored stars with a size of 8. We will update the map extent and add our object to the map.

```

symbol = QgsMarkerSymbol.createSimple({'name': 'star',
    'color':'orange','size':'8'})

```

See the result below.



Line

We define a line object (linestring) with CRS 4326 (WGS84), named "Vias," in memory. We create a dataProvider to add attribute fields to the linestring vector object—specifically two attributes (name and number)—and add them to the linestring object (vlinha).

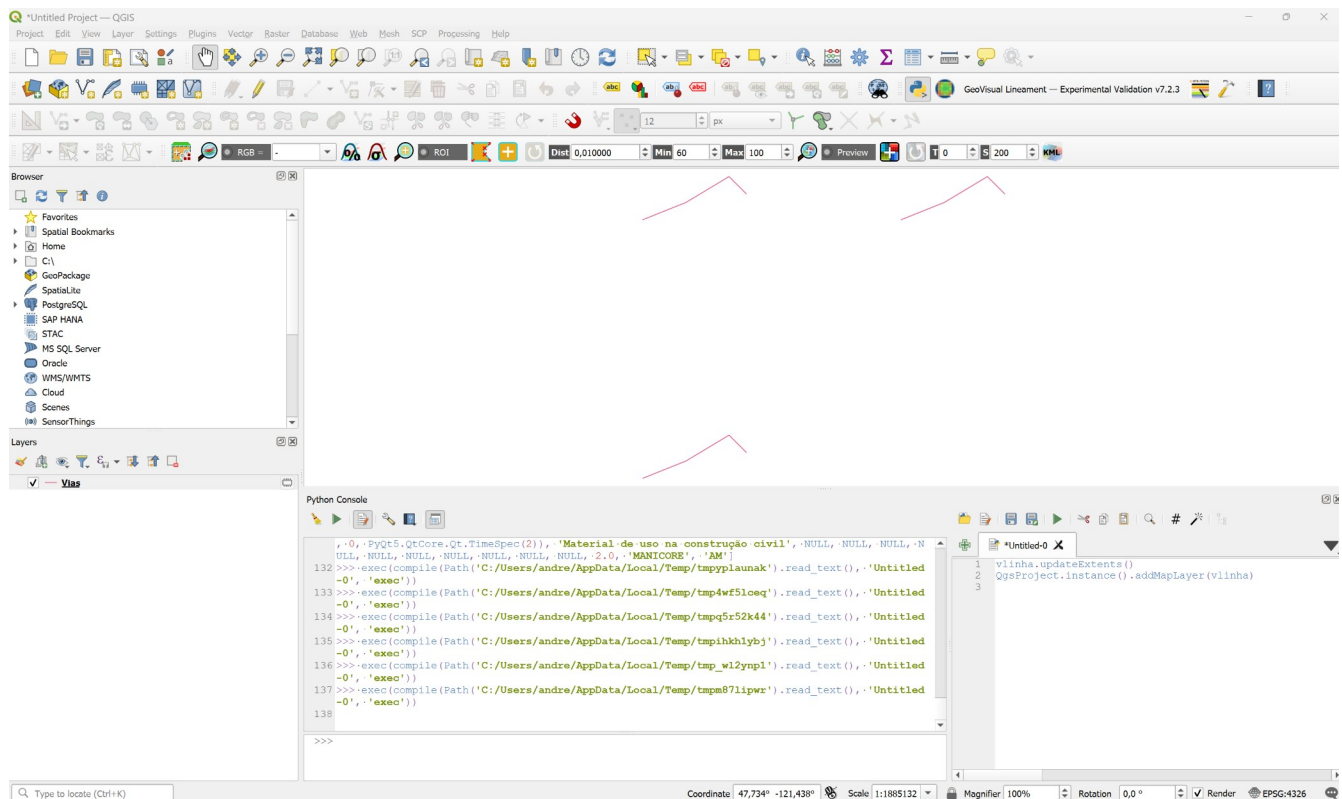
```
vlinha = QgsVectorLayer("Linestring?crs=EPSG:4326", "Vias", "memory")
dPr = vlinha.dataProvider()
dPr.addAttributes([QgsField("nome", QVariant.String), QgsField("número", QVari
ant.Int)])
vlinha.updateFields()
```

We add the lines using a feature object. First, however, we create the list of points that will form part of each line-type feature, and we include the attributes for each feature. We will add three lines.

```
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)
```

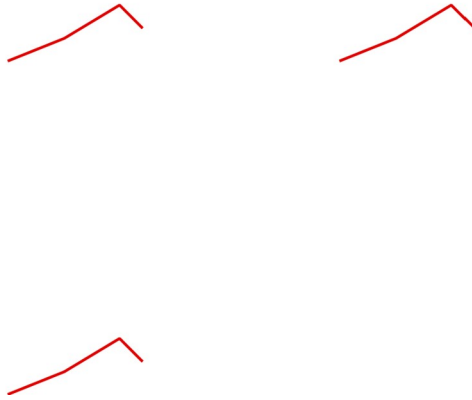
We updated the object's extent and added it to the map (canvas).

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



We can change the appearance of our vector linestring object using:

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



To conclude, we will save the newly created vector to a shapefile.

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

Polygon

We create polygons using Python/QGIS in the same way we create lines, with the difference that, in this case, the last point connects to the first point provided. We define the polygon object, named "Fazendas", in memory, using the CRS 4326 (WGS84) coordinate reference system.

We created the dataProvider to add attribute fields—specifically "name" and "number"—to the polygon vector object, and then added these fields to the polygon object (vpgon).

