support

Zeszyt ćwiczeń
Programowanie Python w QGIS
(poziom podstawowy)

MINISTERSTWO ŚRODOWISKA

Dofinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

Ćwiczenie 1. Wyświetlenie informacji o wczytanych warstwach

Treść zadania

Wyświetl informacje o wszystkich warstwach wczytanych do QGIS. Należy wydrukować nazwę, id oraz źródło danych każdej warstwy za pomocą polecenia `print()`.

Opis

Listę wczytanych warstw można uzyskać za pomocą metody `mapLayers()` z klasy `QgsProject`. Zwraca ona słownik, którego kluczem jest unikalny identyfikator warstwy, a wartością instancja klasy danego typu warstwy. Następnie należy w pętli `for` zrobić iterację po wszystkich elementach słownika i wydrukować informacje o nazwie (`name()`), źródle (`source()`) oraz id (`id()` lub klucz słownika) każdej warstwy.

Kod źródłowy

```
#Pobranie wszystkich wczytanych warstw
#Dane są zwracane w formie słownika
warstwy = QgsProject.instance().mapLayers()
#Iteracja po elementach słownika, klucz i wartość są rozpisane na
osobne zmienne
for warstwa_id, warstwa in warstwy.items():
    #ID warstwy
    print( warstwa_id )
    #Nazwa
    print( warstwa.name())
    #Źródło danych
    print( warstwa.source() )
```

Ćwiczenie 2. Obliczenie statystyk kanału rastra

Treść zadania

Wykorzystując warstwę `dem92` oblicz średnią wysokość Polski. Wynik wydrukuj za pomocą polecenia `print()`.

Opis

W pierwszej kolejności należy pobrać warstwę rastrową wykorzystując metodę `mapLayersByName()` klasy `QgsProject`. Warstwy w QGIS mogą mieć takie same nazwy, dlatego metoda ta zwraca listę wszystkich warstw o podanej nazwie. W naszym

przypadku nazwy się nie powtarzają, więc metoda ta zwraca listę jednoelementową. Mając warstwę (`QgsRasterLayer`) można również pobrać klasę jej sterownika (`QgsRasterDataProvider`). Z poziomu sterownika należy wywołać metodę `bandStatistics()` podając jako argument numer kanału. Raster `dem92` posiada jeden kanał więc podajemy liczbę `1`. Jako wynik otrzymujemy instancję klasy `QgsRasterBandStats`, która przechowuje wszystkie obliczone statystyki wskazanego kanału. W celu wydrukowania wartości średniej należy wywołać atrybut `mean`. Prawidłowy wynik to 173 m.n.p.m.

Kod źródłowy

```
#Wyszukanie warstwy po nazwie
raster = QgsProject.instance().mapLayersByName( 'dem92' )[0]
#Sterownik warstwy rastrowej (QgsRasterDataProvider)
sterownik = raster.dataProvider()
#Obliczenie statystyk kanału pierwszego
statystyki = sterownik.bandStatistics( 1 )
#mean - wartość średnia wszystkich komórek rastra
print( statystyki.mean )
```

Ćwiczenie 3. Wydrukowanie wszystkich wartości atrybutów obiektów warstwy wektorowej

Treść zadania

Wyeksportuj wartości atrybutów wszystkich obiektów warstwy `wojewodztwa` do pliku *wojewodztwa.csv*. Każdy wiersz powinien reprezentować pojedynczy obiekt przestrzenny, wartości kolejnych elementów należy rozdzielić średnikiem.

Opis

Aby uzyskać dostęp do wszystkich obiektów danej warstwy wektorowej należy wykonać pętlę for na iteratorze zwracanym przez jej metodę `getFeatures()`. Przy każdej iteracji uzyskujemy instancję klasy `QgsFeature` reprezentującej dany obiekt przestrzenny. Aby otrzymać wartości wszystkich atrybutów należy wywołać metodę `attributes()`, a następnie złączyć w jeden tekst za pomocą metody `join()` łańcuchów znaków. Należy jednak pamiętać, że `join()` wymaga aby wszystkie atrybuty były tekstem więc należy je przekonwertować na ten typ wykorzystując np. pomocniczą listę. Po otrzymaniu tekstu można go zapisać do pliku.

