
Aula 4

2024- 10 - 14

ANA LUISA MAFFINI
2024

Funções

Funções

Em Python, funções são blocos de código reutilizáveis que executam uma tarefa específica.

Eles permitem que você divida seu programa em partes menores e mais gerenciáveis, tornando seu código mais organizado, legível e mais fácil de manter.

Definição de Função

Para criar uma função, use a palavra-chave **def** seguida do nome da função e parênteses ().

O nome da função deve seguir as convenções de nomenclatura do Python. Se sua função aceita parâmetros, liste-os entre parênteses.

Invocação de função (chamando uma função)

Para executar uma função e fazê-la executar a tarefa definida, você a "chama" usando seu nome seguido de parênteses ().

Funções

Definição de Função

Para criar uma função, use a palavra-chave **def** seguida do nome da função e parênteses (). O nome da função deve seguir as convenções de nomenclatura do Python. Se sua função aceita parâmetros, liste-os entre parênteses.

```
def parabens() :  
    print("Parabéns!")
```

Invocação de função (chamando uma função)

Para executar uma função e fazê-la executar a tarefa definida, você a "chama" usando seu nome seguido de parênteses ().

```
parabens()    # Isso irá escrever: "Parabéns!"
```

Funções

Parâmetros e argumentos

As funções podem aceitar valores de entrada chamados parâmetros. Os parâmetros são especificados na definição da função. Ao chamar a função, você passa valores reais chamados argumentos.

```
def parabens_user(name):  
    print(f"Parabéns, {name}!")
```

```
parabens_user("Alice")    # Isso irá escrever: "Parabéns, Alice!"
```

Declaração de retorno (return)

As funções podem retornar valores usando a instrução **return**. Isso permite que a função envie dados de volta ao chamador.

```
def add(a, b):  
    return a + b
```

```
result = add(5, 3)    # result will be 8
```

Funções

Escopo das Variáveis

Variáveis definidas dentro de uma função têm escopo local, o que significa que só são acessíveis dentro dessa função.

Variáveis definidas fora das funções têm escopo global e podem ser acessadas de qualquer lugar no código.

Funções

Documentos (docstrings)

É uma boa prática incluir uma docstring (uma string entre aspas triplas) no início de uma função para descrever sua finalidade, parâmetros, valores de retorno e quaisquer detalhes importantes.

```
def calculate_square(num):  
    """  
    Calcula o quadrado de um número.  
  
    Args:  
        num (int): O número que será elevado ao quadrado.  
  
    Returns:  
        int: O quadrado do número informado.  
    """  
    return num ** 2
```

Funções

Argumentos padrão (default)

Você pode definir valores padrão para parâmetros de função. Se o chamador não fornecer um valor para esses parâmetros, os valores padrão serão usados.

```
def parabens_user(name="Guest") :  
    print(f"Parabéns, {name}!")
```

```
parabens_user()    # Isso irá escrever: "Parabéns, Guest!"
```

```
parabens_user("Alice")    # Isso irá escrever: "Parabéns, Alice!"
```


Funções

F-Strings

Em Python, uma 'f-string' é uma forma de formatar strings que permite incorporar expressões dentro de literais de string.

O f no início da string indica que é uma string f, e as expressões entre chaves {} são avaliadas e substituídas por seus valores.

```
def parabens_user(name="Guest") :  
    print(f"Parabéns, {name}!")
```

```
parabens_user("Alice")    # Isso irá escrever: "Parabéns, Alice!"
```

Neste caso, {name} é uma expressão dentro da string f. Python substitui {name} pelo valor do nome da variável ao imprimir a string.

Funções para Plugin no QGIS

Funções

Em um plugin para QGIS, as funções normalmente são definidas dentro de uma classe herdada de uma classe relevante do QGIS, como **QgsProcessingAlgorithm**. Esta classe deve incluir os métodos e propriedades necessários para definir e executar o algoritmo de processamento.