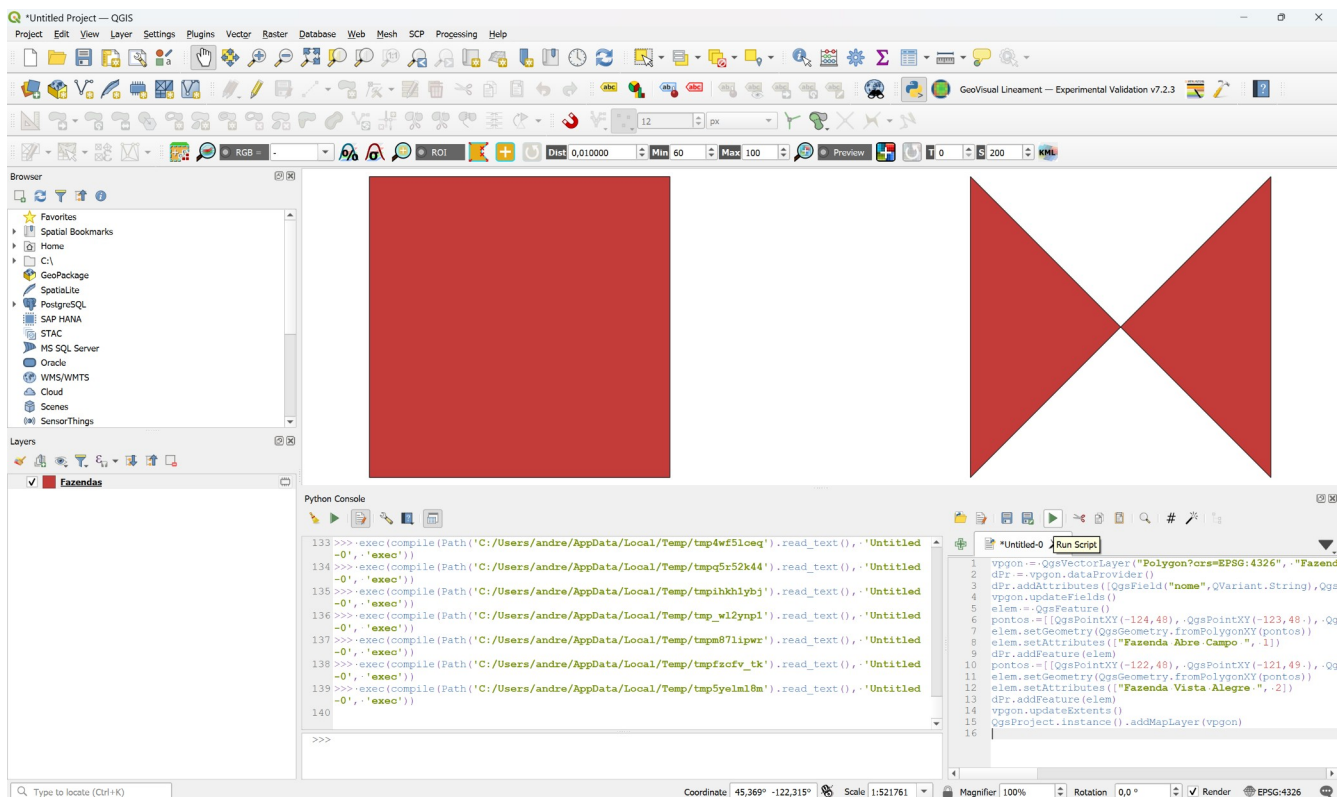
```
vpgon = QgsVectorLayer("Polygon?crs=EPSG:4326", "Fazendas", "memory")
dPr = vpgon.dataProvider()
dPr.addAttributes([QgsField("nome", QVariant.String), QgsField("número",
Variant.Int)])
vpgon.updateFields()
```

We add the polygons using a feature object. First, however, we create a list of points—enclosed in double brackets—that will form part of each polygon feature, and we then insert the attributes for each feature. We will add two polygons.

```
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)
```

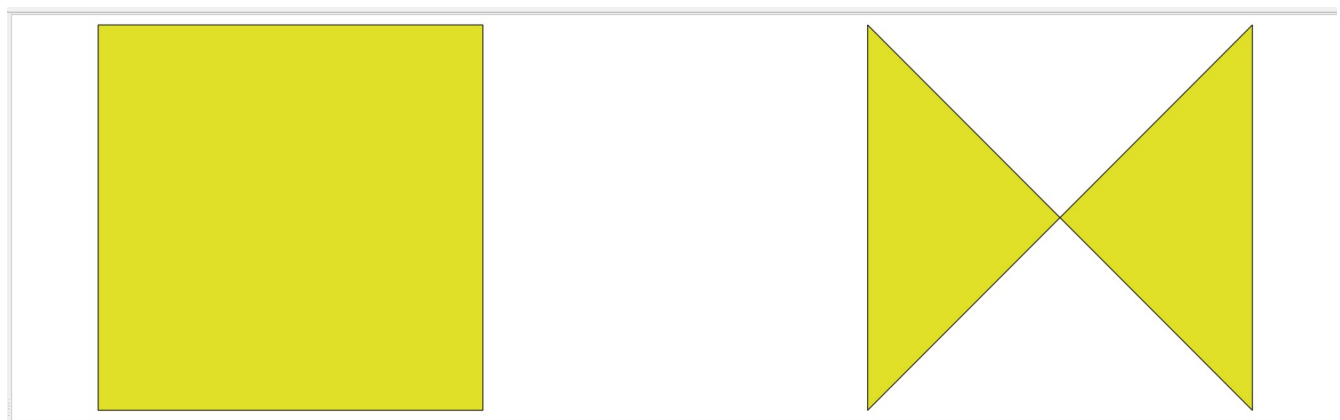
We updated the object's extent and added it to the map (canvas).

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



We can change the appearance of our polygon vector object using:

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



We save the newly created vector to a shapefile.

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

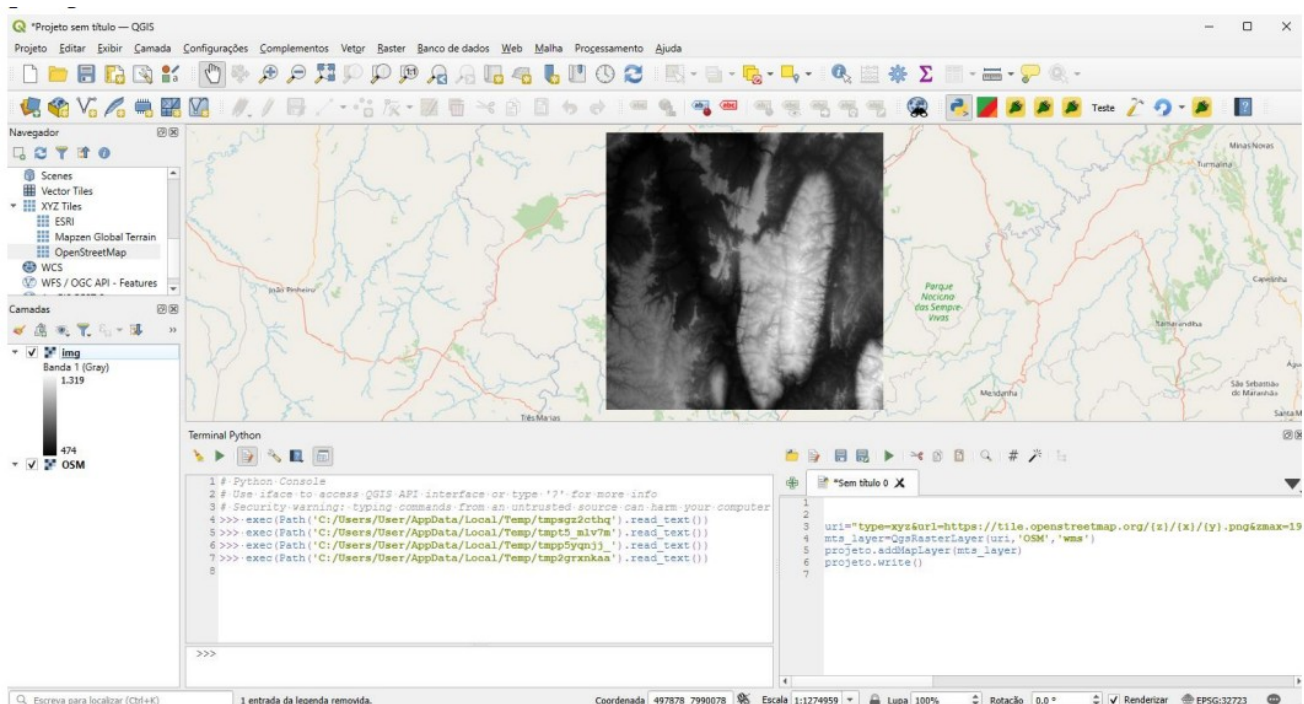
Classe Raster (QgsRasterLayer)

