



# Universidad Autónoma del Estado de México

## Facultad de Geografía




## Programación con Python

Orientado a la automatización de procesos en QGIS

Ing. Adonai Emmanuel  
Nicanor Bautista




# Introducción a la automatización Vectorial

- Consiste en explotar la tecnología para realizar tareas con nula o muy poca intervención humana.
  - Una de las grandes ventajas que conlleva el automatizar procesos es la generación de mas productos en menor tiempo, reduce el cometer errores en tareas complejas, existe una reducción de costos y existe la optimización de trabajos.
- 



# Introducción a la automatización Vectorial

- Gracias a la API de QGIS, podemos tener acceso a las distintas librerías y funciones propias del SIG permitiendo explotar estas a su máximo potencial a través de Python.
  - Las funciones que serán aplicadas en este ejercicio son:
    - Lectura de datos vectoriales
    - Cálculo y procesamiento de variables (atributos de los archivos vectoriales)
    - Categorización y clasificación de datos (Mapas temáticos)
- 

# Lectura de Capas Vectoriales

La primer librería a usar es **QgsVectorLayer** la cual permite abrir un archivo vectorial a través de su **ruta de ubicación**.

- **QgsVectorLayer** recibe 3 parámetros
  - 1)La ruta (path) del archivo.
  - 2)El nombre designado al archivo.
  - 3)El parámetro “ogr” para leer archivo
- **Nota:** Se debe realizar la importación de la librería

```
from qgis.core import (  
    QgsVectorLayer  
)
```

# Lectura de Capas Vectoriales


- Una vez que se leyó el archivo, es necesario crear una instancia de éste para mostrarlo en el workspace y tener acceso a todos su atributos, tanto al objeto mismo como a los propios del vectorial.
- **QgsProject.instance().addMapLayer(capa)** permite crear está instancia y facilita tener acceso a la capa vectorial en cualquier momento.
- **QgsProject.instance()** por si mismo deja tener acceso a distintas funciones como:
  - Agregar las capas - **addMapLayer(capa)**
  - Leer capa por nombre – **mapLayersByName("nombre")** - Lista

# Lectura de Capas Vectoriales

- **QgsProject.instance()** por si mismo deja tener acceso a distintas funciones como:
  - Leer todas las capas que se encuentren en el workspace (**habilitadas**) - **QgsProject.instance().mapLayers()** - Es un dict
  - **QgsProject.instance().mapLayers().values()**
- **Es importante recordar que todos son objetos vectoriales.**
  - De las métodos que más se usarán serán:
    - **source()** - path del archivo
    - **name()** - nombre de la capa
    - **fields()** - Retorna los atributos de la capa vectorial como objeto iterable



# Lectura de Capas Vectoriales

- **fields()** - Dentro de fields podemos acceder a las diferentes características de los atributos como:
    - **name()** - Nombre del campo (Nombre del atributo)
    - **typeName()** - Tipo del campo (Integer, Float, String, etc.)
- 

# Procesamiento de Capas Vectoriales

- Dentro de las tantas librerías que ofrece QGIS, existe **processing** la cual permite tener acceso a todos los algoritmos existentes en el SIG.
- Existen dos formas de realizar la importación de esta librería

```
3 import processing
4
5 from qgis import processing
```



# Procesamiento de Capas Vectoriales

- Pero, ¿Cuáles son todos esos algoritmos?

```
12 for alg in QgsApplication.processingRegistry().algorithms():  
13     |     print(alg.id(), "----", alg.displayName())
```

- Para el ejercicio se hará uso de unicamente dos:
  - **qgis:basicstatisticsforfields** – Resumen de estadísticas básicas
  - **qgis:fieldcalculator** - Calculadora de Campos
- La forma de ejecutar estos algoritmos es través de:
  - **var = processing.run('algoritmo', dict)**

# Procesamiento de Capas Vectoriales

- **qgis:basicstatisticsforfields** – Resumen de estadísticas básicas

```
statistics_data = {  
    "INPUT_LAYER": capa,  
    "FIELD_NAME": "nombre_campo",  
}  
  
stats = processing.run('qgis:basicstatisticsforfields', statistics_data)
```

- Retorna un **dict** con todas las estadísticas básicas
  - Suma, mínimo, máximo, promedio, numero de elementos, etc.

# Procesamiento de Capas Vectoriales

- **qgis:fieldcalculator** - Calculadora de Campos

```
field_calc_data = {  
    "INPUT": layer,  
    "FIELD_NAME": "nombre_campo",  
    "FIELD_TYPE": 0, #flotante  
    "FIELD_LENGTH": 10,  
    "FIELD_PRECISION": 4,  
    "NEW_FIELD": True, #el campo se va a crear  
    "FORMULA": "campo_a + campo_b"  
    "OUTPUT": "C:/ejemplo/calle.shp"  
}  
  
result = processing.run('qgis:fieldcalculator', field_calc_data)
```

- Retorna un **dict** con una llave llamada **OUTPUT**

# Procesamiento de Capas Vectoriales

- ¿Cómo saber que llaves y valores debe contener el dict para cada algoritmo?
- **processing.algorithmHelp("algoritmo")**

```
>>> processing.algorithmHelp("qgis:basicstatisticsforfields")  
Estadísticas básicas para campos (qgis:basicstatisticsforfields)
```

Este algoritmo genera estadísticas básicas a partir del análisis de valores en un campo en la tabla de atributos de una capa vectorial. Se admiten campos numéricos, de fecha, hora y de cadena. Las estadísticas devueltas dependerán del tipo de campo. Las estadísticas se generan como un archivo HTML.