- 1) **Defina uma classe:** Comece definindo uma classe para o seu plugin. Esta classe deve herdar de uma classe base do QGIS. Por exemplo, se você estiver criando um algoritmo de processamento, você herdará de `QgsProcessingAlgorithm`.

```
from qgis.core import QgsProcessingAlgorithm

class YourAlgorithmClass(QgsProcessingAlgorithm):
    def initAlgorithm(self, config=None):
        # Código de inicialização vai aqui
        pass

    def processAlgorithm(self, parameters, context, feedback):
        # Código do algoritmo de processamento vai aqui
        pass
```

Funções

- 2) **Inicialize o algoritmo:** Use o método **initAlgorithm** para definir as entradas e saídas do seu algoritmo. É aqui que você adiciona parâmetros como camadas vetoriais, números e coletores.

```
def initAlgorithm(self, config=None):  
    self.addParameter(QgsProcessingParameterFeatureSource("INPUT_LAYER",  
"Input Layer"))  
    self.addParameter(QgsProcessingParameterFeatureSink("OUTPUT_LAYER",  
"Output Layer"))
```

Funções

- 3) **Processe o Algoritmo:** Implemente o método **processAlgorithm** para definir o que acontece quando o algoritmo é executado. É aqui que você escreve o código que realiza o processamento desejado.

```
def processAlgorithm(self, parameters, context, feedback):  
    input_layer = self.parameterAsVectorLayer(parameters, "INPUT_LAYER",  
context)  
    output_sink, dest_id = self.parameterAsSink(parameters, "OUTPUT_LAYER",  
context, input_layer.fields(), input_layer.wkbType(),  
input_layer.sourceCrs())  
  
    # O restante do seu código vai aqui  
  
    return {"OUTPUT_LAYER": dest_id}
```

Funções

- 4) **Execute o plug-in:** Para executar o plugin, normalmente você precisa criar uma instância da sua classe de algoritmo e executá-la através da estrutura de processamento QGIS. Isso geralmente é feito em resposta a uma ação do usuário, como clicar em um botão na GUI do seu plugin.

```
algorithm = YourAlgorithmClass()  
QgsProcessingAlgorithm.execute(algorithm, parameters, context, feedback)
```

Os parâmetros, contexto e argumentos de feedback dependem do contexto em que seu plugin está sendo executado.

Desenvolvimento de Plugin para o QGIS

Desenvolvimento de Plugin para o QGIS

Plugin:

Plugins é uma possibilidade para a comunidade de usuários agregar valor ao QGIS. Alguns plugins são incorporados ao software principal.

É possível criar plugins na linguagem de programação Python. Em comparação com plugins clássicos escritos em C++, eles são mais fáceis de escrever, entender, manter e distribuir devido à natureza dinâmica da linguagem Python.

Existem duas (três) opções:

1. Escreva um plugin python vai para o menu de plugins
2. Escreva um plugin C++ (não sei sobre isso)
3. Escreva um plugin de processamento para a caixa de ferramentas de processamento

Desenvolvimento de Plugin para o QGIS

Algoritmos de Processamento:

Esses plugins permitem que os usuários criem ferramentas de análise geoespacial personalizadas, implementando novos algoritmos ou agrupando funcionalidades de geoprocessamento existentes. Eles usam o Processing Framework no QGIS.

Plugins baseados em GUI:

Plugins baseados em GUI aprimoram a interface do QGIS adicionando novos painéis, widgets ou ferramentas. Estes podem variar desde ferramentas simples até interfaces complexas que facilitam tarefas geoespaciais específicas.

Desenvolvimento de Plugin para o QGIS

User plugins ficam localizados em:

C:\Users\YOURUSERNAME\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins

Os arquivos necessários são:

init__.py = A inicialização do plugin. Precisa ter classFactory(), method e pode ter outro código de inicialização.

mainPlugin.py = O código principal do plugin. Contém toda a informação sobre as ações do plugin e o código principal.

resources.qrc = Documento .xml criado pelo Qt Designer. Contém os caminhos relativos aos recursos dos formulários.

resources.py = A tradução do arquivo qrc descrito acima para Python.

form.ui = A GUI criada pelo Qt Designer.

form.py = A tradução do form.ui descrito acima para Python.

metadata.txt = Contém informações gerais, versão, nome, e outros metadados usados pelos websites e estruturas dos plugins.

