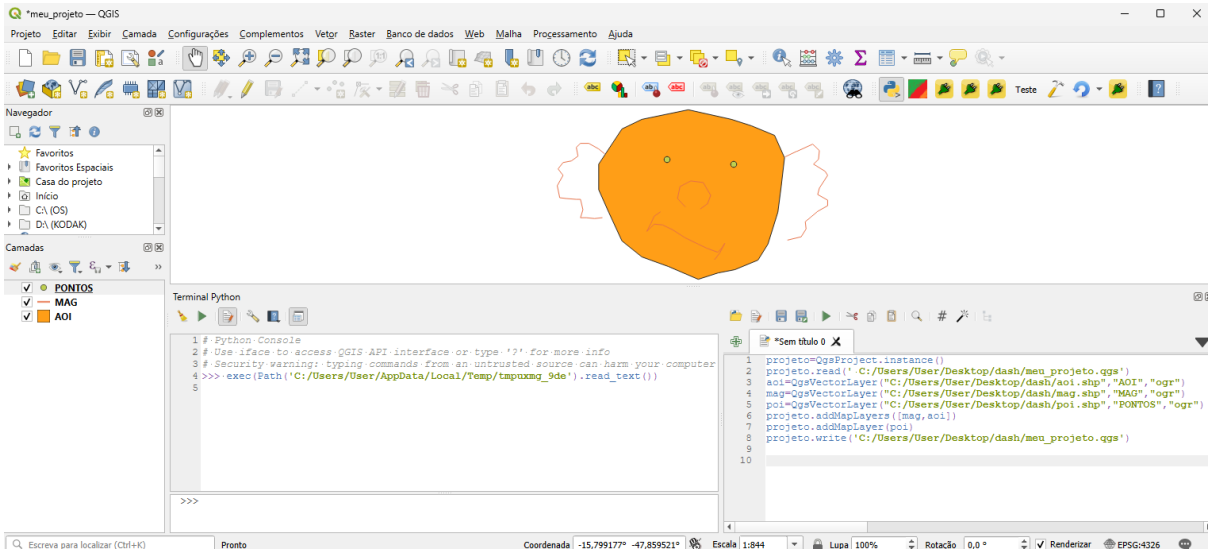
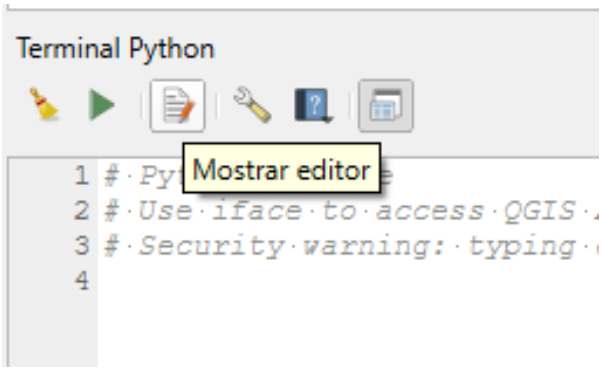