Much like how we add vector layers, we can add raster images to our project using an object from the raster class. Let's create a project and add a raster image to it.

```
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')
```

We can also add raster-type data using providers such as TMS (Tile Map Service) or WMS (Web Map Service).

```
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()
```

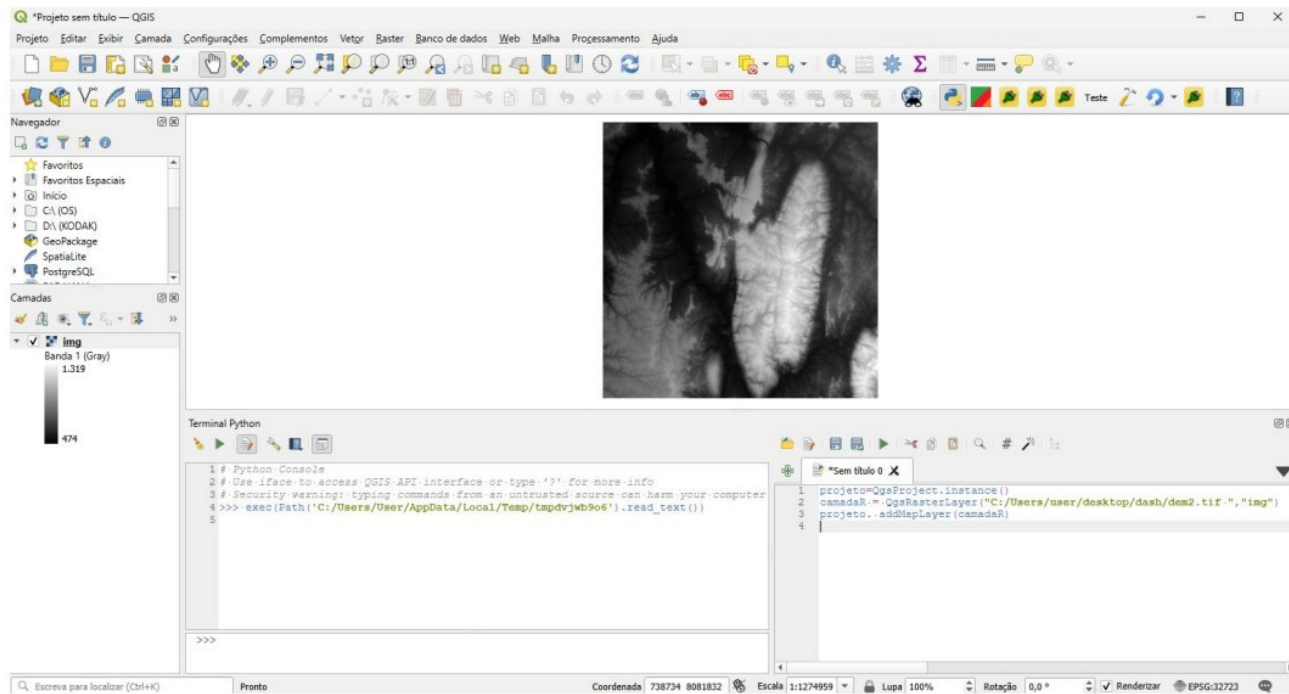


Interacting with information from objects of the Raster class

We can also extract relevant information from a raster object, such as dimensions, resolution, number of bands, the value of a pixel, etc. Let's start by loading an image.

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

This will open the image as shown below:



Raster object dimension information

We can access information regarding the dimensional parameters of a raster image using specific methods for this purpose.

```
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
```

This important information about the raster can be used for future spatial analyses. There are other ways to use these methods to obtain the same information.

We can calculate the resolution in X using the code below instead of using the `rasterUnitsPerPixelX()` method:

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

We can also view information using the `htmlMetadata()` method.

```
camadaR.htmlMetadata()
```

HTML text generated above, as viewed in the browser.

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 with a single band of values

Let us now look at methods for single-band rasters. The methods below indicate the number of bands and the type of the raster image: 0 for single-band grayscale or undefined, 1 for single-band paletted, and 2 for multiband.

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

The `dataProvider()` function acts as an interface between the raster object and its individual data; its `sample()` method takes two arguments: a point object (XZ coordinates) and the band number. If the coordinate lies within the image and the band exists, the result is a tuple containing the pixel value and a boolean indicating whether the data is valid.

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

The color ramp assigned to a raster object can be checked using the `type()` method of the `renderer()` method. The `singlebandgray` type is the default setting.

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

We can modify the color ramp using Python; the process is shown below. It involves creating a `'ColorRampShader'` object and setting the fill color ramp to an interpolated type.

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

We now create a list containing the colors that represent the two extreme values of the raster (0 and 2046), which will be interpolated between blue and yellow. Next, we add this list as an item to the `ColorRampShader` created above.

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

The next step is to create the `RasterShader` (color filler) and associate the `RampShader` with it.

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

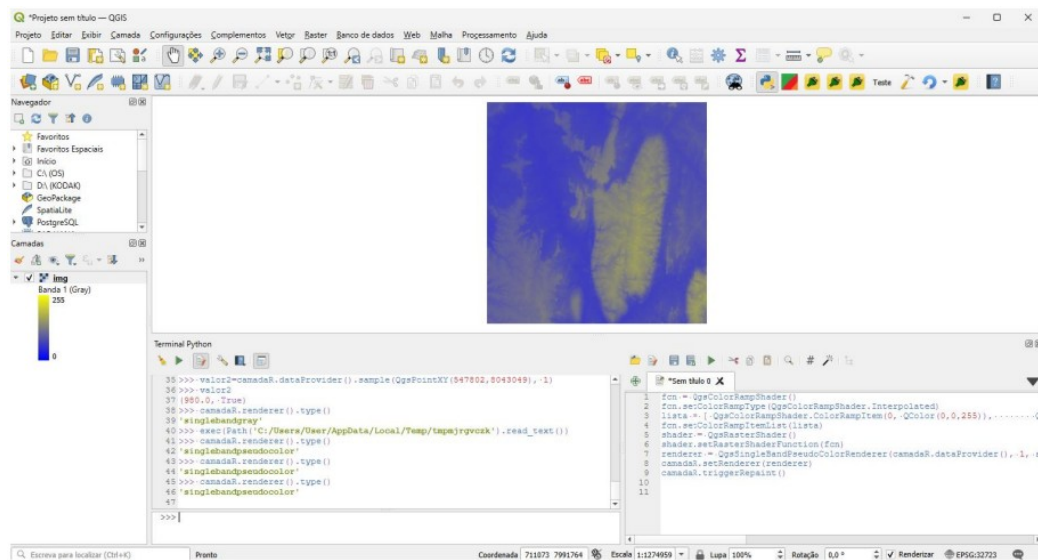
Finally, we create the color renderer object using the raster object data, band 1, and the shader mentioned above. We then apply this to the raster object and trigger a repaint of the object.