Kod źródłowy

```
#Wyszukanie warstwy po nazwie
wektor = QgsProject.instance().mapLayersByName( 'wojewodztwa' )[0]
```

```
with open( r'C:\Users\ppociask\Desktop\wojewodztwa.csv', 'w' ) as
f:
    #Iteracja po obiektach warstwy, obiekt - instancja klasy
QgsFeature
    for obiekt in wektor.getFeatures():
        #Metoda join () wymaga listy tekstów
        #więc trzeba skonwertować wszystkie elementy
        wiersz = []
        for wartosc in obiekt.attributes():
            wiersz.append( str(wartosc) )
        #Złączenie wartości w jeden tekst
        tekst = ';'.join( wiersz ) + '\n'
        #Zapis do pliku
        f.write( tekst )
```

Ćwiczenie 4. Wydrukowanie wartości atrybutu wybranych obiektów warstwy wektorowej

Treść zadania

Wydrukuj kod, nazwę oraz powierzchnię (w km^2) wszystkich mezoregionów należących do makroregionu *Nizina Środkowomazowiecka*. Dane o podziale fizjograficznym Polski znajdują się na warstwie `kondracki92`.

Opis

Metoda `getFeatures()` może przyjąć dodatkowy argument określający filtr danych. Na jego podstawie QGIS zwróci w pętli jedynie te obiekty przestrzenne, które spełniają podane kryteria. W zadaniu należy ograniczyć obiekty na podstawie wartości atrybutów zatem należy podać tekst określający wyrażenie filtra. W warstwie `kondracki92` informacje o makroregionach przechowywane są w kolumnie `MAKROREGIO`, tak więc należy sprawdzić czy wartość z tego pola dla danego obiektu jest równa *Nizina Środkowomazowiecka:* `"MAKROREGIO" = 'Nizina Środkowomazowiecka'`.
Wyrażenie można przetestować w *Kreatorze wyrażeń* np. poprzez narzędzie *Zaznacza obiekty wyrażeniem*. Treść wyrażenia należy podać jako argument metody `getFeatures()`. Należy pamiętać, że jeśli w Pythonowym tekście użyto tego samego znaku co do określenia łańcucha znaków `str` należy poprzedzić go znakiem ucieczki `\`.
Aby wyciągnąć wartość konkretnego atrybutu należy podać nazwę lub indeks pola w kwadratowych nawiasach po obiekcie przestrzennym. Jest to taka sama składnia jak w przypadku Pythonowych kolekcji gdy chcemy wyciągnąć z nich element po indeksie lub kluczu.
Do obliczenie geometrii potrzebna jest instancja klasy `QgsGeometry`, którą można uzyskać wywołując na obiekcie przestrzennym metodę `geometry()`. Klasa geometrii posiada

metody zwracające informacje o niej, dla powierzchni jest to `area()`. Wartość zwracana jest w jednostkach układu współrzędnych danej warstwy. W przypadku warstwy `kondracki92` jest to PUWG 1992 i powierzchnia zwracana jest w m^2, więc otrzymany wynik należy podzielić przez 1 000 000 aby uzyskać wartość w km^2.

Kod źródłowy

```
#Wyszukanie warstwy po nazwie
wektor = QgsProject.instance().mapLayersByName( 'kondracki92' )[0]
#W metodzie getFeatures podajemy wyrażenie
#Ponieważ zawiera ono zarówno cudzysłów jak i apostrof znak,
#który jest używany do określenia tekstu należy poprzedzić \
for obiekt in wektor.getFeatures( " \"MAKROREGIO\" = 'Nizina
Środkowomazowiecka' " ):
    #Wartość atrybutu po nazwie kolumny
    kod = obiekt['KOD']
    #Wartość atrybutu po indeksie kolumny
    nazwa = obiekt[2]
    #Oblicznie powierzchni obiektu
    powierzchnia = obiekt.geometry().area()/1000000
    #Wydrukowanie wymaganych atrybutów
    print( kod, nazwa, powierzchnia )
```

Ćwiczenie 5. Odczyt wartości rastra w punkcie niezależnie od układu współrzędnych

Treść zadania

Wykorzystując warstwę `dem84` odczytaj wysokość terenu dla współrzędnych X: 635990, Y:485275 w układzie PUWG 1992. Wynik wydrukuj za pomocą polecenia `print()`.

Opis

Do wyszukania wartości rastra w punkcie potrzebne są dwa obiekty:

1. Instancja klasy `QgsRasterLayer` reprezentująca analizowaną warstwę
2. Instancja klasy `QgsPointXY` reprezentująca analizowany punkt