# Usando pyQGIS

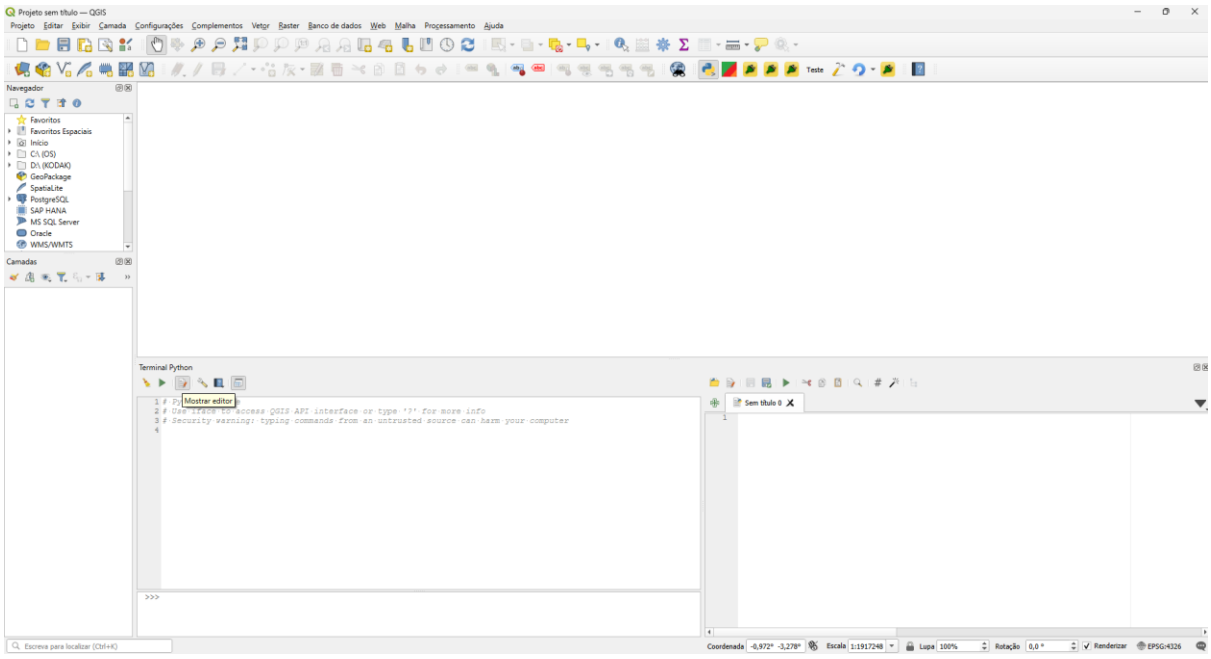
## Introdução às Primeiras Classes

## Preparando o terreno

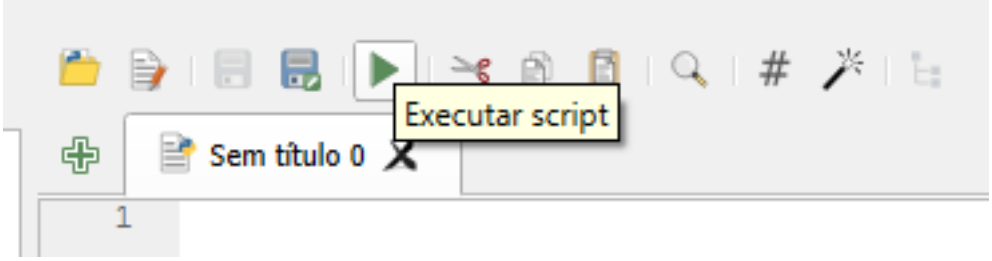
Vamos abrir o QGIS e deixar tudo preparado para podermos operar o ambiente usando o terminal python (CTRL+ALT+P) e clique no botão Mostrar editor:




Teremos a tela desta forma.



Estamos prontos para trabalhar. No editor na parte inferior direita escreveremos o código e para executar 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
```

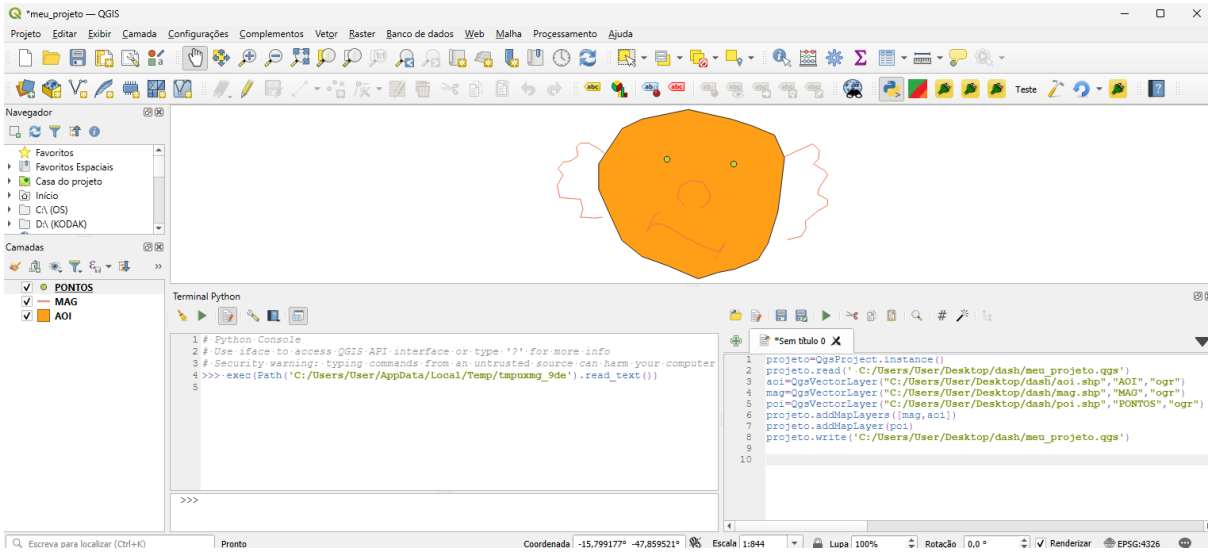
Na medida que formos vendo as outras classes, vamos ver outros métodos associados à classe Projeto.

**Classe Vetor (*QsgVectorLayer*) – 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')
```

