PyQGIS - Python no QGIS

Disciplina: Programação aplicada à engenharia cartográfica

Maurício C. M. de Paulo - D.Sc.

20 de fevereiro de 2026

Arquitetura do QGIS

QGIS é estruturado em camadas (layers de software):

- **Core** → Modelo de dados geoespacial
- **GUI** → Interface gráfica (Qt)
- **Analysis** → Algoritmos e Processing
- **Providers** → OGR, GDAL, PostGIS, etc.

PyQGIS é o binding Python da API C++ do QGIS.

QgsApplication e Inicialização

Dentro do QGIS:

```
# Já inicializado automaticamente
layer = iface.activeLayer()
```

Script standalone:

```
from qgis.core import QgsApplication

QgsApplication.setPrefixPath("/usr", True)
app = QgsApplication([], False)
app.initQgis()

# código aqui

app.exitQgis()
```

Projeto e Singleton

QgsProject

- Representa o projeto aberto (.qgz)
- Gerencia camadas
- Singleton

Acesso global:

```
QgsProject.instance()
```

Permite:

- Adicionar camadas
- Remover camadas
- Acessar CRS do projeto

Providers

Providers são responsáveis pelo acesso aos dados

- OGR → Shapefile, GeoPackage
- GDAL → Raster
- PostGIS → Banco espacial

Ao criar uma camada:

```
QgsVectorLayer(path, name, "ogr")
```

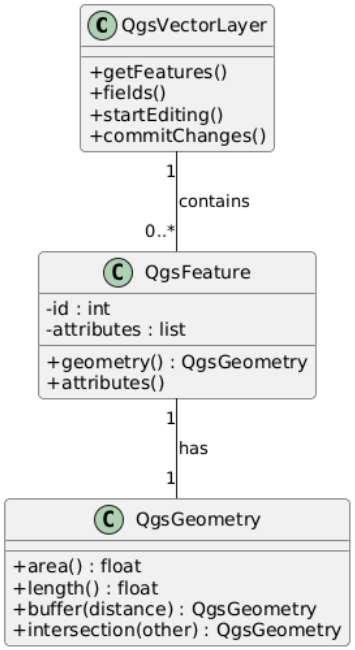
"ogr" é o provider.

Modelo de Dados Vetorial

Principais classes:

- **QgsVectorLayer**
- **QgsFeature**
- **QgsGeometry**
- **QgsField / QgsFields**

Diagrama de classes



Relação conceitual:

$$\text{Layer} \rightarrow \text{Features} \rightarrow \text{Geometry} + \text{Attributes}$$

Iterando Feições

```
layer = iface.activeLayer()

for feat in layer.getFeatures():
  geom = feat.geometry()
  attrs = feat.attributes()

print(geom.area(), attrs)
```

Observação:

- Iteração é lazy (generator)
- Geometria e atributos são acessados separadamente

Geometria (QgsGeometry)

Operações comuns:

- `area()`
- `length()`
- `buffer()`
- `intersection()`
- `difference()`

Conversões:

- WKT
- WKB
- GeoJSON

Edição de Camadas

```
layer.startEditing()

for feat in layer.getFeatures():
  feat["area"] = feat.geometry().area()
  layer.updateFeature(feat)

layer.commitChanges()
```

Regra importante:

- Sempre iniciar e finalizar modo de edição

CRS e Transformações

Classes importantes:

- **QgsCoordinateReferenceSystem**
- **QgsCoordinateTransform**

Transformação típica:

- CRS origem
- CRS destino
- Contexto do projeto

Essencial para operações métricas corretas.

Automação e Scripts Standalone

Objetivo:
Executar rotinas de geoprocessamento fora da interface gráfica do QGIS.

Casos típicos:

- Processamento em lote
- Integração com pipelines ETL (extract, transform, load)
- Execução em servidor
- Agendamento (cron / task scheduler)

Comparação: Standalone vs Plugin

	Plugin	Standalone
Interface gráfica	Sim	Não
Execução agendada	Limitado	Sim
Escalabilidade	Média	Alta
Uso em servidor	Difícil	Ideal

Arquitetura: GUI vs Standalone

Dentro do QGIS:

- iface disponível
- Projeto já carregado
- Ambiente inicializado

Standalone:

- Inicializar QApplication
- Inicializar QgsApplication
- Configurar prefixPath

Estrutura Mínima de Script Standalone

```
from qgis.core import *
from qgis.PyQt.QtWidgets import QApplication
import sys

QgsApplication.setPrefixPath(
"/usr", True
)

app = QApplication(sys.argv)
qgs = QgsApplication([], False)
qgs.initQgis()

# --- código aqui ---

qgs.exitQgis()
```

Componentes Críticos

QgsApplication:

- Inicializa provedores
- Carrega drivers GDAL/OGR
- Gerencia ambiente QGIS

setPrefixPath():

- Define onde o QGIS está instalado
- Essencial para localizar recursos internos

Carregando Camada Vetorial

```
layer = QgsVectorLayer(
"/dados/limites.shp",
"limites",
"ogr"
)

if not layer.isValid():
raise Exception("Camada inválida")
```

Importante: Sem projeto QGIS carregado, tudo deve ser instanciado manualmente.

Executando Processing Standalone

```
import processing
from processing.core.Processing import Processing

Processing.initialize()

result = processing.run("native:buffer", {
  "INPUT": layer,
  "DISTANCE": 100,
  "OUTPUT": "/dados/buffer.shp"
})
```

Observação: Processing precisa ser explicitamente inicializado.

Automação em Lote

Padrão típico:

- Listar arquivos com glob/os
- Iterar camadas
- Executar algoritmo
- Salvar saída

Aplicações:

- Processamento municipal
- Geração de buffers por estado
- Reprojeção em massa

Integração com CLI

Script pode receber argumentos:

- argparse
- variáveis de ambiente
- arquivos de configuração (YAML/JSON)

Permite:

- Execução parametrizada
- Integração com Airflow
- Pipelines CI/CD

Boas Práticas

- Encapsular lógica em funções
- Evitar código global
- Validar camadas antes de processar
- Usar logs ao invés de print
- Garantir qgs.exitQgis()

Arquitetura Standalone — PyQGIS