Pierwszy obiekt można uzyskać poprzez wyszukanie warstwy po nazwie wykorzystując metodę `mapLayersByName()` klasy `QgsProject`. Warstwy w QGIS mogą mieć takie same nazwy, dlatego metoda ta zwraca listę wszystkich warstw o podanej nazwie. W naszym przypadku nazwy się nie powtarzają, więc metoda ta zwraca listę jednoelementową. Mając warstwę (`QgsRasterLayer`) można również pobrać klasę jej sterownika (`QgsRasterDataProvider`).
Następnie należy utworzyć punkt z wykorzystaniem podanych współrzędnych. Współrzędne są podane w układzie PUWG 1992, natomiast raster jest w układzie WGS84. Należy zatem

dokonać transformacji punktu do układu rastra. W tym celu należy stworzyć instancję klasy `QgsCoordinateReferenceSystem` dla układu PUWG 1992. Mając ten układ współrzędnych oraz układ rastra należy utworzyć instancję klasy `QgsCoordinateTransform` odpowiedzialną za transformację współrzędnych pomiędzy wybranymi układami. W tym kroku należy podać jako źródłowy układ PUWG 1992 (układ, w którym podane są współrzędne punktu), a jako docelowy układ rastra (do tego układu dokonamy konwersji współrzędnych). Jako trzeci argument zawsze należy podawać instancję klasy `QgsProject`.
Następnie wywołujemy metodę `identify()` sterownika rastra podając jako argumenty punkt oraz format danych (dla Pythona zawsze podajemy `QgsRaster.IdentifyFormatValue`). Aby uzyskać dane w łatwej do operowania formie należy na wyniku wywołać metodę `results()`, która zwróci Pythonowy słownik. Ma on tyle elementów ile kanałów raster źródłowy. Kluczami w słowniku są numery kanałów, a wartością odczytane w danym punkcie wartości komórek rastra. Warstwa `dem84` ma jeden kanał więc aby uzyskać jego wartość należy podać klucz `1`. Prawidłowy wynik to 106 m.n.p.m.

Kod źródłowy

```
#Wyszukanie warstwy po nazwie
#Raster może mieć dowolny układ współrzędnych
raster = QgsProject.instance().mapLayersByName( 'dem84' )[0]
#Sterownik warstwy rastrowej (QgsRasterDataProvider)
sterownik = raster.dataProvider()
#Punk zainteresowania
punkt92 = QgsPointXY(635990, 485275)
#Stworzenie definicji układu PUWG 1992
puwg = QgsCoordinateReferenceSystem( 'EPSG:2180' )
#Obiekt do transformacji między układem PUWG 1992 i układem rastra
ct = QgsCoordinateTransform( puwg, raster.crs(),
QgsProject.instance() )
punkt84 = ct.transform( punkt92 )
#Odpytanie sterownika o wartość rastra w danym punkcie
wynik = sterownik.identify( punkt84, QgsRaster.IdentifyFormatValue
)
#Wyciągnięcie informacji w formie słownika
wartosci = wynik.results()
#Wydrukowanie wartości rastra dla kanału pierwszego
print( wartosci[1] )
```

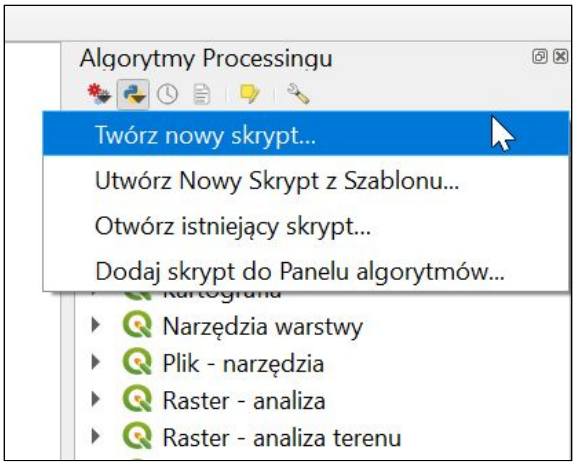
Ćwiczenie 6. Stworzenie szablonu skryptu Narzędzi geoprocessingu - kopiowanie obiektów do nowej warstwy

Treść zadania

Stwórz algorytm do Narzędzi geoprocessingu kopiujący obiekty z warstwy źródłowej do warstwy docelowej. Wszystkie atrybuty i geometrie mają zostać skopiowane.