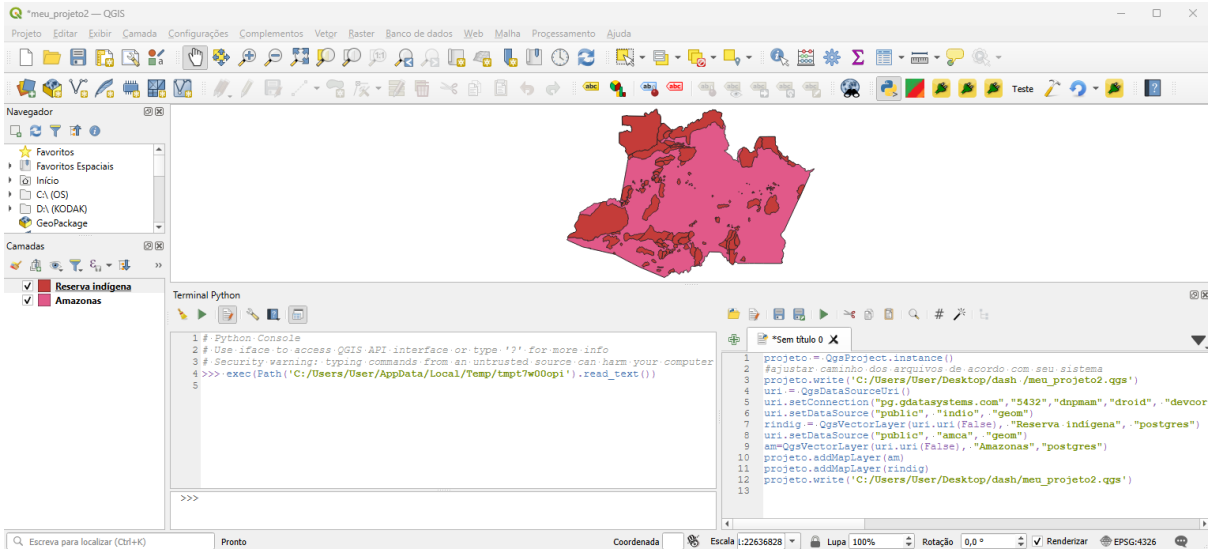
Algo similar ao apresentado abaixo deverá aparecer:




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.

