Criando Plugins QGIS com pyQGIS

Módulo 1 - Arquitetura do plugin e conceitos básicos

1 - O mínimo necessário

Para iniciarmos mostraremos como criar um plugin mínimo que consiste de dois arquivos: **metadata.txt** e **__init__.py** . Estes arquivos devem ser colocados em uma nova pasta (mínimo) dentro da pasta de plugins. Normalmente, para cada tipo de sistema, a pasta fica em:

Linux
~/.local/share/QGIS/QGIS3/profiles/default/python/plugins/minimo
Windows
C:\Users\USER\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\minimo
macOS
~/Library/Application Support/QGIS/QGIS3/profiles/default/python/plugins/minimo

Onde ~ é o diretório home do usuário no Linux ou no mac e USER é a pasta do usuário no windows.

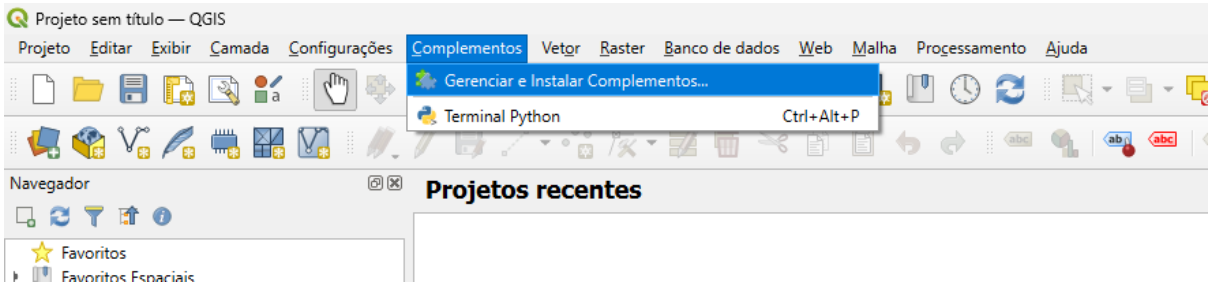
Arquivo **metadata.txt** com os oito campos obrigatórios

```
[general]
name=Minimo
description=Plugin Mínimo
about=Sobre este plugin
version=1.0
qgisMinimumVersion=3.0
author=Você
email=seu@email.com
repository=URL para o repositório do código na web
```

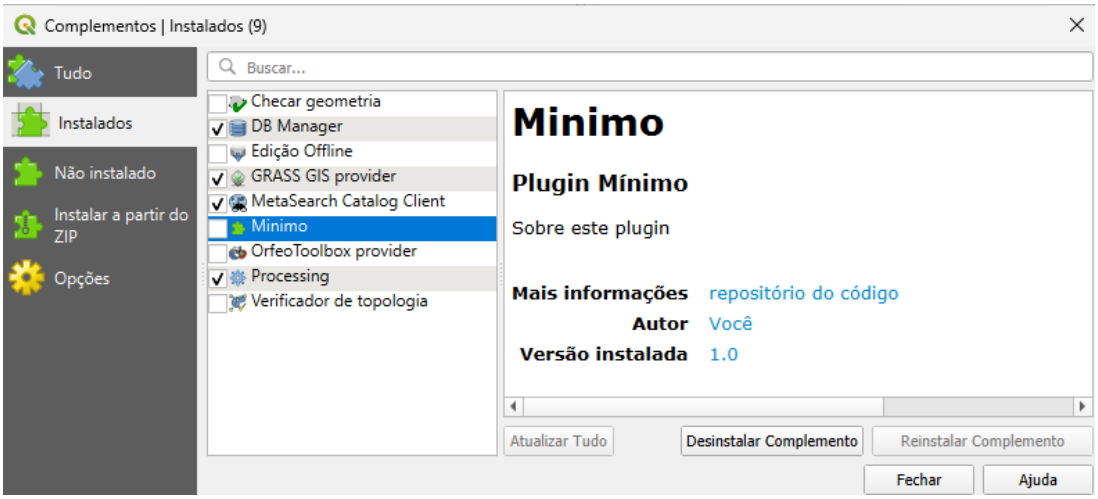
Arquivo **__init__.py**

```
#-----------------------------------------------------------------------
from PyQt5.QtWidgets import QAction, QMessageBox

def classFactory(iface):
    return Minimo(iface)

class Minimo:
    def __init__(self, iface):
        self.iface = iface

    def initGui(self):
        self.action = QAction('Teste', self.iface.mainWindow())
        self.action.triggered.connect(self.run)
        self.iface.addToolBarIcon(self.action)

    def unload(self):
        self.iface.removeToolBarIcon(self.action)
        del self.action

    def run(self):
        QMessageBox.information(None,'Plugin Mínimo','Kreegah
bundolo!')
```

Depois de ter criado os arquivos na nova pasta dentro da pasta plugins do QGIS, inicie o QGIS. Vá no menu **Complementos->Gerenciar e instalar Complementos**.