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

If we call the type again, we can see the change.

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

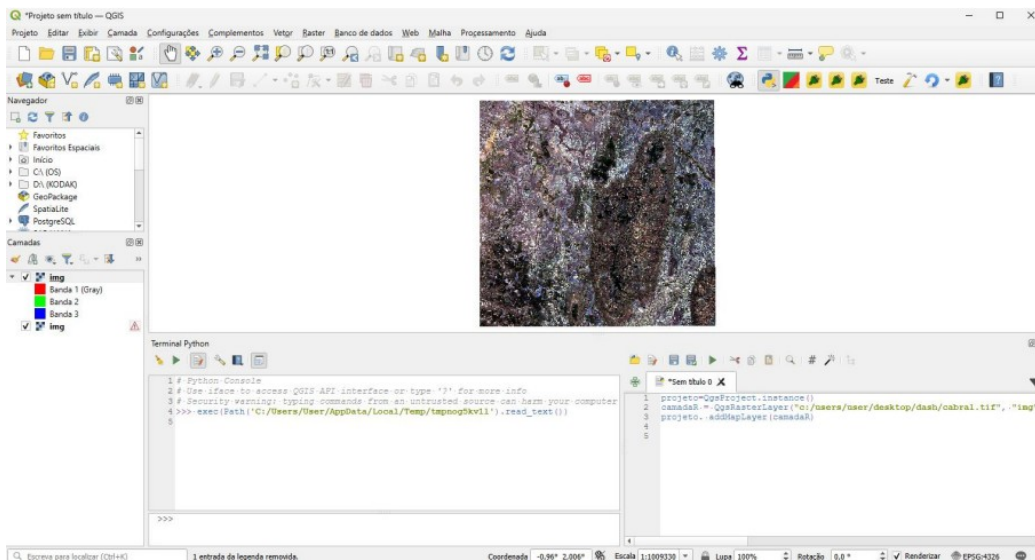
O resultado é mostrado na imagem abaixo.



Raster with more than one band of values

Now, let's look at methods for multi-band rasters. We will work with a 3-band raster image. We'll load the raster in a similar way and then extract some information from it.

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



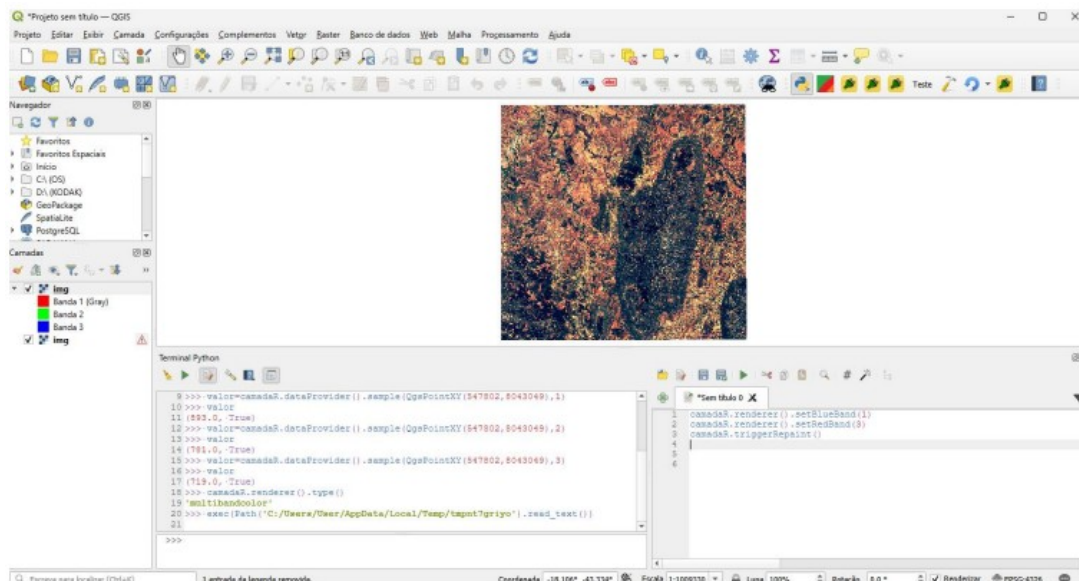
We can see some of the information by using:

```
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'
```

Below, we will see how to modify the image so that band 1 is placed in the blue (B) channel and band 3 is placed in the red (R) channel.

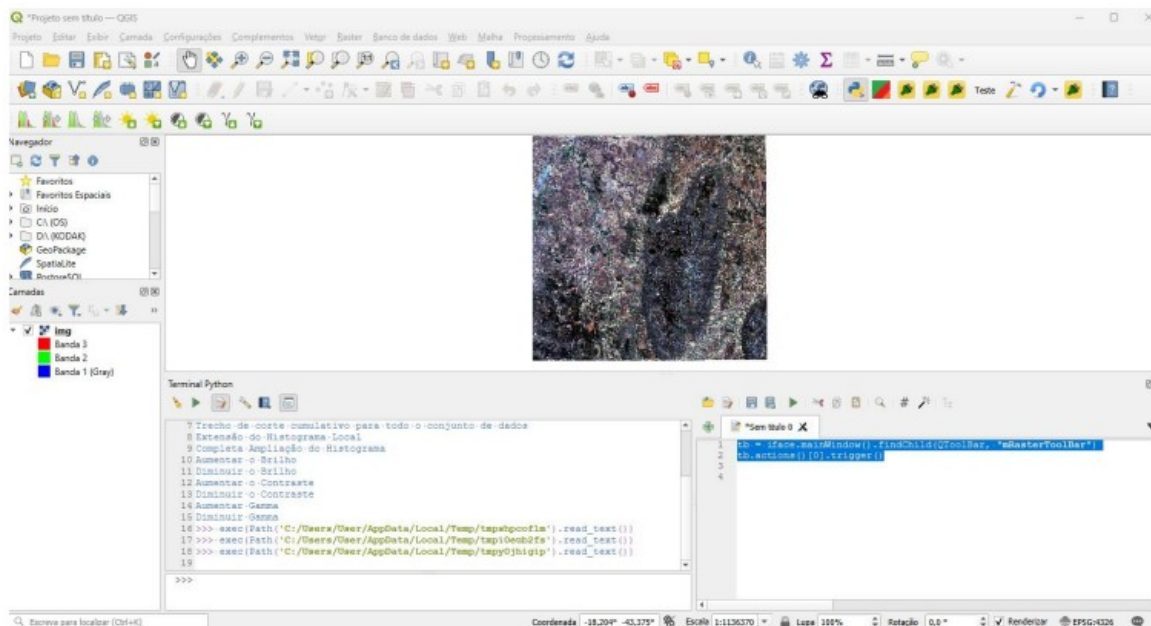
```
camadaR.renderer().setBlueBand(1)
camadaR.renderer().setRedBand(3)
camadaR.triggerRepaint()
```

Note that the image histogram has not been properly adjusted, as it still uses the maximum and minimum values from the previous bands.