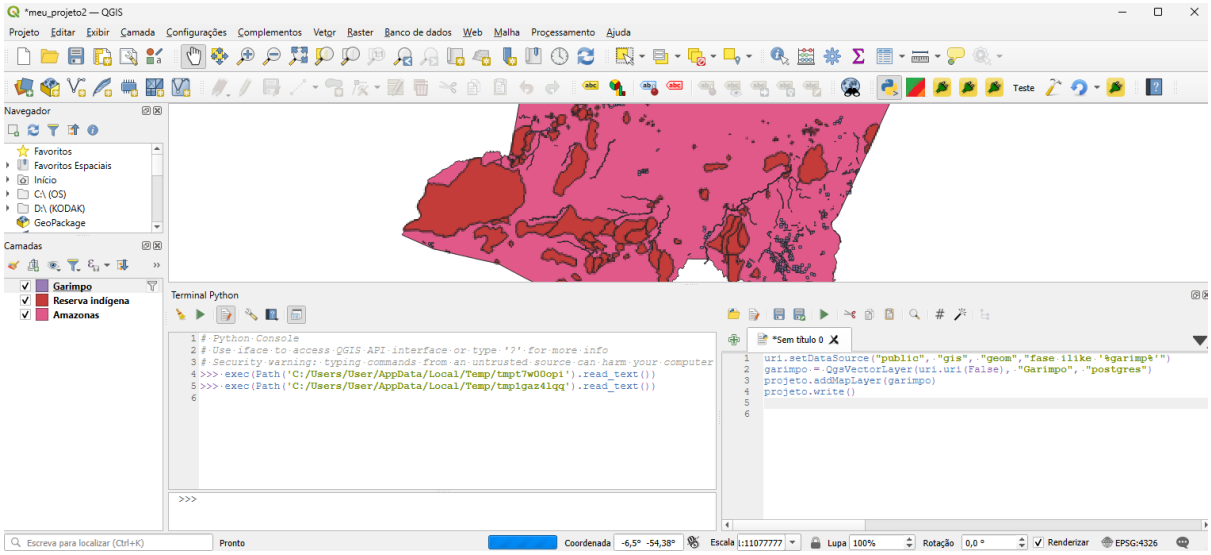
```
projeto = QgsProject.instance()
#ajustar caminho dos arquivos de acordo com seu sistema
projeto.write('C:/Users/User/Desktop/dash /meu_projeto2.qgs')
uri = QgsDataSourceUri()
uri.setConnection("pg.gdatasystems.com","5432","dnpmam","droid", "devcor")
uri.setDataSource("public", "indio", "geom")
rindig = 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 deverá aparecer.



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 (QsgProject) relacionados a classe Camadas Vetoriais (QsgVectorLayer).

**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)>}

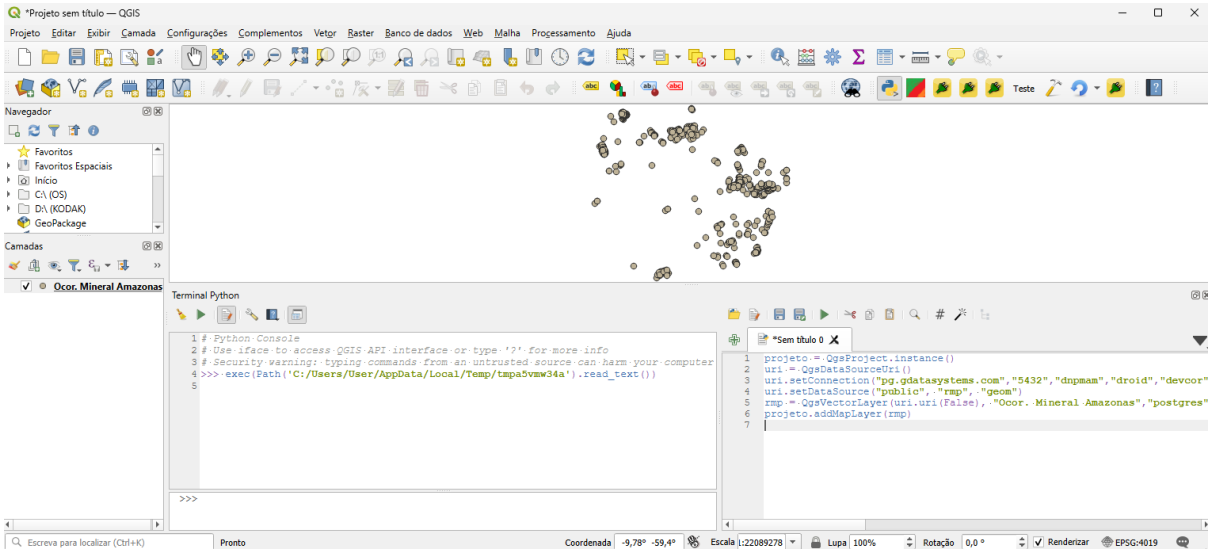
**Interagindo com informações de objetos da classe Vector**

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","droid","devcor")
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())
```
Unknown datum based upon the GRS 1980 ellipsoid