Selecione **Instalados** e o nosso plugin **Minimo** aparecerá na lista.

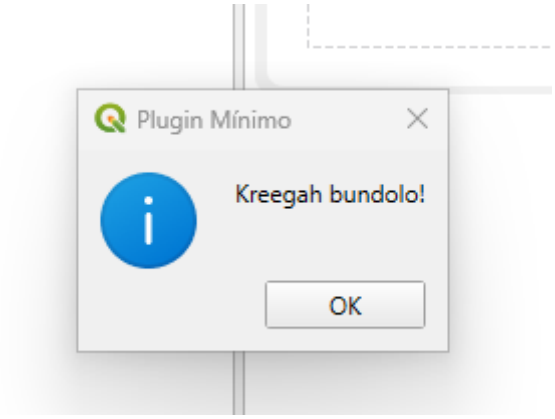


Selecione ele e feche.

O seguinte Icone “Teste” aparecerá na barra de ferramentas:

Clique nele.

Pronto, seu primeiro plugin com pyQGIS foi executado mostrando a seguinte mensagem:



O arquivo metadata.txt contém as informações sobre o plugin tais como quem criou, e-mail de quem criou, repositório web dos arquivos e informações que serão apresentadas quando adicionamos o plugin.

O arquivo __init__.py é requerido pelo sistema import do Python. Ele contém a função classFactory() que é chamada quando carregamos o plugin no QGIS.

Aqui colocamos o código fonte da classe do plugin dentro do arquivo também, mas normalmente a classe do plugin (class Minimo) iria em um arquivo separado. Veremos mais adiante como fazemos isso.

Dentro da classe Mínimo temos 4 funções:

__init__

Onde ganhamos acesso à interface do QGIS.

initGui

Essa função é chamada quando o plugin é carregado. Aqui se faz toda a inicialização das variáveis do plugin, inclusive as da interface gráfica do usuário (GUI) que detalharemos adiante.

unload

Essa função é chamada quando desativamos o plugin

run

Nessa função temos a execução do plugin propriamente dita.

Neste exemplo ainda não temos nenhuma interface gráfica de usuário (GUI). A seguir mostraremos como construir um plugin básico mais completo que esse.

2 - O básico

O plugin básico apresenta mais arquivos que são basicamente os diálogos (GUI) do plugin, os recursos para o diálogo e plugin.

O diálogo consiste em um arquivo XML com a extensão **.ui** que será transformado em um arquivo **.py** para ser acionado quando rodamos o plugin.

O arquivo de recurso é o **resources.qrc** que também é em XML e será transformado em arquivo **resources.py** para ser acionado ao executarmos o plugin. Um exemplo de recurso é um ícone para o plugin que aqui será o arquivo icon.png (download em https://gdatasystems.com/pyqgis/index.php).

Vamos separar a classe principal do plugin do arquivo __init__.py conforme mencionamos anteriormente, assim teremos basicamente que escrever o código do plugin nesse arquivo somente.

Na pasta de plugin crie uma nova pasta chamada **básico** e carregue os seguintes arquivos:

metadata.txt

```
[general]
name=Basico
description=Plugin Basico
about=Sobre este plugin
version=1.0
qgisMinimumVersion=3.0
author=Você
email=seu@email.com
repository=URL para o repositório do código na web
```

__init__.py

```
def classFactory(iface):
    from .basico import Basico
    return Basico(iface)
```

basico_dialog.py

```
import os
from PyQt5 import uic
from PyQt5 import QtWidgets

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'basico.ui'))


class BasicoDialog(QtWidgets.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(BasicoDialog, self).__init__(parent)
        self.setupUi(self)
```

resouces.qrc

```
<RCC>
    <qresource prefix="/plugins/basico" >
        <file>icon.png</file>
    </qresource>
</RCC>
```

basico.ui **(**esse arquivo pode ser feito automaticamente com o QtDesigner que é instalado junto com o QGIS. Use este por praticidade, mostraremos como usar o QtDesigner mais adiante**.)**