Opis

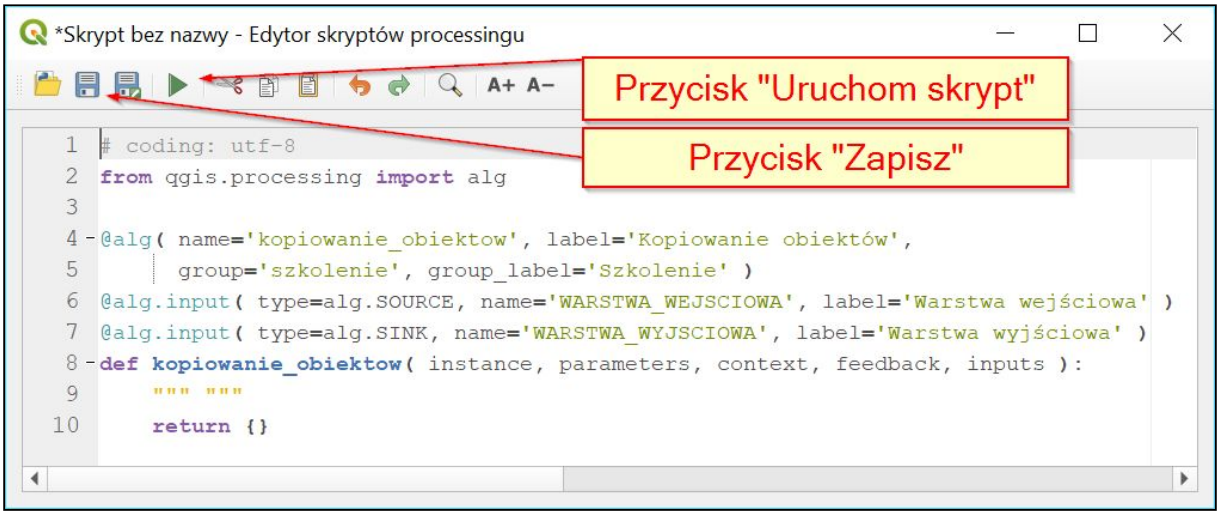
Aby utworzyć nowy skrypt należy w panelu Narzędzi geoprocessingu kliknąć przycisk *Skrypty* i wybrać opcję *Twórz nowy skrypt…* Otworzy się specjalny edytor, w którym można pisać kod źródłowy algorytmu i testować go w trakcie kolejnych etapów pracy.



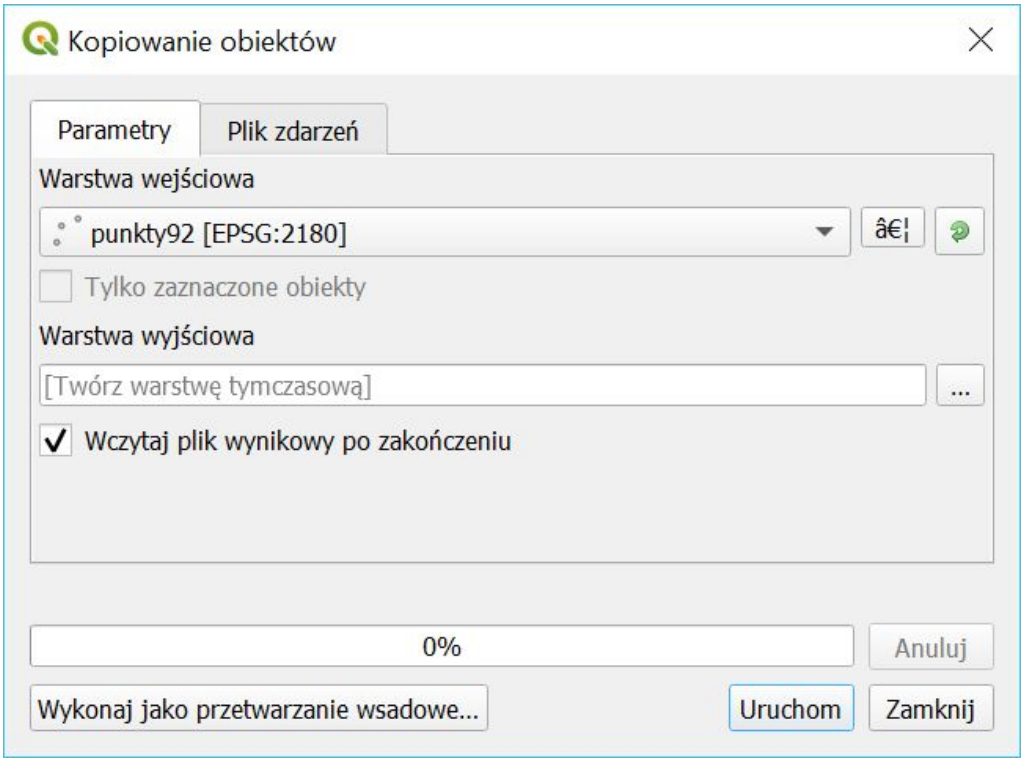
Przy pisaniu nowego skryptu w pierwszej kolejności należy ustawić odpowiednie metadane algorytmu z wykorzystaniem dekoratora `@alg`. Aby z niego skorzystać należy go zaimportować z modułu qgis.processing. Następnie tworzymy definicję funkcji `kopiowanie_obiektow`, która będzie wykonywała operacje na danych przekazanych przez użytkownika. Przyjmuje ona zawsze 5 argumentów: `instance`, `parameters`, `context`, `feedback` i `inputs`. Kolejnym krokiem jest dodanie do niej dekoratorów z wykorzystaniem obiektu `alg`. Pierwszy dekorator `@alg` służy do opisania algorytmu, podajemy w nim jego nazwy oraz nazwy grupy, w której ma się wyświetlić w paneli algorytmów. Rejestruje on również nasz algorytm w QGIS. Dodatkowo tuż pod definicją funkcji należy wpisać dokumentację danego algorytmu w formie komentarza wielolinijkowego. Musi to być w pierwszej linijce funkcji. Sam komentarz może być pusty. Na końcu funkcja musi zwracać słownik, w którym znajdują się wartości parametrów wyjściowych - na początku może to być pusty słownik.
Kolejne dekoratory (`@alg.input` i `@alg.output`) dodajemy dla każdego parametru algorytmu osobno. Kolejność dodawania będzie odzwierciedlona w kolejności odpowiednich kontrolek w oknie dialogowym algorytmu. Definiujemy w nich typ i nazwy danego parametru