A extensão da Camada

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 -9.53860000030878 -56.7497 2.21390000007294
```

Quantidade de itens

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

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

Obtendo informações dos campos de atributos

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_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

Metadata de camada vetorial

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>
<body>
<h1>Informação do provedor</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Nome</td><td>Ocor. Mineral
Amazonas</td></tr>
<tr><td class="highlight">fonte</td><td>dbname='dnpmam'
host=pg.amazeone.com.br port=5432 user='droid' key='tid'
checkPrimaryKeyUnicity='1' table="public"."rmp" (geom)</td></tr>
<tr><td class="highlight">Armazenamento</td><td>PostgreSQL database
with PostGIS extension</td></tr>
<tr><td class="highlight">Comentário</td><td></td></tr>
<tr><td class="highlight">Codificação</td><td></td></tr>
<tr><td class="highlight">Geometria</td><td>Point (Point)</td></tr>
<tr><td class="highlight">SRC</td><td>EPSG:4019 - Unknown datum based
upon the GRS 1980 ellipsoid - Geográfico</td></tr>
<tr><td class="highlight">Extensão</td><td>-70.0999999999999943,-
9.5386000003087794 : -56.7496999999999971,2.2139000000729401</td></tr>
<tr><td class="highlight">Unidade</td><td>graus</td></tr>
<tr><td class="highlight">Contagem de feições</td><td>624</td></tr>
</table>
<br><br><h1>Identificação</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>Extensão</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">CRS</td><td>EPSG:4019 - Unknown datum based
upon the GRS 1980 ellipsoid - 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>Acesso</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>Campos</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Contagem</td><td>26</td></tr>
</table>
<br><table width="100%" class="tabular-view">
<tr><th>Campo</th><th>Tipo</th><th>Comprimento</th><th>Precisão</th><t
h>Comentário</th></tr>
<tr ><td>codigo_obj</td><td>text</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>text</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>text</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>Contatos</h1>
<hr>
No contact yet. <br><br>

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

Essa informação em HTML acima apareceria  em um navegador assim:

## Informação do provedor

| | |
|---|---|
| Nome | Ocor. Mineral Amazonas |
| fonte | dbname='dnpmam' host=pg.amazeone.com.br port=5432 user='droid' key='tid' checkPrimaryKeyUnicity='1' table="public"."rmp" (geom) |
| Armazenamento | PostgreSQL database with PostGIS extension |
| Comentário | |
| Codificação | |
| Geometria | Point (Point) |
| SRC | EPSG:4019 - Unknown datum based upon the GRS 1980 ellipsoid - Geográfico |
| Extensão | -70.0999999999999943,- 9.5386000003087794 : -56.7496999999999971,2.2139000000729401 |
| Unidade | graus |
| Contagem de feições | 624 |

## Identificação

| | |
|---|---|
| Identifier | |
| Parent Identifier | |
| Title | |
| Type | dataset |
| Language | |
| Abstract | |
| Categories | |
| Keywords | |

## Extensão

| | |
|---|---|
| CRS | EPSG:4019 - Unknown datum based upon the GRS 1980 ellipsoid - Geographic |
| Spatial Extent | |
| Temporal Extent | |

## Acesso

| | |
|---|---|
| Fees | |
| Licenses | |
| Rights | |
| Constraints | |

## Campos

Contagem 26

Comentário

| Campo | Tipo | Comprimento | Precisão |
|---|---|---|---|
| codigo_obj | text | - 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 | text | - 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 | text | - 1 | 0 |
| municipio | text | - 1 | 0 |
| uf | text | - 1 | 0 |

## Contatos

No contact yet.

### Links

No links yet.

### Histórico

No history yet.

Obtendo os elementos de cada item da camada vetorial

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, 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, NULL, NULL, NULL, '2', 'MANICORE', 'AM']

### 3.4 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()
```

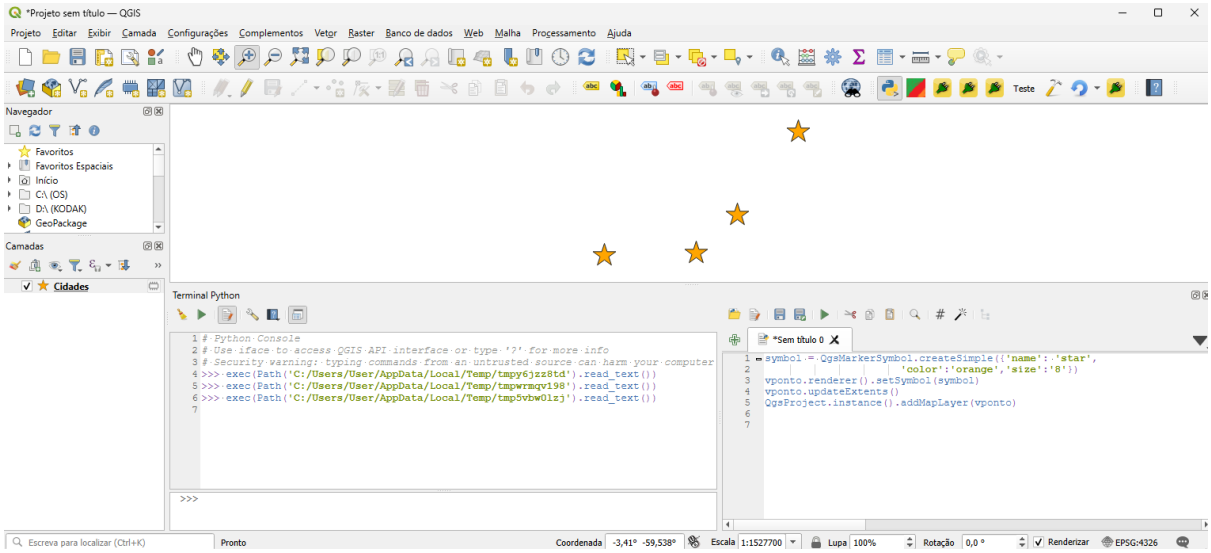
Uma vez criado o objeto ponto e seus campos de atributo vamos adicionar dados nele usando um objeto feature (elemento). Definimos a geometria que será um ponto nesse caso com coordenadas x e y e adicionaremos os atributos deste ponto.

```
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)
```

A aparência do mapa e de sua tabela de atributos será conforme a imagem mostrada abaixo.