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <widget class="QDialog" name="Dialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>201</width>
    <height>115</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Básico</string>
  </property>
  <widget class="QDialogButtonBox" name="buttonBox">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>50</y>
     <width>171</width>
     <height>32</height>
    </rect>
   </property>
   <property name="orientation">
    <enum>Qt::Horizontal</enum>
   </property>
   <property name="standardButtons">
    <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
   </property>
  </widget>
  <widget class="QgsFileWidget" name="mQgsFileWidget">
   <property name="geometry">
    <rect>
     <x>20</x>
     <y>10</y>
     <width>151</width>
     <height>27</height>
    </rect>
   </property>
  </widget>
 </widget>
```

```
 <customwidgets>
  <customwidget>
   <class>QgsFileWidget</class>
   <extends>QWidget</extends>
   <header>qgsfilewidget.h</header>
  </customwidget>
 </customwidgets>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>Dialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>248</x>
     <y>254</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>Dialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>316</x>
     <y>260</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

basico.py

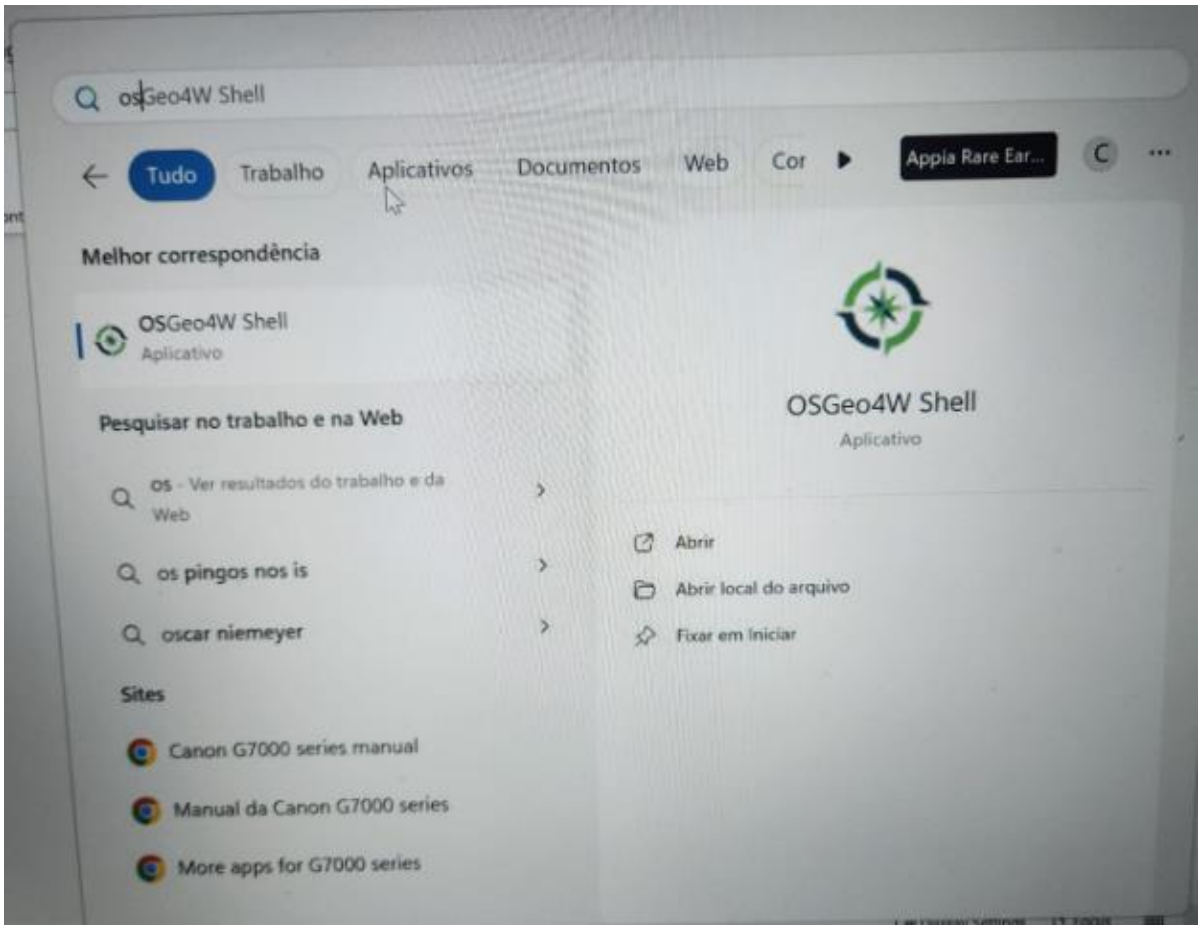
```
from PyQt5.QtCore import QSettings, QTranslator, QCoreApplication
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QAction, QMessageBox
from qgis.gui import QgsFileWidget
import os.path
from .resources import *
from .basico_dialog import BasicoDialog

class Basico:
    def __init__(self, iface):
```

```
        self.iface = iface
        self.actions = []
        self.menu = '&Básico'
        self.first_start = None

    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            self.iface.addToolBarIcon(action)

        if add_to_menu:
            self.iface.addPluginToMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        icon_path = ':/plugins/basico/icon.png'
        self.add_action(
            icon_path,
            text='Básico',
            callback=self.run,
            parent=self.iface.mainWindow())

        self.first_start = True

    def unload(self):
        for action in self.actions:
            self.iface.removePluginMenu(
                '&Básico',
                action)
            self.iface.removeToolBarIcon(action)
```