oraz ewentualnie dodatkowe ustawienia zależne od jego rodzaju (wartość domyślna, filtry warstw itp). Każdy algorytm musi mieć przynajmniej jeden parametr wyjściowy.
W przypadku kopiowania warstw potrzebne są dwa parametry:

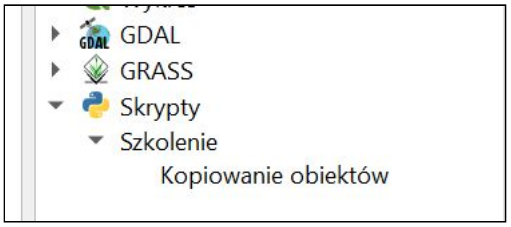
- Warstwa wejściowa - warstwa, z której będą pobierane obiekty przestrzenne, typ parametru `SOURCE`,
- Warstwa wyjściowa - warstwa do zapisu obiektów przestrzennych, typ parametru `SINK`



Można przetestować jego działanie klikając przycisk *Uruchom skrypt*. Jeśli nie ma błędów pojawi się okno algorytmu ze zdefiniowanymi polami dla poszczególnych parametrów.



Można teraz zapisać skrypt do pliku. Po kliknięciu przycisku *Zapisz* pojawi się okno dialogowe otworzone w katalogu, w którym znajdują się wszystkie tworzone skrypty Narzędzi geoprocessingu. Plik należy zapisać w tym właśnie folderze z rozszerzeniem .py np. *kopiowanie_obiektow.py*. Po tej operacji skrypt powinien pojawić się w panelu Narzędzi geoprocessingu w grupie *Skrypty*.



Kolejnym etapem jest stworzenie kodu wewnątrz utworzonej funkcji. W pierwszej kolejności należy pobrać wartości parametrów ustawione przez użytkownika. Służy do tego obiekt `instance` przekazany przez QGISa jako pierwszy argument funkcji. Jest to instancja klasy `QgsProcessingAlgorithm`, która zawiera metody pozwalające na uzyskanie tych informacji. Wszystkie te metody zaczynają się od `parameterAs…`, z dopiskiem określającym typ parametru. Zawsze jako trzy pierwsze argumenty tych metod podajemy:

1. obiekt `parameters`,
2. nazwę systemową parametru (`name`),
3. obiekt `context`.

W większości przypadków wystarczy podać te 3 argumenty, jednak w kilku wyjątkach może być wymagane podanie dodatkowych informacji. Tak jest w przypadku metody `parameterAsSink` dla wektorowej warstwy wyjściowej, która wymaga podania dodatkowo:

1. schematu tabeli atrybutów (`QgsFields`),
2. typu geometrii (`QgsWkbTypes`),
3. układu współrzędnych (`QgsCoordinateReferenceSystem`).

W związku z tym, że warstwa wyjściowa jest kopią wejściowej wszystkie te elementy należy wyciągnąć z kopiowanej warstwy. Dodatkową różnicą metody `parameterAsSink` w stosunku do innych pokrewnych funkcji jest zwracanie przez nią dwuelementowej tupli, w której pierwszym elementem jest specjalny obiekt reprezentujący warstwę wyjściową, a drugi ścieżkę do tworzonego pliku. Drugi element powinien być zwrócony jako wynik działania algorytmu. W tym celu należy do pustego wcześniej słownika dodać element dla tego parametru, jako klucz podajemy nazwę systemową, a jako wartość ścieżkę do pliku.

```
#Warstwa wejściowa
source = instance.parameterAsSource(
     parameters, 'WARSTWA_WEJSCIOWA', context )
#Warstwa wyjściowa
sink, destination = instance.parameterAsSink(
     parameters, 'WARSTWA_WYJSCIOWA', context,
     source.fields(), source.wkbType(), source.sourceCrs() )

return { 'WARSTWA_WYJSCIOWA':destination }
```

Można teraz przetestować działanie algorytmu. Uwaga: przed testowaniem zawsze należy zapisać plik tak, aby w przypadku zawieszenie się QGIS nie stracić wprowadzonych zmian. Po uruchomieniu algorytmu można kliknąć przycisk *Uruchom*. Jeśli nie wystąpił żaden błąd w QGIS powinna pojawić się nowa warstwa o tych samych cechach (układ, geometria, tabela) jak wybrana warstwa wejściowa. Na razie jest ona pusta ponieważ nie dodaliśmy jeszcze do niej żadnego obiektu.