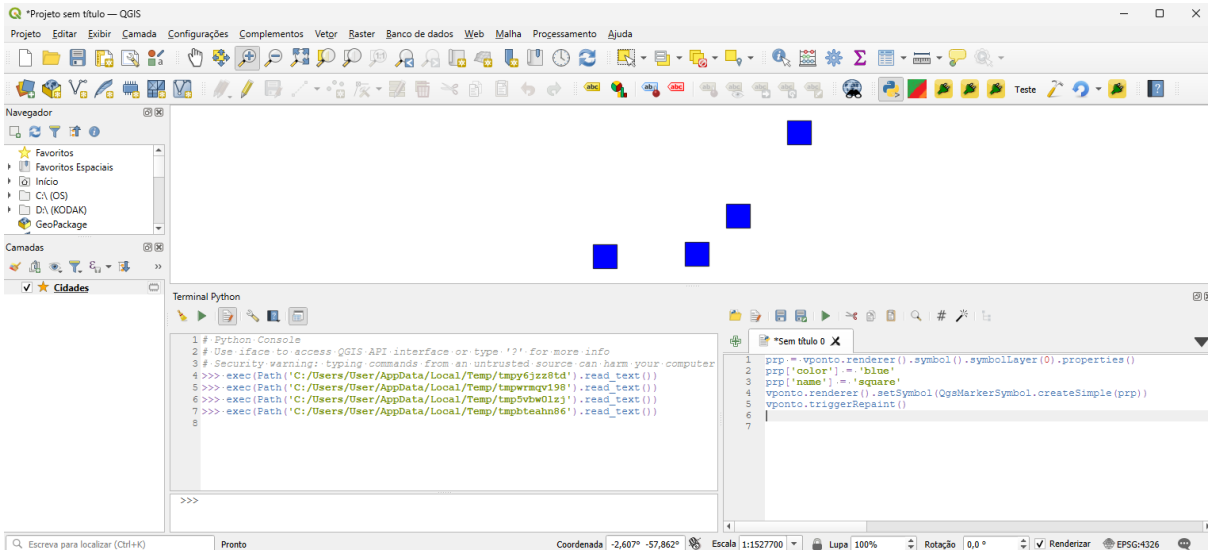

Os parâmetros abaixo podem ser usados com QgsMarkerSymbol.createSimple():

'angle': '0',
'color': '255,165,0,255',
'horizontal_anchor_point': '1',
'joinstyle': 'bevel',
'name': 'star',
'offset': '0,0',
'offset_map_unit_scale': '3x:0,0,0,0,0,0',
'offset_unit': 'MM',
'outline_color': '35,35,35,255',
'outline_style': 'solid',
'outline_width': '0',
'outline_width_map_unit_scale': '3x:0,0,0,0,0,0',
'outline_width_unit': 'MM',
'scale_method': 'diameter',
'size': '8',
'size_map_unit_scale': '3x:0,0,0,0,0,0',
'size_unit': 'MM',
'vertical_anchor_point': '1'

Veja a documentação para cada uma das opções que podem ser usadas com cada parâmetro listado acima. Vamos Modificar a aparência do símbolo que criamos acima.

```
prp = vponto.renderer().symbol().symbolLayer(0).properties()
prp['color'] = 'blue'
prp['name'] = 'square'
vponto.renderer().setSymbol(QgsMarkerSymbol.createSimple(prp))
vponto.triggerRepaint()
```

Veja o resultado abaixo.