# Clasificación y Categorización de Capas Vectoriales

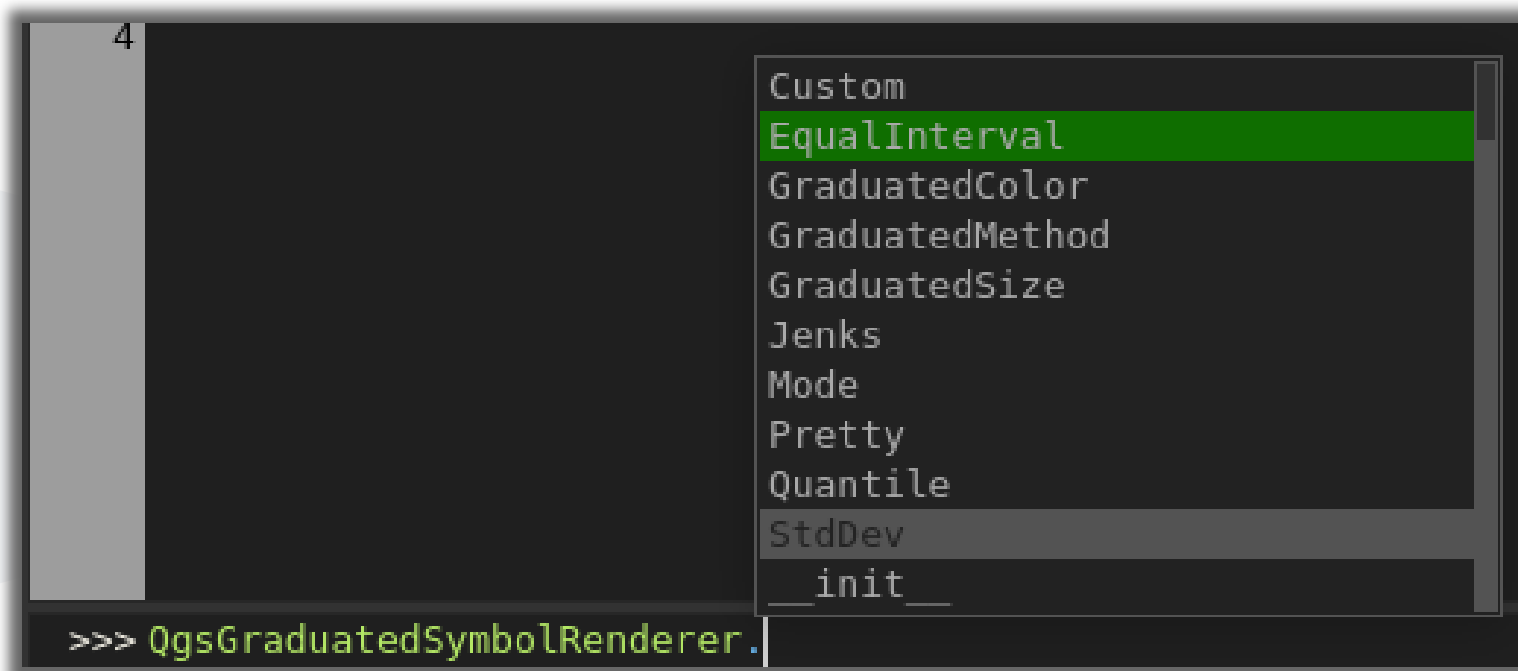
- También podemos crear mapas temáticos de forma rápida en QGIS con Python.
- Para ello se requieren 5 aspectos importantes:

1) El símbolo al que se desea aplicar el temático

- **QgsFillSymbol** – Para polígonos
- **QgsMarkerSymbol** – Para puntos
- **QgsLineSymbol** – Para líneas

# Clasificación y Categorización de Capas Vectoriales

2) El tipo de clasificación [**QgsGraduatedSymbolRenderer.tipo**]



# Clasificación y Categorización de Capas Vectoriales

## 3) Aplicación de Estilo `style = QgsStyle.defaultStyle()`

- Para aplicar estilos como cambiar de tamaño, agregar etiquetas, etc.

## 4) Aplicación e Identificación de rampa de colores `color_ramp = style.colorRampNames()`

```
['Blues', 'BrBG', 'BuGn', 'BuPu', 'Cividis', 'GnBu', 'Greens', 'Greys', 'Inferno', 'Magma', 'Mako', 'OrRd', 'Oranges', 'PRGn', 'PiYG', 'Plasma', 'PuBu', 'PuBuGn', 'PuOr', 'PuRd', 'Purples', 'RdBu', 'RdGy', 'RdPu', 'RdYlBu', 'RdYlGn', 'Reds', 'Rocket', 'Spectral', 'Turbo', 'Viridis', 'YlGn', 'YlGnBu', 'YlOrBr', 'YlOrRd']
```



# Clasificación y Categorización de Capas Vectoriales

4) Aplicación e Identificación de rampa de colores **color\_ramp = style.colorRampNames()**

- Aplicación de la rampa de colores a través del índice del color elegido (Recordando que es una lista) **ramp = style.colorRamp(color\_ramp[34])**