To adjust, run:

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



Creating a raster object

Raster images can also be created very efficiently via script, using a list of point data with associated values. In this example, we will create a raster showing the average temperature of an area, based on point-based information from various locations. The CSV file `tfinal.csv` contains the data, including coordinates and their corresponding values. We will load this information into a `QgsInterpolator` object (referred to as `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
```

We will perform the interpolation using inverse distance weighting (IDW) squared (coefficient 2).

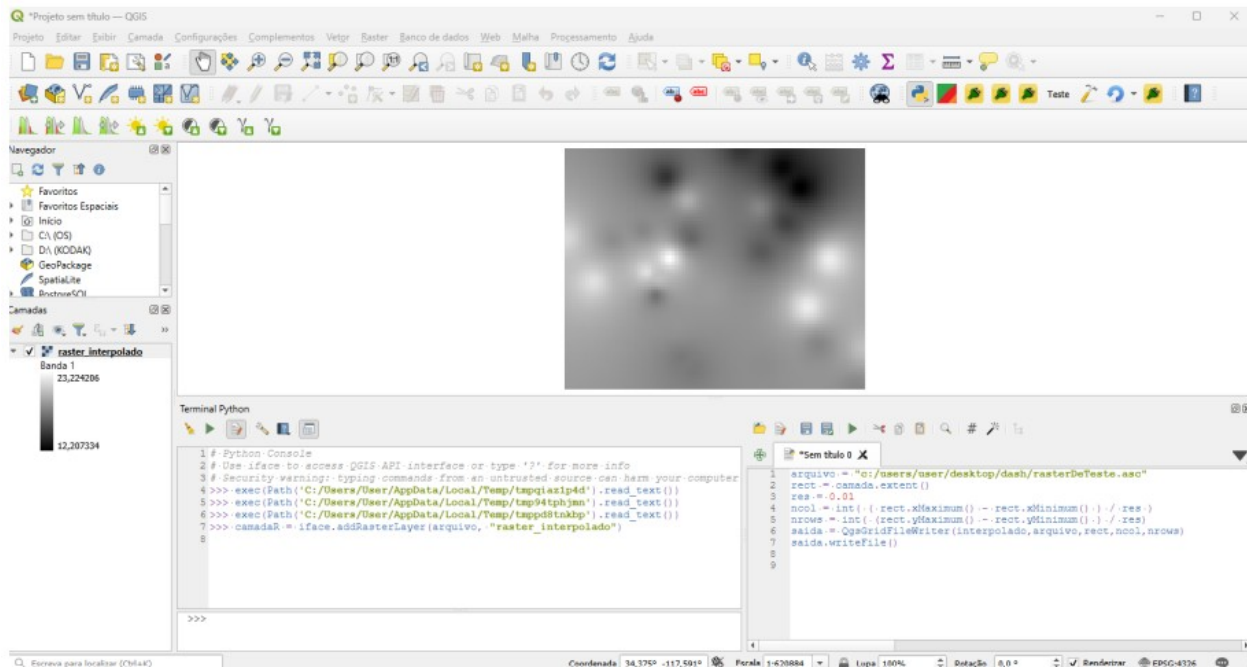
```
interpolado = QgsIDWInterpolator([c_data])
interpolado.setDistanceCoefficient(2)
```

Now we define which file will be created and the parameters for the grid to be used.

```
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()
```

We load the grid file using.

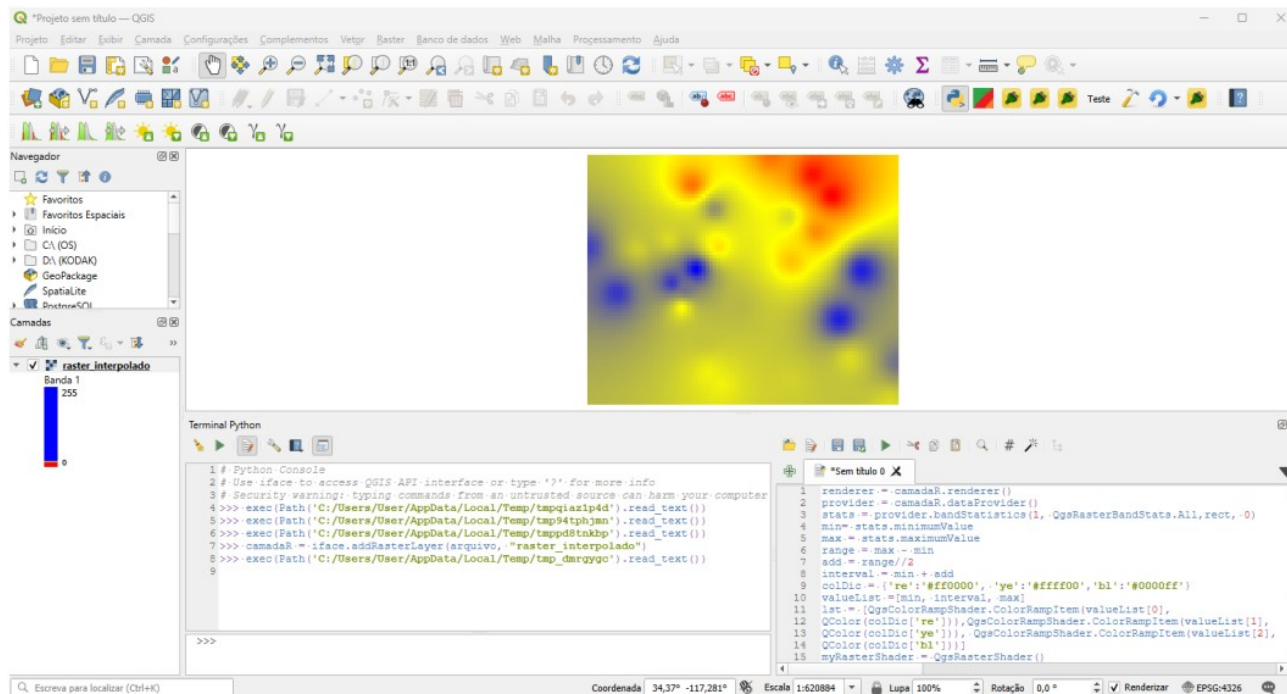
```
camadaR = iface.addRasterLayer(arquivo, "raster_interpolado")
```



We can change the appearance of the raster we created using the following code.