Finalizamos mostrando como escrever o nosso objeto ponto em um arquivo.

```
QgsVectorFileWriter.writeAsVectorFormat(vponto,'C:/Users/User/Desktop/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()
```

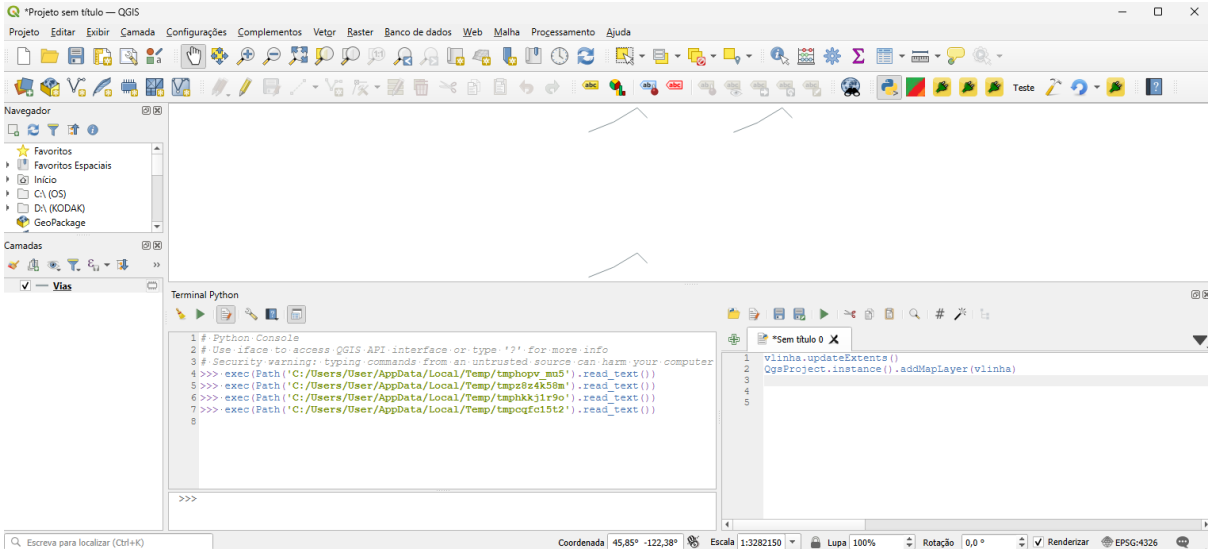
Adicionamos as linhas usando um objeto feature (elemento). Mas antes criamos a lista de pontos que farão parte de cada um dos elementos do tipo linha e inserimos os atributos de cada elemento. Vamos inserir três linhas.

```
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)
```

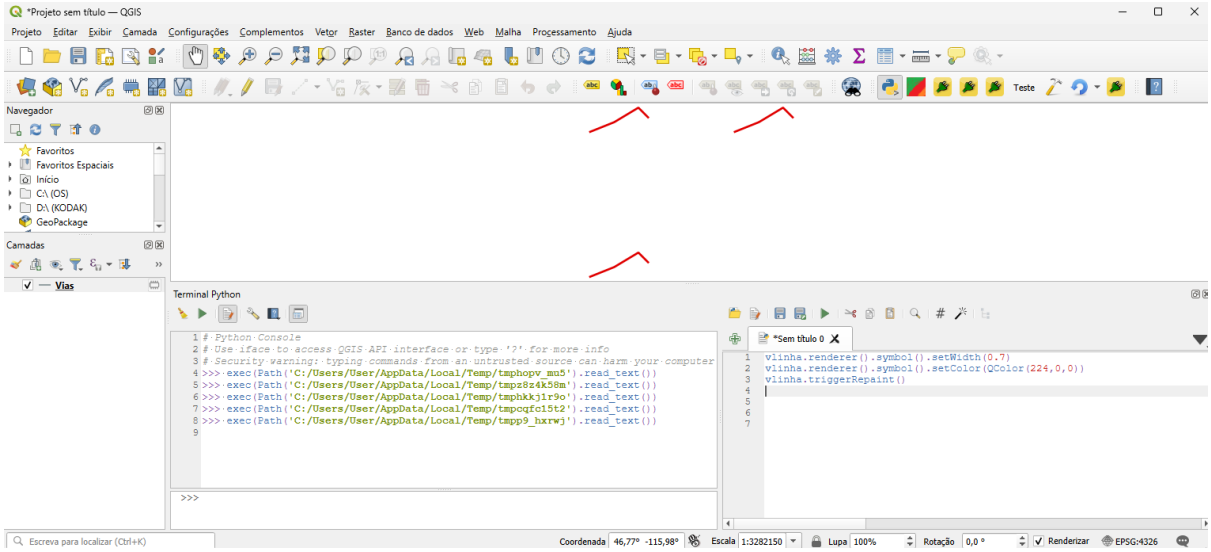
Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
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/Deskto
p/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).