5) Selección del campo/campos/expresión a categorizar

- Índice calculado, una operación de campos, etc.
- 

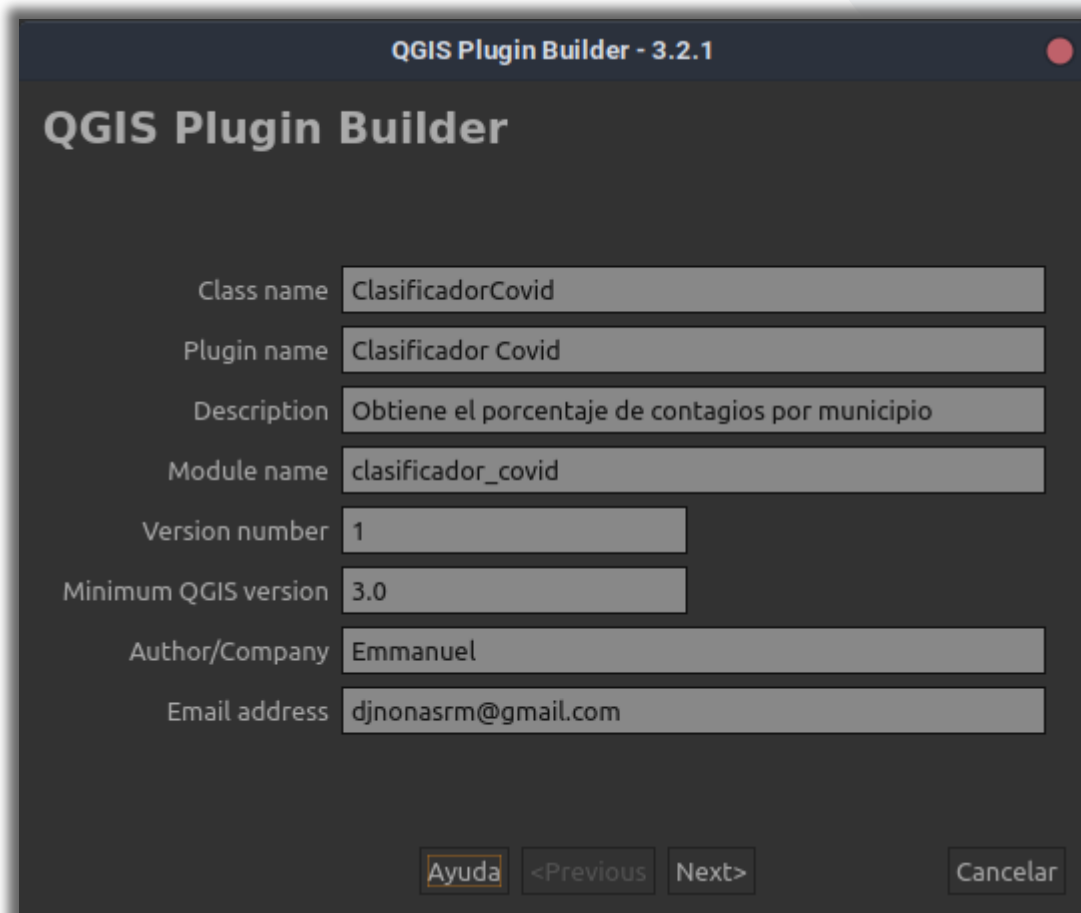


# Clasificación y Categorización de Capas Vectoriales

- Se configura la aplicación del temático con **QgsGraduatedSymbolRenderer.createRenderer()** donde recibe los siguientes parámetros:
  - 1) La capa vectorial (polígono)
  - 2) El campo/expresión que será categorizado
  - 3) El número de clases para categorizar
  - 4) El tipo de clasificación
  - 5) El tipo de símbolo
  - 6) La rampa de colores para aplicar
- Se aplica la configuración a la capa deseada **layer.setRenderer(renderer)**

# Creación de Plugin

- Se requiere instalar dos plugins:
  - Plugin Builder
  - Plugin Reloader



The image shows a screenshot of the 'QGIS Plugin Builder - 3.2.1' dialog box. The dialog has a dark gray background and a title bar with the text 'QGIS Plugin Builder - 3.2.1' and a red close button. The main title 'QGIS Plugin Builder' is displayed in a large, bold, white font. Below the title, there are several input fields with labels to their left. The fields are: 'Class name' with the value 'ClasificadorCovid', 'Plugin name' with the value 'Clasificador Covid', 'Description' with the value 'Obtiene el porcentaje de contagios por municipio', 'Module name' with the value 'clasificador\_covid', 'Version number' with the value '1', 'Minimum QGIS version' with the value '3.0', 'Author/Company' with the value 'Emmanuel', and 'Email address' with the value 'djnonasrm@gmail.com'. At the bottom of the dialog, there are four buttons: 'Ayuda' (highlighted with a yellow border), '<Previous', 'Next>', and 'Cancelar'.

