API QGIS

Utile

PyGIS Cookbook & Official cheat sheet
Documentation
Console : `Extensions > Console Python` puis lancer le script :

```
exec(open('/home/[...]/script.py'.encode('utf-8')).read())
exec(Path('C:/Users/LCMercier/[...]/script.py').read_text())
```

INTERFACE GRAPHIQUE PRINCIPALE `IFACE`

Récupérer la couche active :

```
couche_active = iface.activeLayer()
```

Modifier affichage :

```
iface.mapCanvas().setExtent(emprise) #définir emprise carte
iface.mapCanvas().refresh() #rafraichir affichage
```

Récupérer informations sur la carte :

```
set = iface.mapCanvas().mapSettings().extent()
set.extent()
```

CLASSE `PROJECT`

Récupérer les layers du projet :

```
toutes_couches = QgsProject.instance().mapLayers() #dictionnaire, layer = valeur
```

Supprimer tous les layers du projet :

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

Ajouter un/des layers au projet :

```
QgsProject.instance().addMapLayer(layer)
QgsProject.instance().addMapLayers([layer1,layer2, layer3])
```

OBJETS `LAYER` (COUCHES)

Objets couche : `QgsVectorLayer` (couches vectorielles) ou `QgsMapLayer` (toutes couches)
Caractéristiques du layer :

```
nom = layer.name()
emprise = layer.extent()
type_couche = layer.type() #si vecteur = QgsMapLayer.VectorLayer
nbr_obj = layer.featureCount() #nombres de feature (entités)
liste_attr = layer.fields() #éventuellement, .names()
```

OBJETS `QGSFEATURE` (ENTITÉS)

Récupérer tous les objets d'un layer :

```
for feature in layer.getFeatures():
        geom = feature.geometry()
        attrs = feature.attributes()
        attr_test = feature.attribute("attribut")
```

Faire une requête de features :

```
request = QgsFeatureRequest().setFilterExpression("ATTR=45")
request = QgsFeatureRequest().setFilterRect(QgsRectangle(0,0,1,1))
request = QgsFeatureRequest().setFilterFid(45)
request = QgsFeatureRequest().setDistanceWithin(geom, dist)

for feature in layer.getFeatures(request):
        print(feature)
```

Récupérer un feature par id :

```
layer.getFeature(id)
layer.getGeometry(id)
```

OBJETS `QGSVECTORLAYER` (COUCHES VECTORIELLES)

CHARGEMENT

Charger une coucher vecteur (fichiers géographiques `.shp`):

```
data_source = "D:/.../file.shp"
provider_name = "ogr" #constance pour fichiers géos ; pourrait être "wms" ou autre
couche = QgsVectorLayer(data_source, layer_name, provider_name)
#Puis ajouter au projet :
QgsProject.instance().addMapLayer(couche)
```

Charger une coucher vecteur (fichiers texte `.csv`, `.wkt`) :

```
# Fichier texte : paramètres à préciser (ou regarder manuellement dans propriété)
data_source = "file:///D:/.../file.csv?delimiter=;&xField=x&yField=y&crs=EPSG:XXX"
provider_name = "delimitedtext"
couche = QgsVectorLayer(data_source, layer_name, provider_name)

#Puis ajouter au projet :
QgsProject.instance().addMapLayer(couche)
```

EDITION

Créer une couche vecteur :

```
layerLigne = QgsVectorLayer("LineString?crs=epsg:4326", "nom", "memory")
layerPoint = QgsVectorLayer("Point?crs=epsg:4326", "nom", "memory")
```

Modifier les attributs d'une coucher vecteur :

```
# Préparation :
dp = layer.dataProvider()
layer.startEditing()

# Ajouter des attributs :
dp.addAttributes([
        QgsField("str", QVariant.String),
        QgsField("int", Qvariant.Int),
        QgsField("double", QVariant.Double)
])
layer.updateFields()

# Ou supprimer des attributs :
attrs = [2, 4] #numéros des attributs à supprimer, à récupérer dans table attributaire
dp.deleteAttributes(attrs)
layer.updateFields()

# Modifier la valeur des attributs :
for f in layer.getFeatures():
        id = f.id()
        attr = 3 #numéro de l'attribut à modifier, à récupérer dans table attributaire
        val = "valeur à donner à l'attribut"
        dp.changeAttributeValue({
                id:{attr:val}
                })

# Alternativement, fournir tout le tableau attributaire :
for f in layer.getFeatures():
        f.setAttributes(["attr1", 54, 89.03])
```

```
# Enregistrer
layer.commitChanges
```