Desenvolvimento de Plugin para o QGIS

Preparações:

Prepare um repositório para compartilhar seu código e para os usuários relatarem problemas. Utilizaremos o GitHub (www.github.com)

Se você não tem uma conta, crie uma.

Crie um novo repositório.

Utilizaremos outros plugins para o desenvolvimento de plugins (**Plugin Builder** e **Plugin Reloader**)

Também usaremos pb_tool para implantar, etc. o nosso plugin

Criando Plugin

Plugin Builder



É um construtor de plug-ins no QGIS.

O Plugin Builder fornece um modelo funcional a partir do qual você pode criar seu próprio plugin.

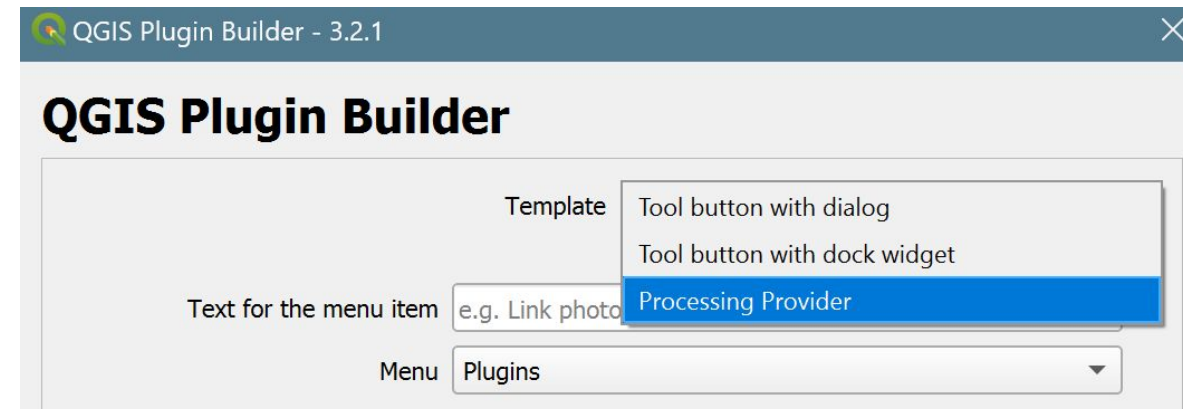
As etapas para usar o Plugin Builder são bastante simples:

- Abra o Plugin Builder no QGIS
- Preencha as informações necessárias para o modelo de plugin selecionado
- Designe onde armazenar seu novo plugin
- Se a compilação automática falhar durante a geração do plugin, compile manualmente seu arquivo de recursos usando pyrcc5
- Instale o plug-in
- Teste

Criando Plugin

Existem três formas de criar um plugin com o Plugin Builder: **Tool Button with dialog, Tool button with dock widget, Processing Provider.**

Nós vamos utilizar o Processing Provider.