Field	Value
Class name	ClasificadorCovid
Plugin name	Clasificador Covid
Description	Obtiene el porcentaje de contagios por municipio
Module name	clasificador_covid
Version number	1
Minimum QGIS version	3.0
Author/Company	Emmanuel
Email address	djnonasrm@gmail.com

Ayuda <Previous Next> Cancelar

# Creación de Plugin



# Creación de Plugin



The image shows a screenshot of the 'QGIS Plugin Builder - 3.2.1' dialog box. The dialog has a dark gray background and a title bar with the text 'QGIS Plugin Builder - 3.2.1' and a red close button. The main title 'QGIS Plugin Builder' is displayed in a large, bold font. Below the title, there are three input fields: 'Template' with a dropdown menu showing 'Tool button with dialog', 'Text for the menu item' with a text box containing 'Clasificador Covid por Municipio', and 'Menu' with a dropdown menu showing 'Plugins'. At the bottom of the dialog, there are four buttons: 'Ayuda', '<Previous', 'Next>', and 'Cancelar'. The 'Next>' button is highlighted with a yellow border.

QGIS Plugin Builder - 3.2.1

## QGIS Plugin Builder

Template: Tool button with dialog

Text for the menu item: Clasificador Covid por Municipio

Menu: Plugins

Ayuda <Previous Next> Cancelar

# Creación de Plugin



# Creación de Plugin

QGIS Plugin Builder - 3.2.1

## QGIS Plugin Builder

**Publication (mandatory Items)**

Bug tracker

Repository

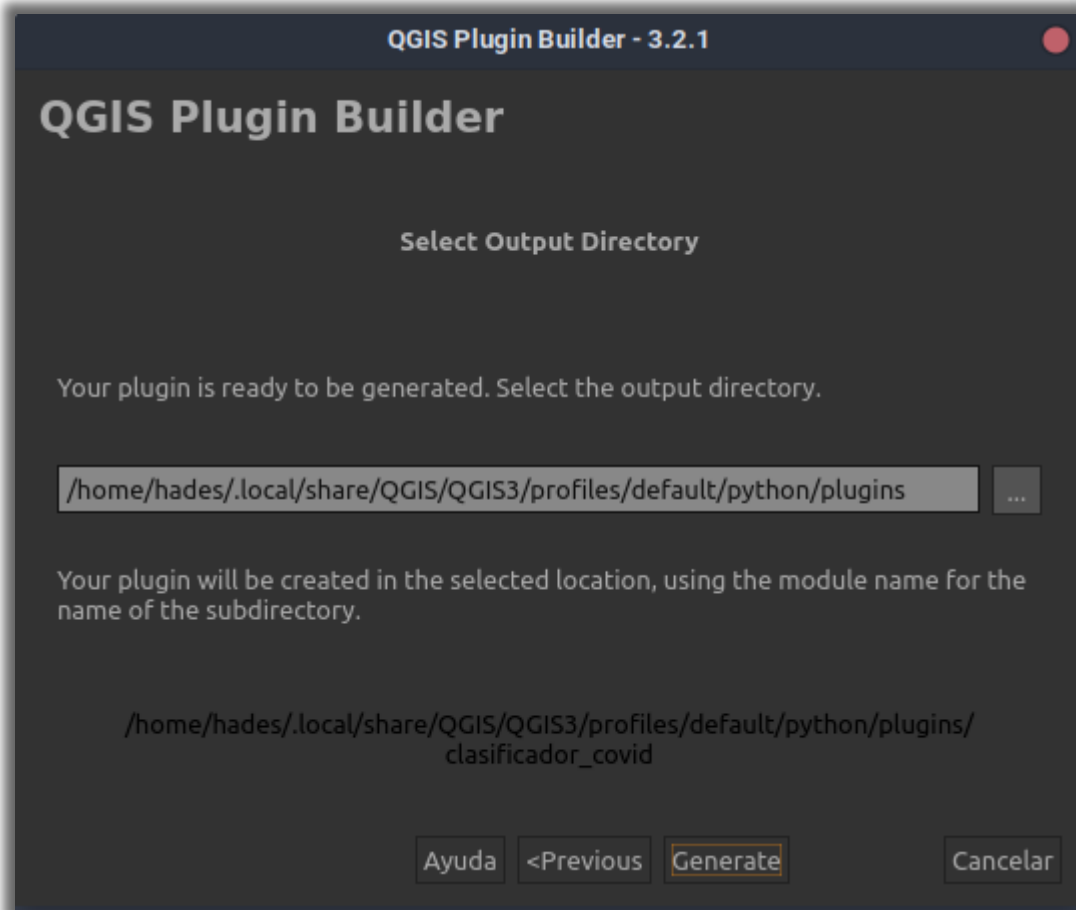
**Publication (recommended Items)**

Home page

Tags

☐ Flag the plugin as experimental

# Creación de Plugin



```
/Users/nonas/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins
```

```
/home/hades/.local/share/QGIS/QGIS3/profiles/default/python/plugins
```

# Creación de Plugin

## Plugin Builder Results

Congratulations! You just built a plugin for QGIS!