Po wyciągnięciu wartości parametrów można przystąpić do pisania właściwej części algorytmu. Operowanie na obiektach przestrzennych odbywać się będzie w pętli for na warstwie wejściowej dzięki metodzie `getFeatures()`. Przy każdej iteracji otrzymamy dostęp do pojedynczego obiektu przestrzennego - instancji klasy `QgsFeature`. Przy pętli warto dodać obsługę paska postępu. Do tego potrzebujemy informacji o tym ile jest obiektów na warstwie wejściowej oraz który aktualnie obiekt przetwarzamy (numer iteracji). Pierwszą wartość można uzyskać za pomocą metody `featureCount()`, drugą wykorzystując funkcję `enumerate` w definicji pętli. Mając obie liczby należy podzielić numer iteracji przez liczbę obiektów i pomnożyć przez 100, dzięki czemu otrzymamy procentowy postęp iteracji po obiektach. Aby ustawić pasek postępu należy wykorzystać metodę `setProgress()` obiektu `feedback`. Wymaga ona podania liczby całkowitej.

```
liczbaObiektow = source.featureCount()
for indeks, obiekt in enumerate(source.getFeatures()):
    procent = int( (indeks*100)/liczbaObiektow )
    feedback.setProgress( procent )
```

Kolejnym udogodnieniem dla użytkownika, które warto wprowadzić jest możliwość przerwania działania algorytmu przez użytkownika poprzez przycisk *Anuluj* w oknie dialogowym. Informacja o tym, czy użytkownik nacisnął ten przycisk zwracana jest przez metodę `isCanceled()` obiektu `feedback`. Zwraca ona prawdę jeśli przycisk *Anuluj* został naciśnięty, fałsz w przeciwnym wypadku. W związku z tym, że najdłużej nasz algorytm (jak i większość podobnych algorytmów) będzie działał w pętli warto sprawdzanie tego warunku wprowadzić zaraz po definicji pętli.

```
for indeks, obiekt in enumerate(source.getFeatures()):
   if feedback.isCanceled():
        break
```

Ostatnim krokiem jest utworzenie nowego obiektu przestrzennego, wypełnienie go danymi i zapisanie w warstwie wyjściowej. Tworzą instancję klasy `QgsFeature` należy podać jako argument schemat tabeli atrybutów (`QgsFields`) warstwy, do której trafi dany obiekt przestrzenny. Dzięki temu nie natrafimy na błędy związane z nierozpoznaniem atrybutów przy ustawianiu ich wartości.

Mając istniejący i nowy obiekt należy skopiować geometrie i wartości atrybutów między nimi. Z obiektu iterowane wyciągamy geometrię metodą `geometry()`, a atrybuty `attributes()`. Aby przypisać te informacje do nowego obiektu przekazujemy wyniki odpowiednio do metody `setGeometry()` i `setAttributes()` utworzonego obiektu.

Na zakończenie należy dodać nowy obiekt do warstwy wyjściowej poprzez wywołanie metody `addFeature()` obiektu `sink`. Jako drugi argument warto wpisać `QgsFeatureSink.FastInsert` co przyspieszy zapisywanie danych w warstwie.
W związku z tym, że w tym fragmencie zostały użyte dwie nowe klasy należy dodać je do importów na początku skryptu.

```
from qgis.core import QgsFeature, QgsFeatureSink

...

nowy_obiekt = QgsFeature( source.fields() )
nowy_obiekt.setGeometry( obiekt.geometry() )
nowy_obiekt.setAttributes( obiekt.attributes() )
sink.addFeature( nowy_obiekt, QgsFeatureSink.FastInsert )
```

Tak utworzony skrypt może stanowić szablon dla kolejnych algorytmów operujących głównie na danych wektorowych.