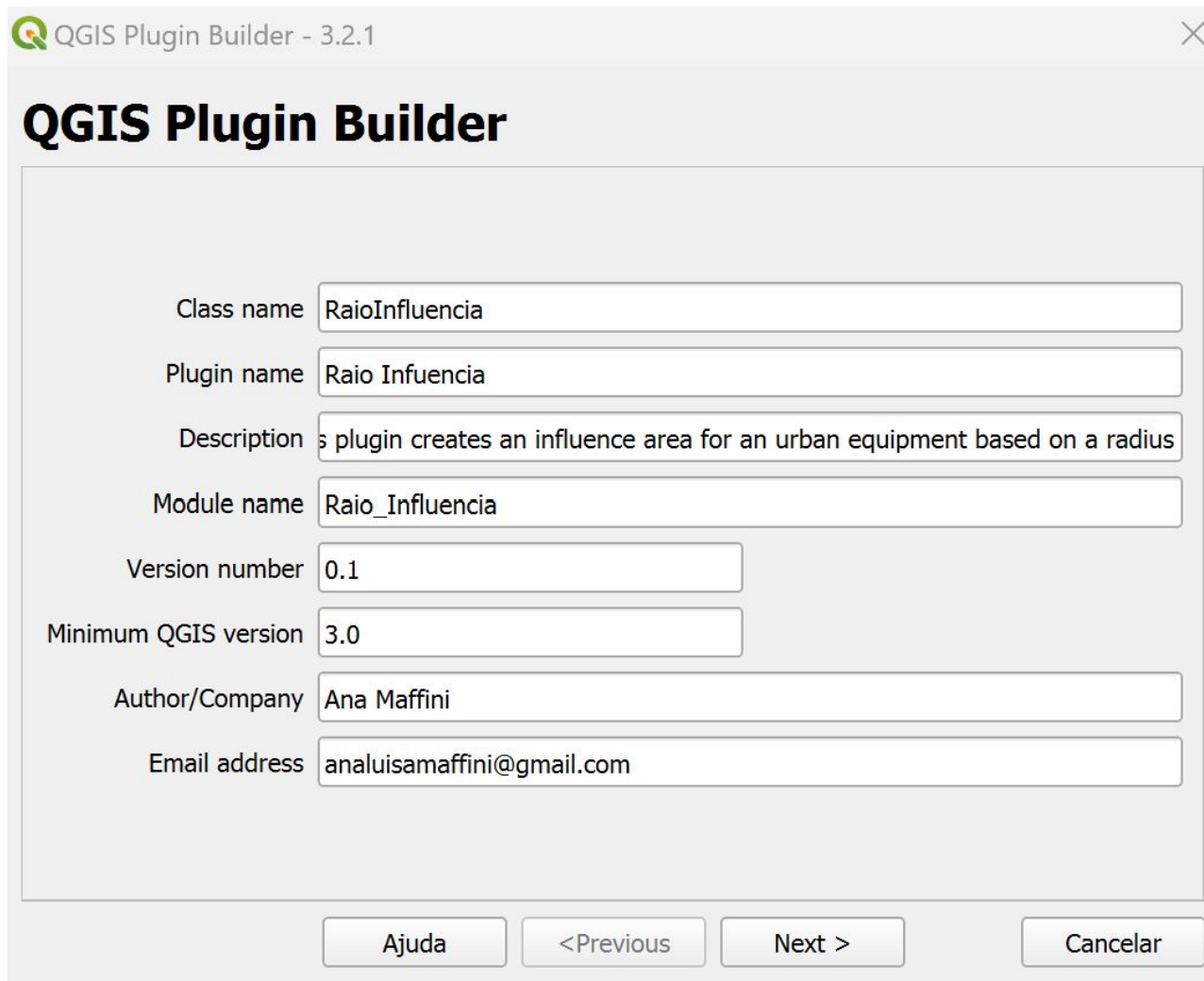


Algumas vantagens de fazer um plugin usando a estrutura de processamento:

- Uma API comum torna o plugin mais acessível, por exemplo para o construtor de modelo, como um processo em lote ou como uma função Python independente.
- Seu plugin é automaticamente portado para um thread no seu computador (o computador não irá congelar).
- Nenhum arquivo de interface do usuário é necessário (também pode ser uma desvantagem).

Criando Plugin

No QGIS, em Complementos, selecione Plugin Builder.



The screenshot shows the 'QGIS Plugin Builder - 3.2.1' window. It contains several input fields for plugin metadata. The fields are: 'Class name' with value 'RaioInfluencia', 'Plugin name' with value 'Raio Infuencia', 'Description' with value 's plugin creates an influence area for an urban equipment based on a radius', 'Module name' with value 'Raio_Influencia', 'Version number' with value '0.1', 'Minimum QGIS version' with value '3.0', 'Author/Company' with value 'Ana Maffini', and 'Email address' with value 'analuisamaffini@gmail.com'. At the bottom, there are four buttons: 'Ajuda', '<Previous', 'Next >', and 'Cancelar'.