Your plugin **ClasificadorCovid** was created in:

`/home/hades/.local/share/QGIS/QGIS3/profiles/default/python/plugins/clasificador_covid`

Your QGIS plugin directory is located at:

`/home/hades/.local/share/QGIS/QGIS3/profiles/default/python/plugins`

## What's Next

1. If `resources.py` is not present in your plugin directory, compile the resources file using `pyrcc5` (simply use **pb\_tool** or **make** if you have automake)
2. Optionally, test the generated sources using **make test** (or run tests from your IDE)
3. Copy the entire directory containing your new plugin to the QGIS plugin directory (see Notes below)
4. Test the plugin by enabling it in the QGIS plugin manager
5. Customize it by editing the implementation file **clasificador\_covid.py**
6. Create your own custom icon, replacing the default **icon.png**
7. Modify your user interface by opening **clasificador\_covid\_dialog\_base.ui** in Qt Designer

## Notes:

- You can use **pb\_tool** to compile, deploy, and manage your plugin. Tweak the `pb_tool.cfg` file included with your plugin as you add files. Install **pb\_tool** using `pip` or `easy_install`. See [http://loc8.cc/pb\\_tool](http://loc8.cc/pb_tool) for more information.
- You can also use the **Makefile** to compile and deploy when you make changes. This requires GNU make (gmake). The Makefile is ready to use, however you will have to edit it to add additional Python source files, dialogs, and translations.

For information on writing PyQGIS code, see [http://loc8.cc/pyqgis\\_resources](http://loc8.cc/pyqgis_resources) for a list of resources.

©2011-2019 GeoApt LLC - [geoapt.com](http://geoapt.com)