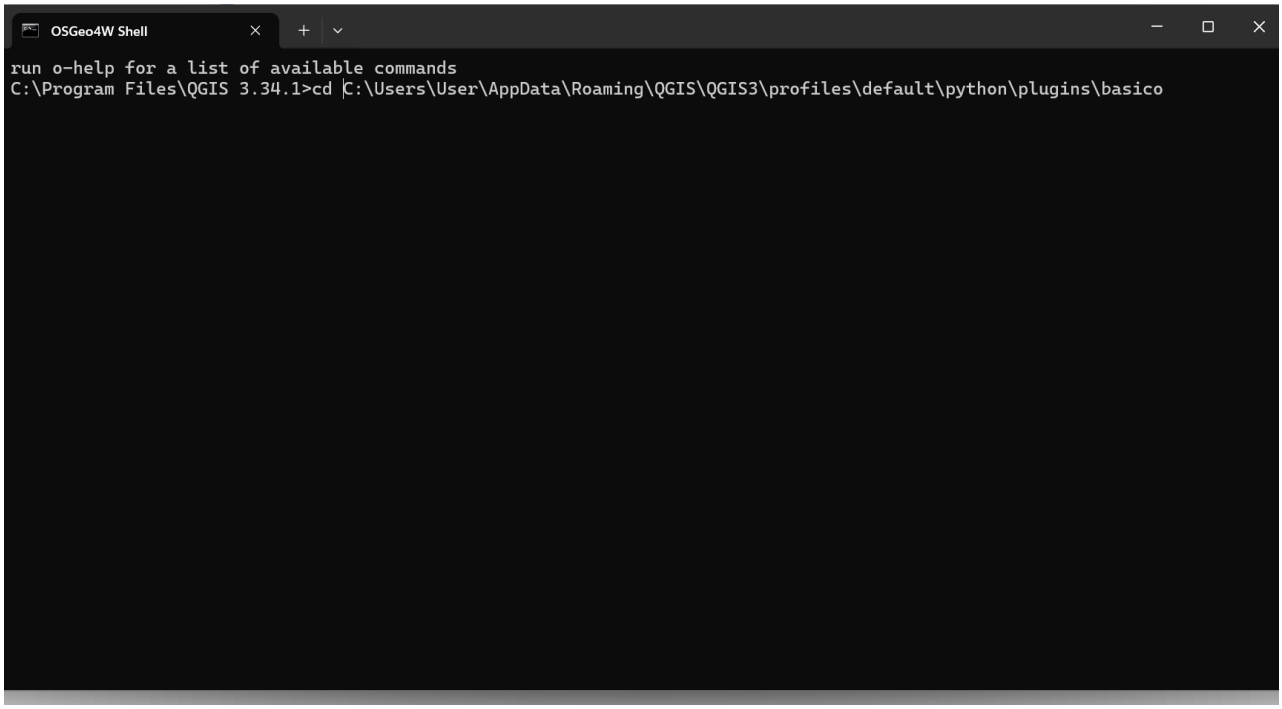
```
def run(self):
    if self.first_start == True:
        self.first_start = False
        self.dlg = BasicoDialog()

    self.dlg.show()
    result = self.dlg.exec_()
    if result:
        arqui=self.dlg.mQgsFileWidget.filePath()
```

Copie na pasta também o arquivo **icon.png** que foi baixado.

Um último passo é gerar o arquivo **resouces.py** a partir do arquivo **resouces.qrc** acima. No Windows isso é feito usando um programa que foi instalado junto com o QGIS chamado **pyrcc5** que deve ser executado usando o OSGeo4w shell (linha de comando).



Navegue até a pasta onde fica o plugin usando cd C:\Users\.....