Pełny kod źródłowy

```
# coding: utf-8
#import użytych obiektów
from qgis.processing import alg
from qgis.core import QgsFeature, QgsFeatureSink

#Metadane algorytmu, jego nazwa i grupa, w której się znajdzie w
panelu Narzędzi geoprocessingu
@alg( name='kopiowanie_obiektow', label='Kopiowanie obiektów',
     group='szkolenie', group_label='Szkolenie' )
#Warstwa wejściowa
@alg.input( type=alg.SOURCE, name='WARSTWA_WEJSCIOWA',
     label='Warstwa wejściowa' )
#Warstwa wyjściowa
@alg.input( type=alg.SINK, name='WARSTWA_WYJSCIOWA',
     label='Warstwa wyjściowa' )
#Funkcja główna
def kopiowanie_obiektow( instance, parameters, context, feedback,
     inputs ):
    """ Kopiowanie obiektów z warstwy wektorowej """
    #Wyciągnięcie informacji o wybranej przez użytkownika warstwie
wejściowej
    source = instance.parameterAsSource( parameters,
     'WARSTWA_WEJSCIOWA', context )
```

```
    #Określenie warstwy wyjściowej, metadane kopiowane są z warstwy
wejściowej
    sink, destination = instance.parameterAsSink( parameters,
        'WARSTWA_WYJSCIOWA', context,
        source.fields(),    #Schemat tabeli atrybutów
        source.wkbType(),   #Typ geometrii
        source.sourceCrs() )#Układ współrzędnych
    #Liczba obiektów w warstwie wejściowej, do obliczenia postępu
algorytmu
    liczbaObiektow = source.featureCount()
    #Iteracja po obiektach warstwy wejściowej
    #Dzięki enumerate pobieramy również indeks aktualnego obiektu
do obliczenia postępu algorytmu
    for indeks, obiekt in enumerate(source.getFeatures()):
        if feedback.isCanceled():
            #Jeśli użytkownik klikną Anuluj algorytm pętla jest
przerywana
            break
        #Stworzenie nowego obiektu przestrzennego
        nowy_obiekt = QgsFeature( source.fields() )
        #Skopiowanie geometrii z obiektu wejściowego
        nowy_obiekt.setGeometry( obiekt.geometry() )
        #Skopiowanie wartości artybutów z obiektu wejściowego
        nowy_obiekt.setAttributes( obiekt.attributes() )
        #Dodanie obiektu do warstwy wyjściowej, drugi argument
przyspiesza dodawanie obiektów do warstwy
        sink.addFeature( nowy_obiekt, QgsFeatureSink.FastInsert )
        #Oblicznie postępu algorytmu
        procent = int( (indeks*100)/liczbaObiektow )
        #Ustawienie paska postępu na obliczoną wartość
        feedback.setProgress( procent )
    #Zwrócenie informacji o parametrach wyjściwoych, czyli ścieżki
do stworzonego pliku
    return { 'WARSTWA_WYJSCIOWA':destination }
```

Ćwiczenie 7. Skrypt - bufor

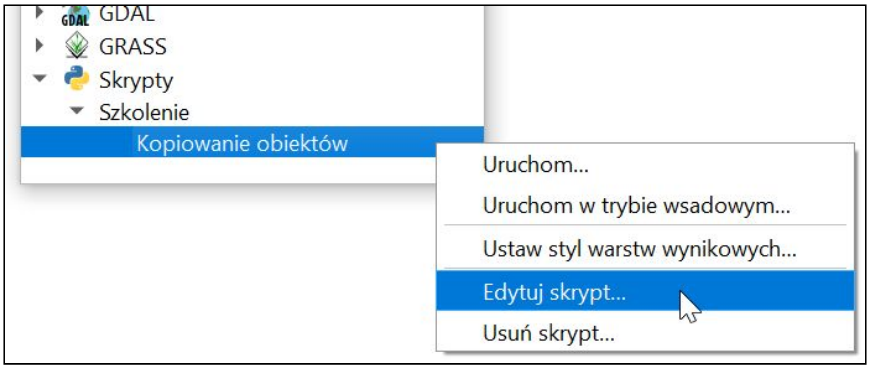
Treść zadania

Napisz algorytm do tworzenia warstwy z buforem geometrii warstwy wejściowej. Użytkownik może określić, poza warstwami wejściową i wyjściową, wielkość bufora (domyślnie 1000).

Warstwami wejściowymi mogą być tylko te z typem geometrii punktowym i liniowym. Tabela atrybutów warstwy wyjściowej ma te same pola oraz nową kolumnę zawierającą powierzchnię obliczonego bufora.

Opis

Do utworzenia nowego algorytmu najwygodniej wykorzystać stworzony wcześniej szablon do kopiowania obiektów między warstwami. W tym celu należy kliknąć na istniejący algorytm i wybrać opcję *Edytuj skrypt…*



Otworzy się okno edytora. W pierwszej kolejności należy zmodyfikować metadane algorytmu przy dekoratorze `@alg` (tylko nazwy algorytmu, nazwy grupy mogą pozostać bez zmian) oraz nazwę funkcji na `bufor`. Następnie należy kliknąć przycisk *Zapisz jako…* i zapisać skrypt pod nazwą bufor.py.