```
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()
```

The result of the created color ramp, when applied to the raster, will be:



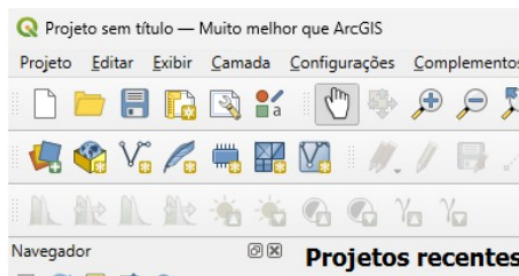
iface – QgisInterface

The `QgisInterface` class is initialized automatically when QGIS starts and is represented by the variable `iface` (short for "interface"). `iface` serves as your gateway to the QGIS application, representing the main window, menus, toolbars, and other components of the program.

In the previous module, we used `iface` on two occasions: once to access a tool from the raster toolbar, and again to load a raster generated via interpolation. Let's now explore other functions available within the `iface` class.

Changing the program name title

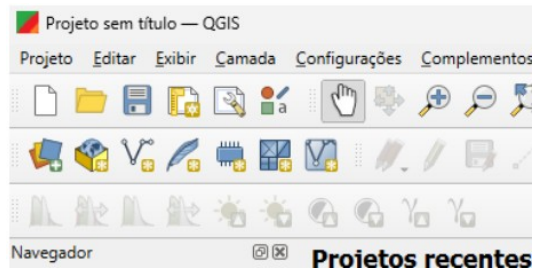
```
titulo = iface.mainWindow().windowTitle()
novo_titulo = titulo.replace('QGIS', 'Muito melhor que ArcGIS')
iface.mainWindow().setWindowTitle(novo_titulo)
```



Changing the program icon

We changed the icon using the code below. `os.path.expanduser('~')` takes you to the user's root directory on your system.

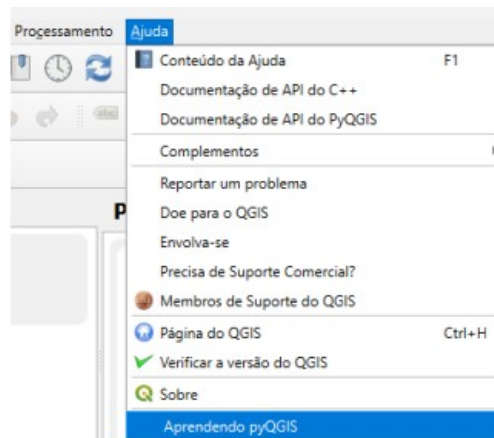
```
import os
icone = 'icon.png'# use um arquivo png 64x64 pixels
dire = os.path.join(os.path.expanduser('~'), 'desktop/dash/')
path = os.path.join(dire, icone)
icone2 = QIcon(path)
iface.mainWindow().setWindowIcon(icone2)
```



Adding a new help menu item

With the following code, we add a new item to the Help menu that directs to our "Learning pyQGIS" page.

```
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)
```

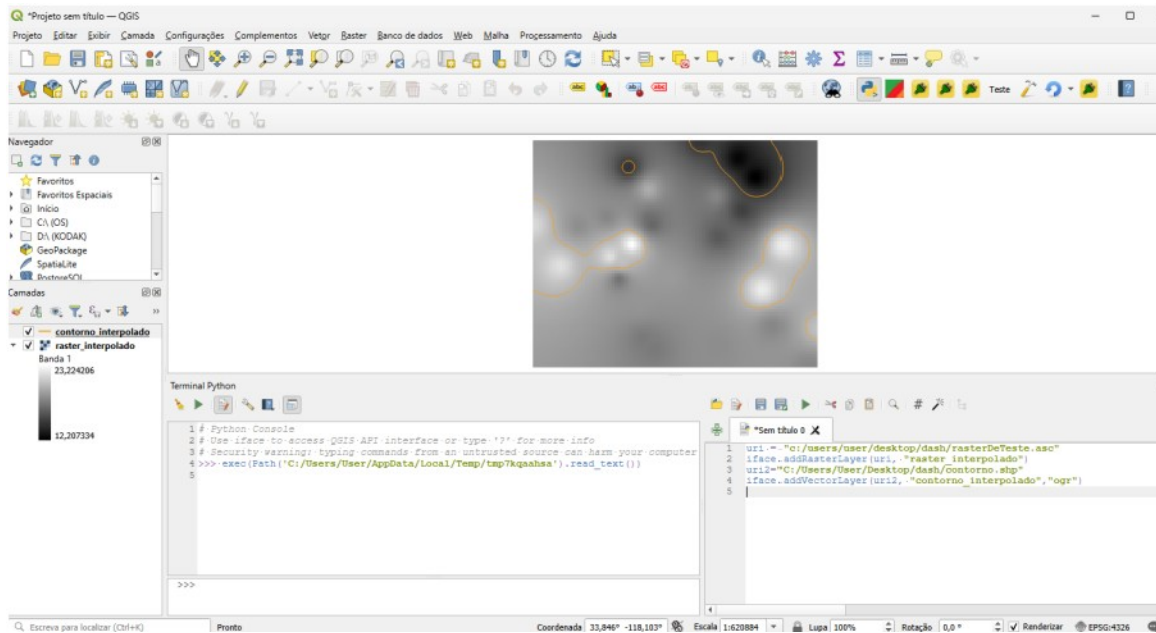


Clicking on the item will open the page in the browser.

Adding a new layer

As we did in the previous module, we can add new layers using `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")
```



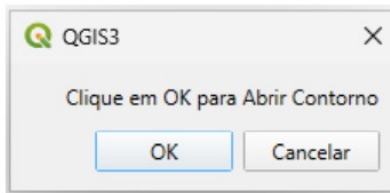
PyQt

Qt is an open-source toolkit or platform of widgets (controls such as buttons, text inputs, labels, etc.) used to create graphical user interfaces (GUIs) and cross-platform applications. QGIS is built upon this platform. PyQt is the Python interface for Qt; it provides classes and functions for interacting with Qt widgets.