Ajouter/supprimer des features à un layer :

```
f_to_add = QgsFeature #()?
# Éventuellement, joindre géométrie et/attributs :
f_to_add.setGeometry(geom)
f_to_add.setAttributes(attr)

# Ajouter au fournisseur de couche
dp.addFeature(f_to_add)
dp.addFeatures([f1, f2, f3])

# Ou supprimer :
dp.deleteFeature(f_to_delete)
```

AUTRE

Sélectionner des objets :

```
layer.select(id_feature)
layer.select(liste_id)
for f in layer.selectedFeatures(): print("Cette entitée est sélectionnée", f)
layer.removeSelection()
```

OBJETS `QGSGEOMETRY` (GÉOMÉTRIES)

Types de géométrie :

```
type(geom)
QgsPoint()
QgsLineSegment2D()
QgsLineString()
# etc...
```

Créer des géométries :

```
p1 = QgsPointXY(10,10)
p2 = QgsPointXY(20,30)
ligne = QgsGeometry.fromPolylineXY([p1,p2])
QgsGeometry().fromRect(QgsRectangle(x1, y1, x2, y2))
```

Retourner une géométrie modifiée :

```
geom.asPoint()
geom.asMultiPoint()
QgsRectangle.buffered(distance)
```

Informations & traitements géométriques :

```
longueur_segment = ligne.length()
distancec = geom1.distance(geom2)
#buffer,intersects, centroid, contain, etc.
```

CLASSE `QGSLAYERTREEGROUP` (ARBORESCENCE DES COUCHES)

Insérer un layer à une position donnée :

```
QgsProject.instance().layerTreeRoot().insertLayer(pos, layer)
pos = len(QgsProject.instance().mapLayers())-1 #avant-dernier (en partant du dessus)
```

Trouver le node correspondant à une couche :

```
layer_node = QgsProject.instance().layerTreeRoot().findLayer(layer.id())
```

Modifier des détails de l'arborescence des couches :

```
# déroulé ou non
layer_node.setExpanded(False)
```

```
# visible ou non (coché/décoché)
layer_node.setItemVisibilityChecked(False)
```

FRAMEWORK `PROCESSING` (GÉO-TRAITEMENTS)

Lister les géo-traitements existants :

```
for alg in QgsApplication.processRegistry().algorithms():
        print(alg.id(), alg.displayName())
```

Obtenir des détails sur un traitement :

```
alg_id = "native:intersection"
processing.algorithmHelp(alg_id)
```

Utiliser un traitement :

```
# Définition des paramètres
param = {
        'INPUT': layer1,
        'OUTPUT': "memory: nom_layer_output"
}

# Lancer le traitement et stocker le résultat
resultat = processing.run(alg_id, param)

# Afficher le résultat
layer_output = resultat["OUTPUT"]
QgsProject.instance().addMapLayer(layer_output)
```

AUTRE

SYMBOLOGIE

Charger et appliquer un style :

```
layer.loadNamedStyle("D:/.../style.qml")
```

FILTRE

Filtrer une couche :

```
layer.setSubsetString("subset_string")
```

REQUÊTES ATTRIBUTAIRES

```
expr_requete = QgsExpression('"id" = \'id_a_match\'') #attention aux quotes simples/doubles qui doivent être respectées (sinon échec silencieux)
resultats = layer.geatFeatures(QgsFeatureRequest(expr_requete))
```

OPTIMISATION DE TRAITEMENTS SPATIAUX AVEC `GEOMETRYENGINE`

```
# Préparation :
f_opti = QgsGeometry.createGeometryEngine(f.geometry().constGet())
f_opti.prepareGeometry()

# Puis utilisation :
if f_opti.intersects(autre.geometry().constGet()):
        print("Intersection")
```

OUVRIR COUCHE RASTER

```
uri = "crs=EPSG:3857&format&http-header:referer=&type=xyz&url=https://tile.openstreetmap.org/%7Bz%7D/%7Bx%7D/%7By%7D.png&zmax=19&zmin=0"
rlayer = QgsRasterLayer(uri, 'nom', 'wms')
```

AFFICHER DES ALERTES

```
mess1 = "1ère partie dui message, en gras"
mess2 = "2ème partie dui message"

iface.messageBar().pushMessage(mess1, mess2, level=Qgis.Success, duration=3) #(durée en secondes)
# niveaux d'alerte possible :
Qgis.Sucess
Qgis.Info
Qgis.Error
Qgis.Critical
```

POUR UTILISER UN MODULE EXTÉRIEUR

copier les fichiers du module dans : `C:\Program Files\QGIS 3.26.3\apps\Python39\Lib`
ou utiliser la console OSGeo : `python -m pip install <package>`