QGIS Plugin Builder - 3.2.1

QGIS Plugin Builder

Class name: RaioInfluencia

Plugin name: Raio Infuencia

Description: s plugin creates an influence area for an urban equipment based on a radius

Module name: Raio_Influencia

Version number: 0.1

Minimum QGIS version: 3.0

Author/Company: Ana Maffini

Email address: analuisamaffini@gmail.com

Ajuda <Previous Next > Cancelar

Criando Plugin

QGIS Plugin Builder - 3.2.1

QGIS Plugin Builder

About

This plugin creates an influence area for an urban equipment based on a radius.

Ajuda <Previous Next> Cancelar

QGIS Plugin Builder - 3.2.1

QGIS Plugin Builder

Template Processing Provider

Algorithm name Raios Influencia Processing

Algorithm group e.g. Algorithms for vector layers

Provider name e.g. Example provider

Provider description e.g. Example algorithms

Ajuda <Previous Next> Cancelar

Criando Plugin

QGIS Plugin Builder - 3.2.1

QGIS Plugin Builder

- ☒ Internationalization
- ☒ Help
- ☒ Unit tests
- ☒ Helper scripts
- ☒ Makefile
- ☒ pb_tool

Ajuda <Previous Next> Cancelar

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

Ajuda <Previous Next> Cancelar

Criando Plugin

 QGIS Plugin Builder - 3.2.1

QGIS Plugin Builder

Select Output Directory

Your plugin is ready to be generated. Select the output directory.

C:/Users/analú/OneDrive/Documents/GitHub/Trabalho_PROPUR

...

Your plugin will be created in the selected location, using the module name for the name of the subdirectory.

C:/Users/analú/OneDrive/Documents/GitHub/Trabalho_PROPUR\Raio_Influencia

Ajuda

<Previous

Generate

Cancelar

Criando Plugin

No VSCode, abra a pasta do plugin criado.

Em seguida, adicione os seguintes códigos em **Raio_Influencia_provider.py**:

Na função id:

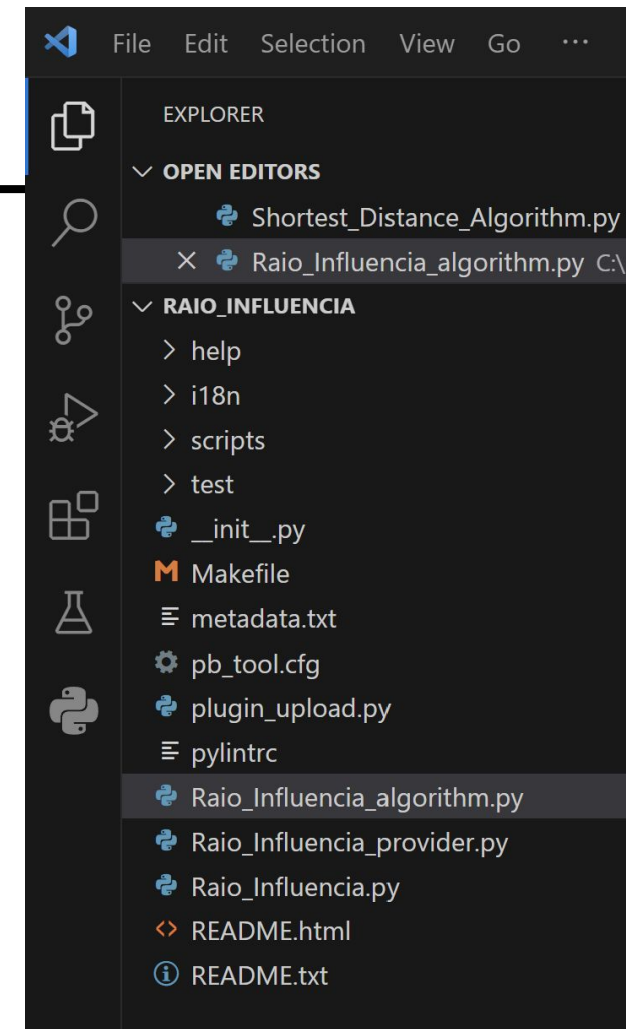
```
return 'RaioInfluencia'
```

Na função name:

```
return 'Raio Influencia'
```

No arquivo **pb_tool.cfg** adicione:

```
python_files:          __init__.py          Raio_Influencia.py
Raio_Influencia_algorithm.py Raio_Influencia_provider.py
```



Criando Plugin

Confira se o **pb_tool** está instalado corretamente, se sim, execute `pb_tool deploy` no terminal para ativar o plugin no QGIS.