Aceptar

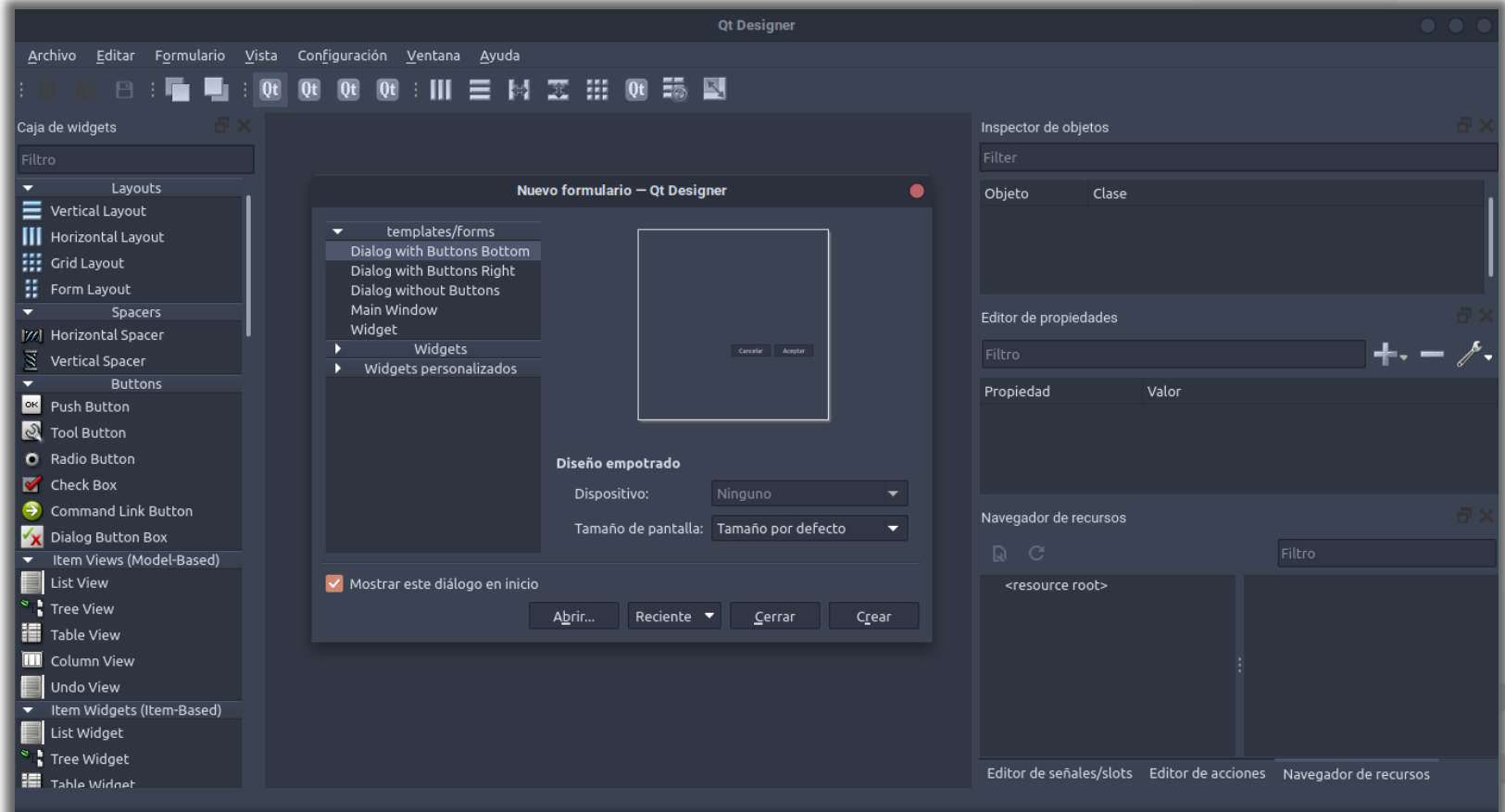


# Creación de Plugin

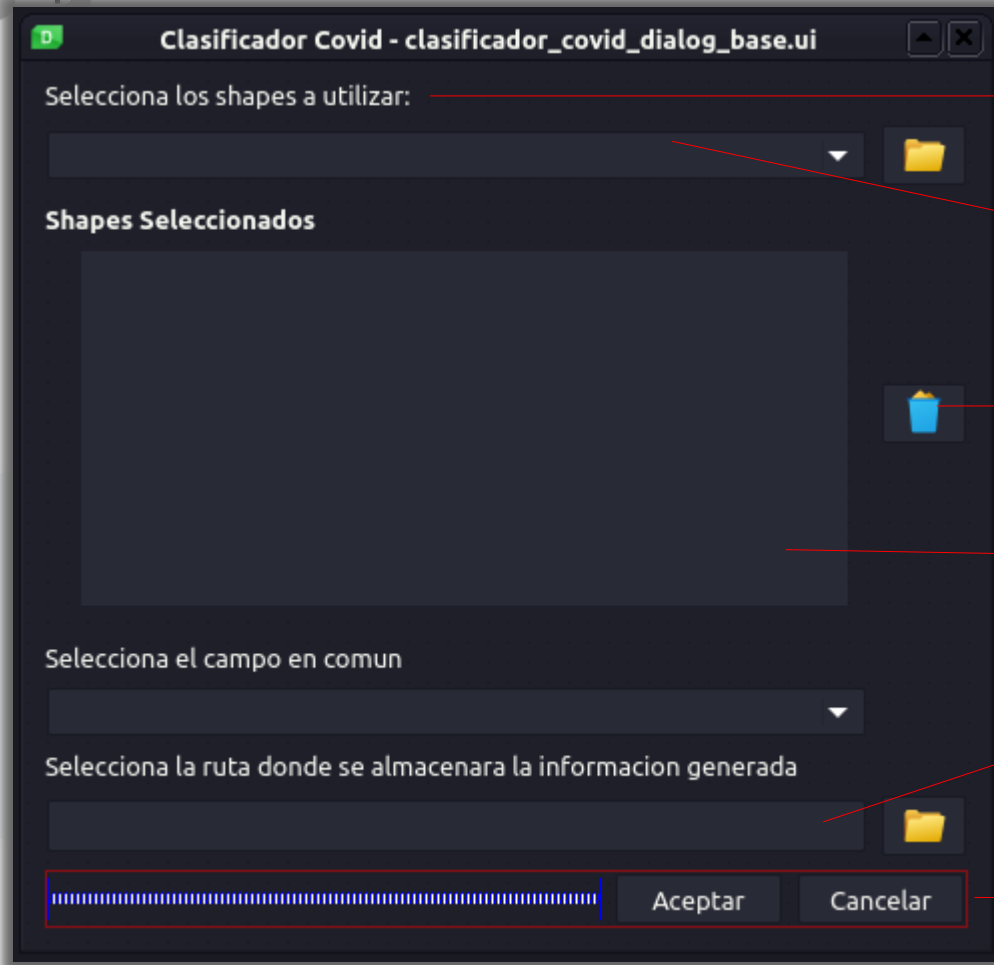
- Para compilar los archivos que necesitaremos se ejecutan las siguientes instrucciones
  - Para todos los recursos (iconos) - **pyrcc5 -o resources.py resources.qrc**
  - Para la interfaz gráfica - **pyuic5 -x file.ui -o file.py**
- En Windows se tiene que realizar a por medio de OSGeo4W
- En Sistemas Unix se puede realizar desde la terminal
- **Nota:** Cada que se realiza un ajuste a la interfaz gráfica o a los iconos, se tiene que volver a compilar uno, otro o ambos.

# Creación de Plugin

- Abrir Qt Designer



# Creación de Plugin



QLabel (Label)

Qcombobox (Combobox)

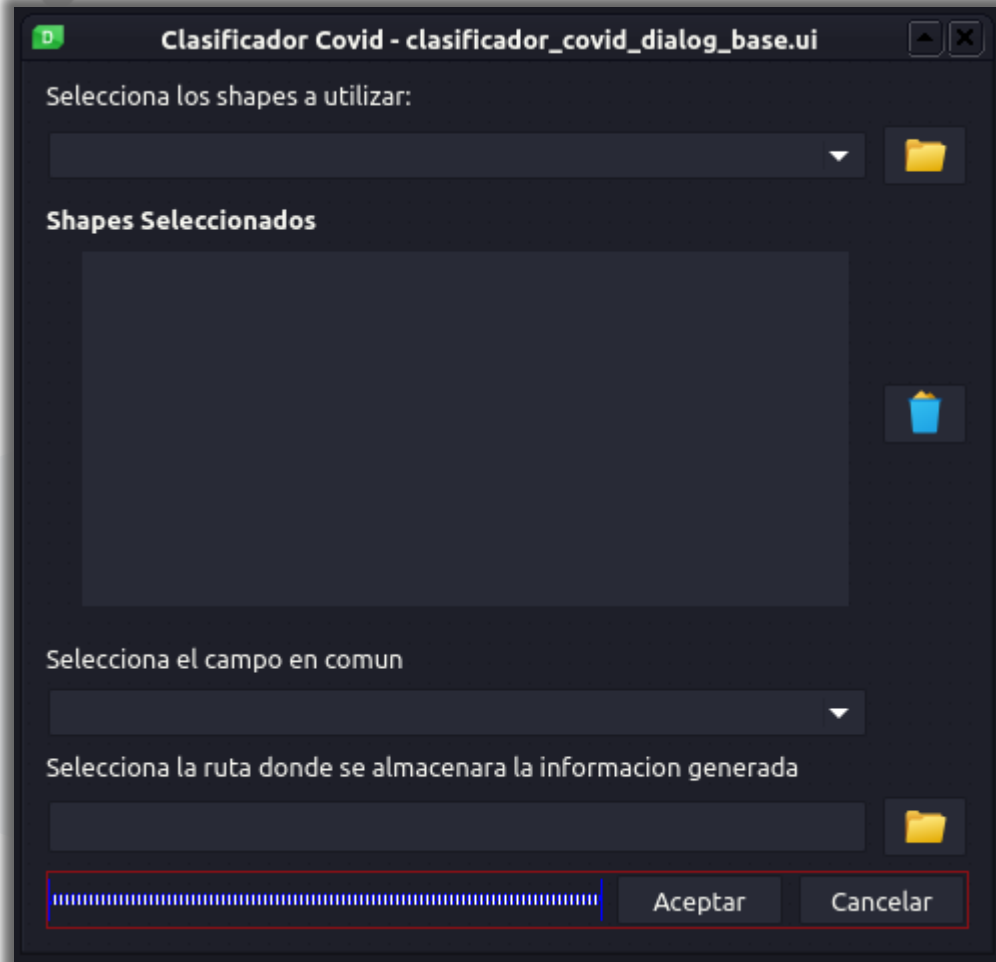
QPushButton (PushButton)