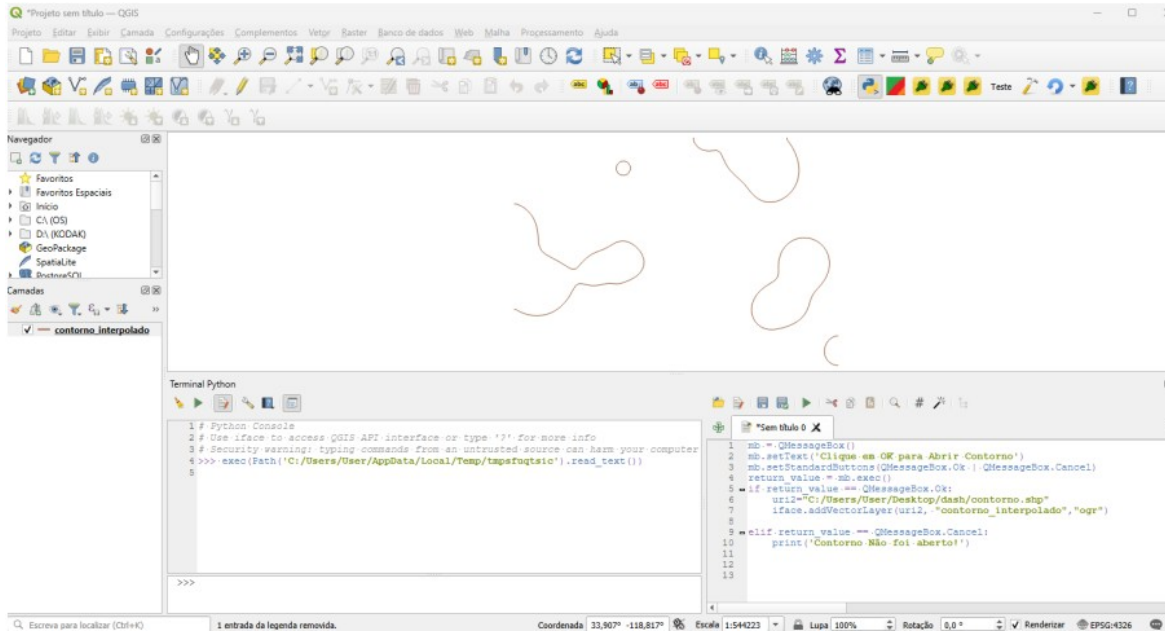
QMessageBox

An example of how to use the Qt class directly within 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!')
```



Upon clicking OK, we will have:



Now, here is a more elaborate example to illustrate how the interaction between QGIS and Qt can be implemented—but don't worry, we have plugins to make everything easier. This is just a demonstration. For these plugins, we will use Qt Designer to create graphical user interfaces visually.

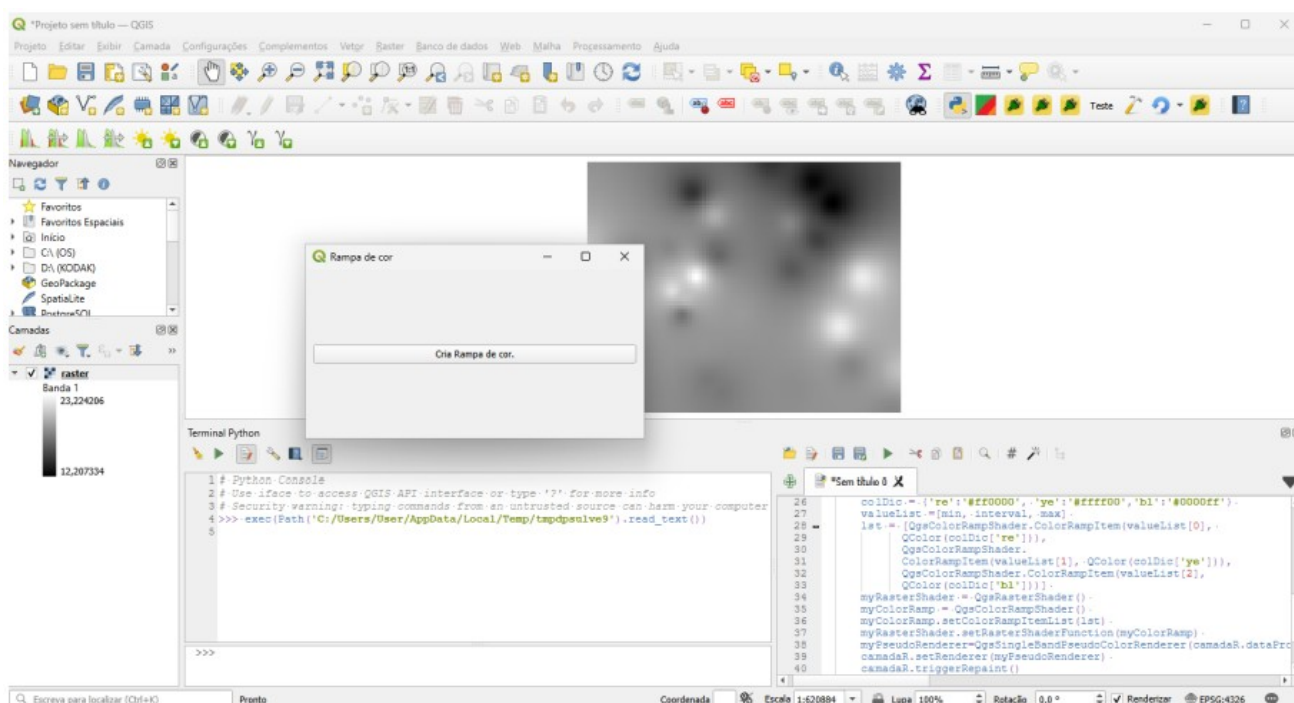
```
class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Rampa de cor")
        self.setGeometry(100, 100, 400, 200)
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        dialog_button = QPushButton("Cria Rampa 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
```

```

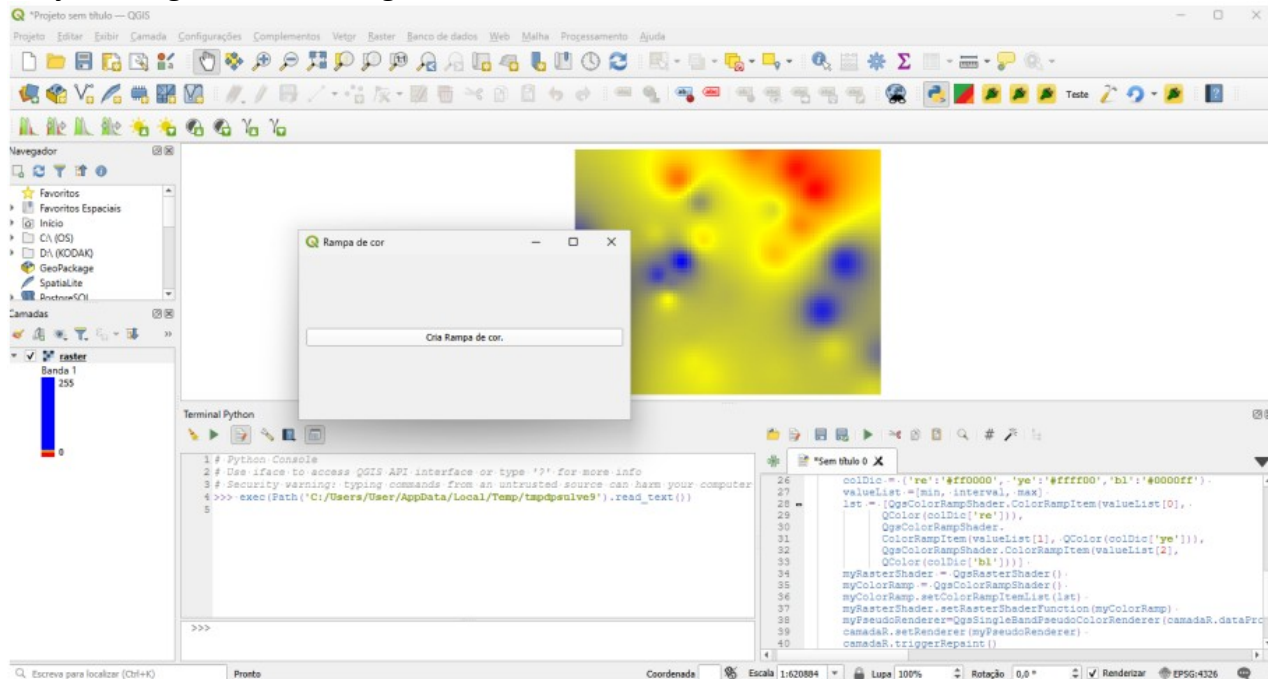
max = stats.maximumValue
range = max - min
add = range//2
interval = min + add
colDic = {'re':'#ff0000', 'ye':'#ffff00', 'bl':'#0000ff'}
valueList = [min, interval, max]
lst = [QgsColorRampShader.ColorRampItem(valueList[0],
QColor(colDic['re'])),
QgsColorRampShader.
ColorRampItem(valueList[1], QColor(colDic['ye'])),
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()

```

Upon execution, we will have:



And by clicking the button, we get:



Running a PyQGIS script outside of QGIS

We can execute QGIS processes without launching the graphical user interface, using only scripts. To do this, we must create a script with the following minimum structure.