Feche o QGIS e abra novamente.

Utilize o **Plugin Reloader** para atualizar seu plugin no QGIS.



Criando Plugin

Agora começamos a adicionar o código!

No arquivo **Raio_Influencia_algorithm.py**:

Substitua OUTPUT e INPUT nas linhas 59 e 60 por:

```
INPUT_VECTOR = 'INPUT_VECTOR'  
RADIUS = 'RADIUS'  
OUTPUT_VECTOR = 'OUTPUT_VECTOR'
```

Criando Plugin

Na função **initAlgorithm** fica o que o usuário irá ver:

```
self.plugin_dir = os.path.dirname(__file__) #necessário para
salvar os arquivos temporários depois
```

```
self.addParameter(
    QgsProcessingParameterFeatureSource(
        self.INPUT_VECTOR,
        self.tr('Input vector layer'),
        [QgsProcessing.TypeVectorPoint]))
```

```
self.addParameter(
    QgsProcessingParameterNumber(
        self.RADIUS,
        self.tr('Influence Radius'),
        QgsProcessingParameterNumber.Double,
        defaultValue=0.0))
```

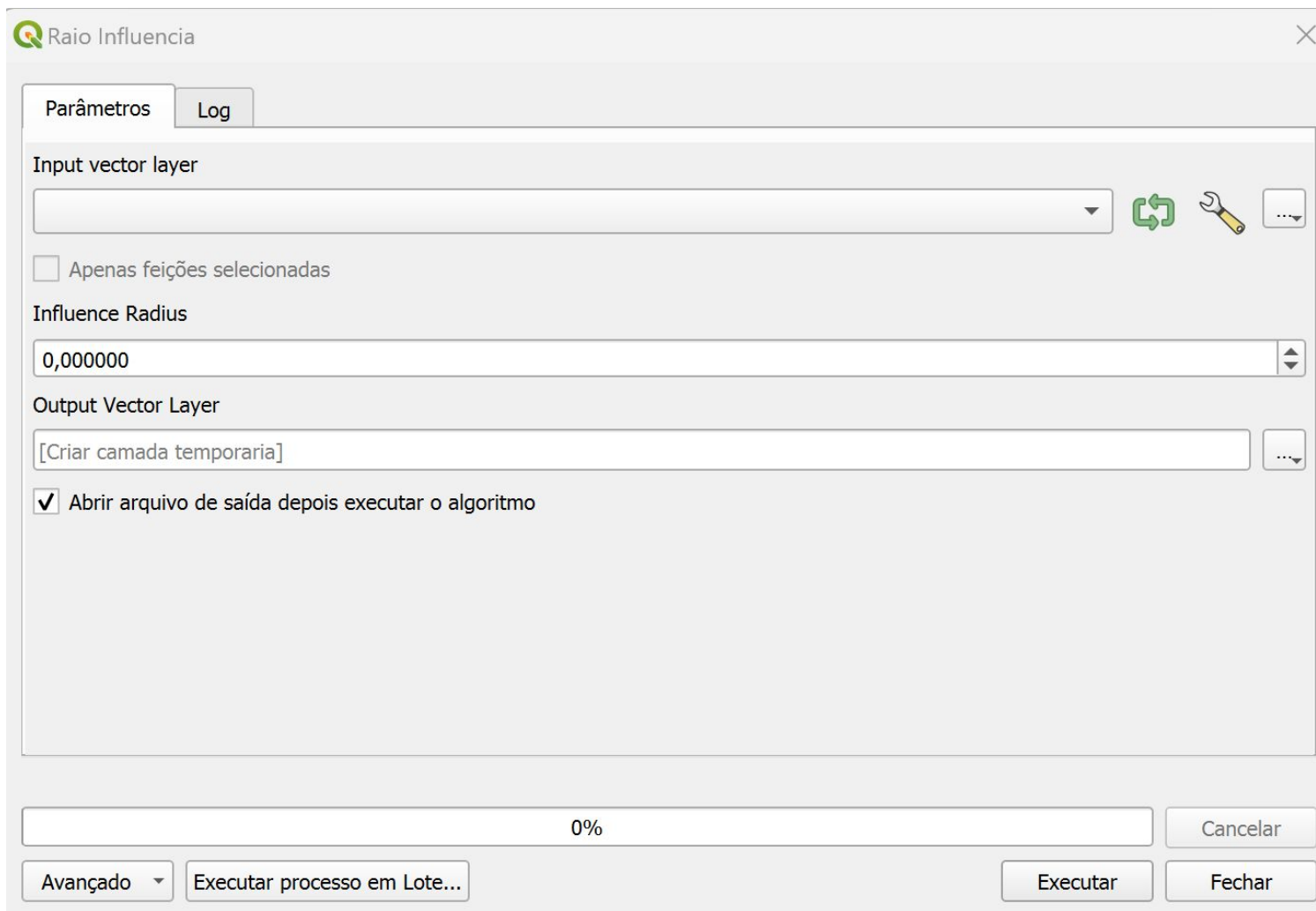
Criando Plugin

Na função **initAlgorithm** fica o que o usuário irá ver:

```
self.addParameter(  
    QgsProcessingParameterFeatureSink(  
        self.OUTPUT_VECTOR,  
        self.tr('Output Vector Layer'),  
        type=QgsProcessing.TypeVectorPolygon,  
        createByDefault=False,  
        supportsAppend=True,  
        defaultValue=None))
```

Criando Plugin

Agora a interface do seu plugin deverá estar assim:



The image shows a software window titled "Raio Influencia" with a close button (X) in the top right corner. The window has two tabs: "Parâmetros" (selected) and "Log".

Under the "Parâmetros" tab, the interface includes:

- Input vector layer:** A dropdown menu, a refresh icon (two green arrows), a settings icon (wrench), and a help icon (question mark).
- ☐ Apenas feições selecionadas