QGroupBox (groupBox)  
QListWidget (listWidget)

QLineEdit (LineEdit)

QHBoxLayout (HorizontalLayout)  
Spacer (HorizontalSpacer)

# Creación de Plugin



- Al modificar el archivo .ui y el resources, es necesario volver a compilar

# Creación de Plugin

- En el archivo **\*\_dialog.py** es necesario modificar la siguiente línea de código de **FORM\_CLASS**
- Importaremos el archivo **.\*\_dialog\_base** con su respectiva clase **UI\_\***

```
# This loads your .ui file so that PyQt can populate your plugin with the elements from Qt Designer
# FORM_CLASS, _ = uic.loadUiType(os.path.join(
#     os.path.dirname(__file__), 'clasificador_covid_dialog_base.ui'))

from .clasificador_covid_dialog_base import Ui_ClasificadorCovidDialogBase

class ClasificadorCovidDialog(QDialog, Ui_ClasificadorCovidDialogBase):
    def __init__(self, parent=None):
```

# Creación de Plugin

- En el archivo **\*\_dialog\_base.py** necesitamos comentar las siguientes líneas y además modificar la línea del **resource**

```
from .resources import *

# if __name__ == "__main__":
#     import sys
#     app = QtWidgets.QApplication(sys.argv)
#     ClasificadorCovidDialogBase = QtWidgets.QDialog()
#     ui = Ui_ClasificadorCovidDialogBase()
#     ui.setupUi(ClasificadorCovidDialogBase)
#     ClasificadorCovidDialogBase.show()
#     sys.exit(app.exec_())
```

# Creación de Plugin

- En el archivo con el **nombre de nuestro plugin.py**, realizaremos varias modificaciones.
- La primera de ellas es en el método constructor **`__init__`**

```
# Declare instance attributes
self.actions = []
self.menu = self.tr(u'&Clasificador Covid')

# agregando nuevas configuracion personal
self.action = None
self.toolbar = self.iface.addToolBar(u'ClasificadorCovid')
self.toolbar.setObjectName(u'ClasificadorCovid')

self.dlg = ClasificadorCovidDialog()

# Check if plugin was started the first time in current QGIS session
# Must be set in initGui() to survive plugin reloads
# self.first_start = None
```

# Creación de Plugin

- En el archivo con el **nombre de nuestro plugin.py**, realizaremos varias modificaciones.
- La segunda de ellas es el método **add\_action**, el cual será eliminado completamente o en su defecto comentado.

```
# 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):  
#     """Add a toolbar icon to the toolbar.  
  
#     :param icon_path: Path to the icon for this action. Can be a resource  
#         path (e.g. ':/plugins/foo/bar.png') or a normal file system path.  
#     :type icon_path: str
```



# Creación de Plugin

- En el archivo con el **nombre de nuestro plugin.py**, realizaremos varias modificaciones.
- La tercera de ellas es en el método **initGui**

```
def initGui(self):  
    # """Create the menu entries and toolbar icons inside the QGIS GUI."""  
  
    # icon_path = './plugins/clasificador_covid/icon.png'  
    # self.add_action(  
    #     icon_path,  
    #     text=self.tr(u'Clasificador Covid por Municipio'),  
    #     callback=self.run,  
    #     parent=self.iface.mainWindow()  
    # )  
  
    # # will be set False in run()  
    # self.first_start = True  
    icon_path = './plugins/clasificador_covid/icon.png'  
    icon = QIcon(icon_path)  
    self.action = QAction(icon, "Clasificador Covid por Municipio", self.iface.mainWindow())  
    self.action.setEnabled(True)  
    self.toolbar.addAction(self.action)  
    self.action.triggered.connect(self.run)  
    self.iface.addPluginToMenu(self.menu, self.action)  
    self.actions.append(self.action)
```

# Creación de Plugin

- En el archivo con el **nombre de nuestro plugin.py**, realizaremos varias modificaciones.
- La cuarta de ellas es el método **unload**, aquí únicamente se agrega la línea **del self.toolbar**

```
def unload(self):  
    """Removes the plugin menu item and icon from QGIS GUI."""  
    for action in self.actions:  
        self.iface.removePluginMenu(  
            self.tr(u'&Clasificador Covid'),  
            action)  
        self.iface.removeToolBarIcon(action)  
    del self.toolbar
```