```
from qgis.core import *
# Specify the location of the QGIS program on your computer
QgsApplication.setPrefixPath("/usr", True)
# Create a reference to QgsApplication. Use False as the second
# argument to avoid activating the graphical user interface.
qgs = QgsApplication([], False)
# Initialize QGIS
qgs.initQgis()
#####
# Write your processing code here #####
#####
# Finalize the script using:
qgs.exitQgis()
```

We will demonstrate how this works by creating a script that generates a raster grid from a CSV text file containing a set of points—the same procedure used in the final example of the previous module. Name the file `criaRaster.py`. Replace `users/user/desktop/dash` with the appropriate path for your own system.

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

```

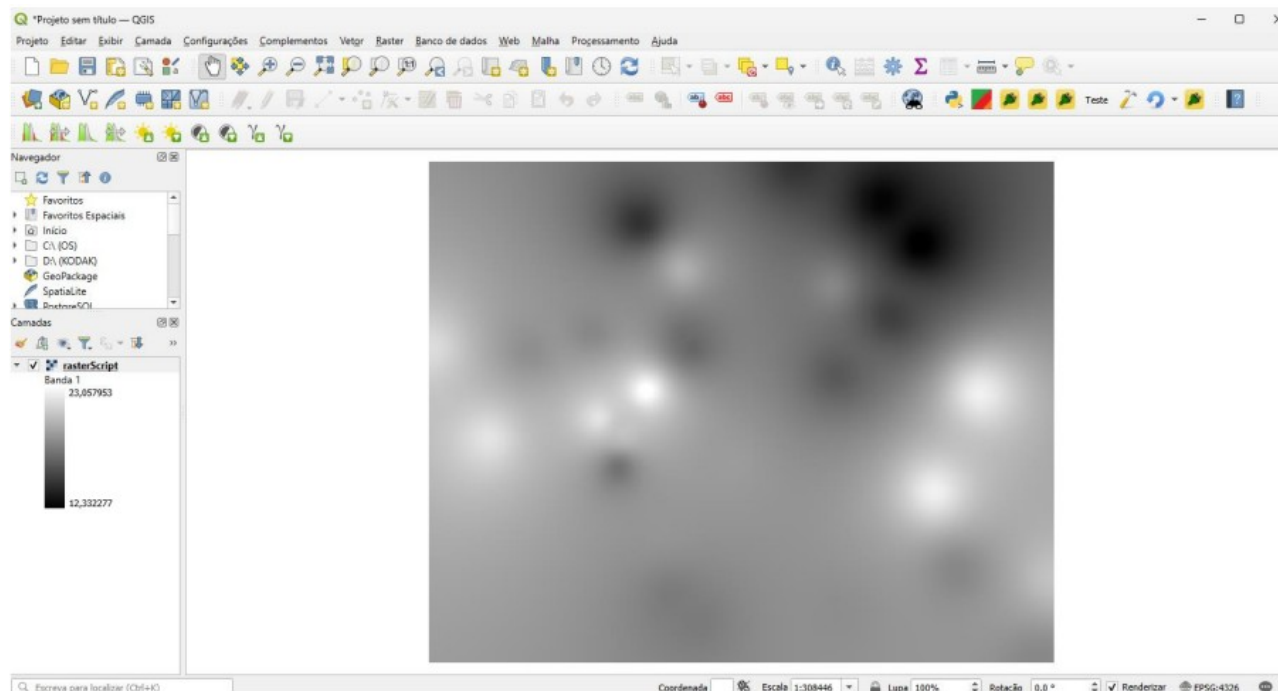
qgs.initQgis()
uri="file:///users/user/desktop/dash/tfinal.csv?type=csv&xField=LONG
ITUDE&yField=LATITUDE&crs=epsg:4326"
camada = QgsVectorLayer(uri, 'Converte', "delimitedtext")
if not camada.isValid():
    print("A camada não carregou apropriadamente!")
else:
    c_data = QgsInterpolator.LayerData()
    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()

```

Open the OSGeo4W shell, navigate to the folder containing the `criaRaster.py` script, and execute it using:

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

When we open the raster file `rasterScript.asc`—created by the script above—in QGIS, we will see the following:



pyQGS cheat sheet

Below is a cheat sheet of the main classes for quick reference. Libraries to be loaded, for use in plugins or in scripts outside of QGIS, are shown in blue.

QgsInterface() iface

Appearance

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

You can load the stylesheet from a file with

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

Changing the program's icon and title

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

Toolbar

Removes and adds a toolbar

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

Remove actions from the toolbar

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

Menus

Remove and add menu

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

Canvas (map canvas)

Accessing Canvas

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

Changing the canvas color

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

Changing the map update interval

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

Layers

Adding a vector layer of the shapefile type

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

Adding a raster layer of the TIFF type

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

Getting the active layer

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

Listing all layers

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

Getting the layer names

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

Getting a layer by its name and making it active

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

Adding a new element

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

Obtaining the elements

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

Getting selected elements

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

Getting the ID of the selected elements

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

Creating a layer in memory with the selected elements

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

Obtaining the layer geometry

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

Translating a geometry

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

Specifying the Coordinate Reference System (CRS) for a layer

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

Checking the CRS of a layer

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

A function to hide an attribute column

```
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
```

Creating a layer in memory from 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])
```

Reading the layers of a vector GeoPackage

```
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(subVamadav)
```

Adding a Tile Layer (XYZ Layer)

```
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')
```

Adding a PostGIS database layer

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

Removing all layers from the project

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

Removing everything from the project

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