- Influence Radius:** A numeric input field containing "0,000000" and a vertical scrollbar.
- Output Vector Layer:** A text field containing "[Criar camada temporaria]" and a help icon.
- ☒ Abrir arquivo de saída depois executar o algoritmo

At the bottom of the window, there is a progress bar showing "0%", a "Cancelar" button, and a status bar with a dropdown menu set to "Avançado", a button "Executar processo em Lote...", and two buttons "Executar" and "Fechar".

Criando Plugin

Na função **processAlgorithm** fica o que o o algoritmo faz:

Aqui primeiro precisamos acessar nossos parâmetros (Input, Radius, Output):

```
# InputParameters
        vlayer = self.parameterAsVectorLayer(parameters,
self.INPUT_VECTOR, context)
        r = self.parameterAsDouble(parameters, self.RADIUS, context)
        output_sink, dest_id = self.parameterAsSink(parameters,
self.OUTPUT_VECTOR, context, vlayer.fields(), QgsWkbTypes.Polygon,
vlayer.sourceCrs())
```


Criando Plugin

Aqui adicionamos o código do cálculo do buffer:

```
# Create buffer around points
for feat in vlayer.getFeatures():
    geom = feat.geometry()
    buffered_geom = geom.buffer(r, 5)
```

Criando Plugin

Agora adicionamos o código para criar os parâmetros da nova geometria do buffer e o resultado:

```
# Create a new feature with buffered geometry
new_feat = QgsFeature()
new_feat.setGeometry(buffered_geom)
new_feat.setFields(vlayer.fields())

output_sink.addFeature(new_feat, QgsFeatureSink.FastInsert)

return {self.OUTPUT_VECTOR: dest_id}
```

Criando Plugin

Sempre que fizer alguma alteração no código é preciso utilizar o `pb_tools deploy` novamente para enviar a nova versão para o QGIS.

No QGIS, para atualizar o plugin para a nova versão é preciso utilizar o **Plugin Reloader**.