# Creación de Plugin

- En el archivo con el **nombre de nuestro plugin.py**, realizaremos varias modificaciones.
- La quinta de ellas es el método **run**

```
def run(self):  
    """Run method that performs all the real work"""  
  
    # # Create the dialog with elements (after translation) and keep reference  
    # # Only create GUI ONCE in callback, so that it will only load when the plugin is started  
    # if self.first_start == True:  
    #     self.first_start = False  
    #     self.dlg = ClasificadorCovidDialog()  
  
    # # show the dialog  
    # self.dlg.show()  
    # # Run the dialog event loop  
    # result = self.dlg.exec_()  
    # # See if OK was pressed  
    # if result:  
    #     # Do something useful here - delete the line containing pass and  
    #     # substitute with your code.  
    #     pass  
  
    self.dlg.show()  
    result = self.dlg.exec_()    ■ "result" is not accessed
```

# Creación de Plugin

## Métodos y señales de los objetos de la interfaz gráfica

- **QComboBox - Métodos**

- **QComboBox.addItem(Str, Object)** – Permite agregar cadenas de y texto y objetos al combobox.
- **QComboBox.itemData(index)** – Permite obtener el objeto a través de la identificación del índice correspondiente.
- **QComboBox.currentIndex()** – Retorna el índice actual del combobox.
- **QComboBox.clear()** – Elimina todos los registros que se encuentren dentro del combobox.
- **QComboBox.setStyleSheet(CSS)** – Permite agregar estilos personalizados con formato de CSS (agrega color, tamaño, forma, sombras, etc.)
- **QComboBox.currentText()** – Retorna el texto actual del combobox.

# Creación de Plugin

## Métodos y señales de los objetos de la interfaz gráfica

- **QcomboBox - Señales**

- `QComboBox.currentIndexChanged.connect(fn)` – Detecta la selección de un elemento dentro del combobox a través de su índice.

- **QPushButton – Señales**

- `QPushButton.clicked.connect(fn)` – Detecta cuando se ha dado clic en un botón.

- **QLineEdit – Métodos**

- `QLineEdit.setText(str)` – Permite agregar texto a cuadro de texto.
- `QLineEdit.text()` - Retorna el valor que existe en el cuadro de texto.

# Creación de Plugin

## Métodos y señales de los objetos de la interfaz gráfica

- **QLineEdit – Métodos**

- QLineEdit.**clear()** – Elimina todo lo existente en el cuadro de texto.
- QLineEdit.**setStyleSheet(CSS)** – Permite agregar estilos personalizados con formato de CSS (agrega color, tamaño, forma, sombras, etc.)

- **QListWidget– Métodos**

- QListWidget.**addItem(str)** – Permite agregar un objeto (texto) a la lista.
- QListWidget.**item(index)** – Retorna el texto de acuerdo a un índice dado.
- QListWidget.**count()** – Retorna el numero de objetos (texto) que se encuentran dentro de la lista.

# Creación de Plugin

## Métodos y señales de los objetos de la interfaz gráfica

- **QListWidget- Métodos**

- `QListWidget.selectedItems()` – Retorna todos los objetos seleccionados que se encuentran dentro de la lista.
- `QListWidget.row(str)` – Retorna el índice del objeto dentro de la lista.
- `QListWidget.takeItem(row)` – Elimina un elemento de la lista y retornar la lista restante.
- `QListWidget.clear()` – Elimina todos los elementos de la lista.

- **QListWidget- Señales**

- `QListWidget.itemClicked.connect(fn)` – Detecta cuando se ha dado clic sobre un elemento de la lista

# Creación de Plugin

## Librerías

```
# nuevas librerías  
from qgis.core import *  
from PyQt5.QtWidgets import *
```



# Creación de Plugin

## Interacción con el administrador de archivos

- **QFileDialog.getOpenFileName(IU, texto, "", tipoArchivo)**
  - Permite obtener la dirección (path) de un archivo en específico en el administrador de archivos.
  - **Parámetros**
    - **IU** – Objeto (Interfaz gráfica)
    - **Texto** – Mensaje que se mostrará al usuario en la ventana del administrador de archivos.
    - **""** - Por default
    - **TipoArchivo** – String que ayuda a identificar el tipo de archivo para filtrar, ejemplo: "\*.shp"

# Creación de Plugin

## Interacción con el administrador de archivos

- **QFileDialog.getExistingDirectory(UI, Texto)**
  - Permite obtener la selección de una carpeta existente en el administrador de archivos del sistema operativo.
  - **Parámetros**
    - **IU** – Objeto (Interfaz gráfica)
    - **Texto** – Mensaje que se mostrará al usuario en la ventana del administrador de archivos.

# Creación de Plugin

## Mensajes

```
def set_message(self, title, text, error, tipo):  
    msjBox = QMessageBox()  
    msjBox.setIcon(tipo)  
    msjBox.setText(text)  
    msjBox.setInformativeText("")  
    msjBox.setWindowTitle(title)  
    msjBox.setDetailedText(error)  
    msjBox.setStandardButtons(QMessageBox.Ok)  
    msjBox.exec_()
```