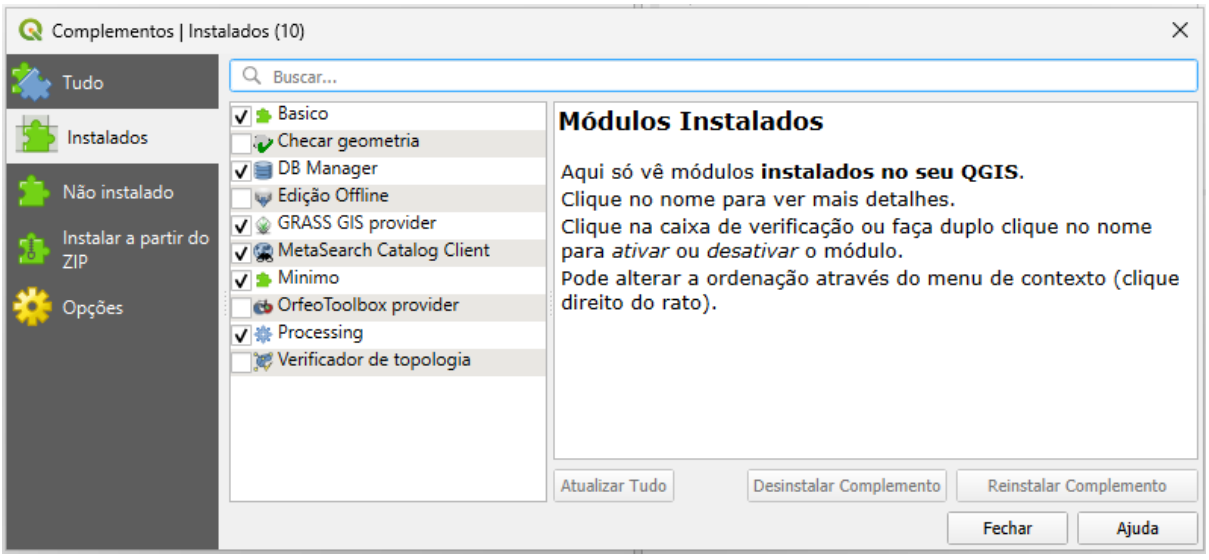


E digite:

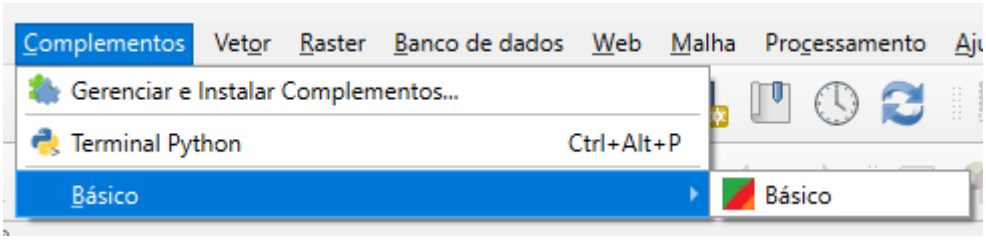
```
pyrcc5 -o resources.py resources.qrc
```

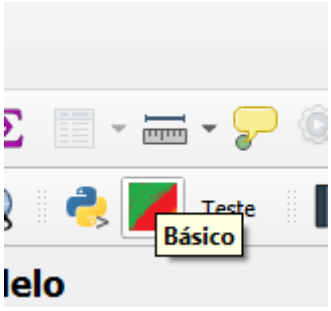
Assim o arquivo **resources.py** é criado na pasta certa. Pode digitar **exit** na linha de comando para fechar a shell.

Inicie o QGIS e instale o plugin **Basico** da mesma forma que fizemos com o plugin **Minimo** acima.

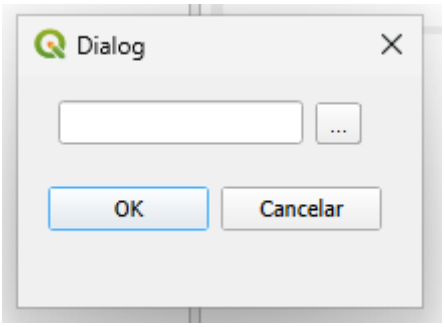


Um item no menu complementos e um ícone na barra de ferramenta serão criados para acessar o plugin.





Este plugin não faz nada no momento, foi só para exemplificar como construir um plugin diretamente.



No próximo módulo vamos criar o primeiro plugin funcional e usaremos o **QtDesigner** e o **PluginBuilder** para automatizar a criação de plugins.