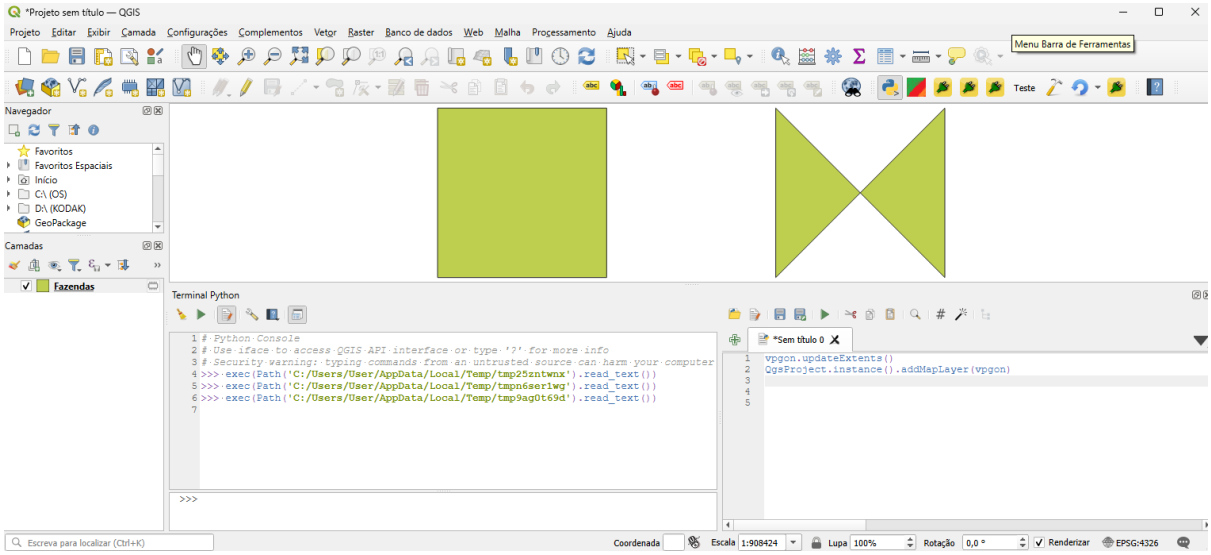
```
vpgon = QgsVectorLayer("Polygon?crs=EPSG:4326", "Fazendas","memory")
dPr = vpgon.dataProvider()
dPr.addAttributes([QgsField("nome",QVariant.String),QgsField("número
",QVariant.Int)])
vpgon.updateFields()
```

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)
```

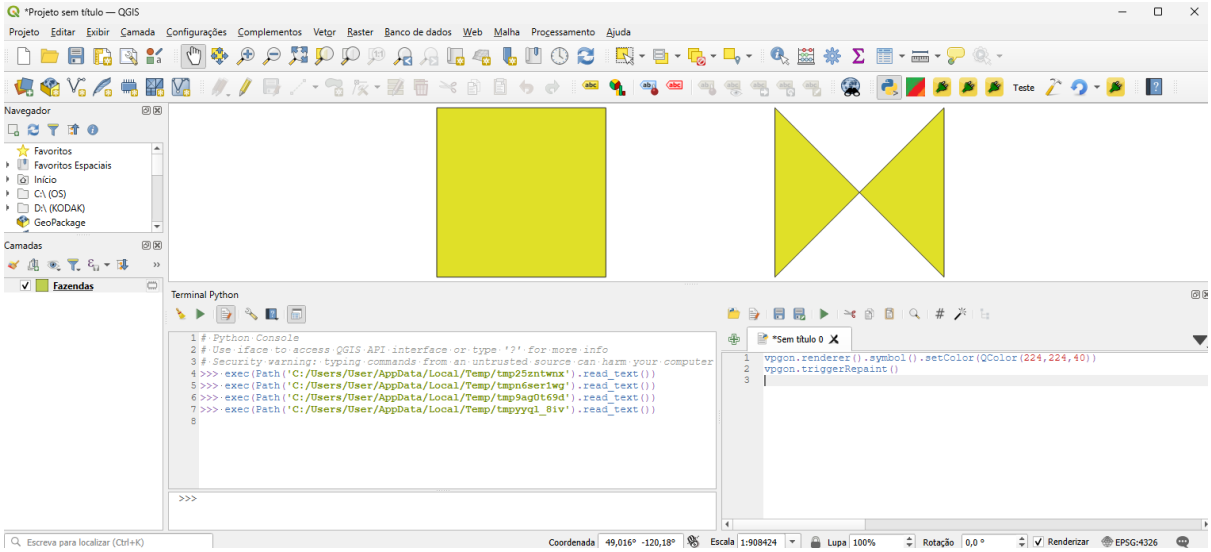
Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
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')
```

No próximo módulo veremos a classe ***QsgRasterLayer***. Até lá!