Następnie należy zmodyfikować parametry algorytmu zgodnie z treścią zadania. Warstwami wejściowymi mają być tylko warstwy liniowe i punktowe, więc należy zmodyfikować pierwszy parametr dodając do niego argumenty types. Jako wartość podajemy listę z typami warstw, jakie mogą być wyświetlane. Definicje typów warstw są w klasie `QgsProcessing`, należy podać dwa atrybuty tej klasy: `TypeVectorPoint` i `TypeVectorLine`. Należy również pamiętać o dopisaniu tej klasy do importów.

Nowym parametrem jest wielkość bufora. Do jego wprowadzenia należy użyć jednego z typów parametrów liczbowych `INT` lub `NUMBER`. W związku z tym, że bufor może mieć część dziesiętną lepszym wyborem będzie drugi typ. Do podania wartości domyślnej parametru należy wykorzystać argument `default`.

```
from qgis.core import QgsFeature, QgsFeatureSink, QgsProcessing

@alg( name='bufor', label='Bufor', group='szkolenie',
      group_label='Szkolenie' )
@alg.input( type=alg.SOURCE, name='WARSTWA_WEJSCIOWA',
      label='Warstwa wejściowa',
      types=[QgsProcessing.TypeVectorPoint, QgsProcessing.TypeVectorLine] )
@alg.input( type=alg.NUMBER, name='BUFOR', label='Wielkość bufora',
      default=1000 )
@alg.input( type=alg.SINK, name='WARSTWA_WYJSCIOWA',
      label='Warstwa wyjściowa' )
def bufor( instance, parameters, context, feedback, inputs ):
      …
```

Kolejnym krokiem jest pobranie wartości parametrów ustawionych przez użytkownika. Dla warstwy wejściowej nic się nie zmienia. Nowym parametrem jest wielkość bufora, wartość liczby zmiennoprzecinkowej należy wyciągnąć za pomocą metody `parameterAsDouble()` obiektu `instance`. W tej metodzie wystarczy podać trzy podstawowe argumenty tj. `parameters`, nazwę systemową i `context`.

```
bufor = instance.parameterAsDouble(parameters, 'BUFOR', context)
```

W przypadku warstwy wyjściowej zmienią się dwa elementy. Zgodnie z treścią zadania do tabeli atrybutów ma zostać dodane nowe pole, a typem geometrii zawsze będzie poligon (wynik buforowania). W szablonie schemat tabeli jest kopiowany z warstwy wejściowej. Teraz, jeszcze przed definiowaniem warstwy wyjściowej, należy przygotować nowy schemat tabeli i podać go w metodzie `parameterAsSink()`. Schemat definiujemy za pomocą nowej instancji klasy `QgsFields`. Jeśli chcemy skopiować istniejący schemat przy tworzeniu tej klasy można go podać - w tym momencie oba schematy będą takie same. Następnie do nowego schematu należy dodać nowe pole liczbowe. Pojedyncze pole to instancja klasy `QgsField`, jako argumenty przy jego tworzeniu należy podać nazwę pola oraz typ danych. Typ danych definiowany jest z pomocą klasy `QVariant` (klasa Qt), która ma wiele atrybutów określających różne rodzaje informacji jak teksty, liczby, daty itp. W naszym przypadku pole będzie liczbą zmiennoprzecinkową więc użyjemy typy `QVariant.Double`. Po utworzeniu pola należy je dodać do nowego schematu tabeli za pomocą metody `append` - zostanie ono dodane na koniec tabeli.

```
tabela = QgsFields( source.fields() )
kolumna_powierzchnia = QgsField('Powierzchnia', QVariant.Double)
tabela.append( kolumna_powierzchnia )
```

Mając nowy schemat można go użyć w metodzie `parameterAsSink()`. Drugim elementem, który należy zmodyfikować jest typ geometrii warstwy wyjściowej. Bufor zawsze zwraca obiekt poligonowy więc należy ustawić właśnie taki rodzaj dla warstwy wyjściowej. Służy do tego klasa `QgsWkbTypes`, w które zdefiniowane są wszystkie rodzaje geometrii wspierane przez QGIS (w tym 3D i krzywe). Poligony określone są atrybutem `Polygon` tej klasy. Wszystkie nowo użyte klasy należy dodać do importów.

```
from qgis.PyQt.QtCore import QVariant
from qgis.core import (QgsFeature, QgsFeatureSink, QgsProcessing
     QgsWkbTypes, QgsFields, QgsField)

...

sink, destination = instance.parameterAsSink( parameters,
        'WARSTWA_WYJSCIOWA', context,
        tabela,
        QgsWkbTypes.Polygon,
        source.sourceCrs() )
```

Mając poprawnie zdefiniowaną warstwę wyjściową należy odpowiednio zmodyfikować tworzenie obiektów, które do niej trafią. Przy tworzeniu nowej instancji klasy `QgsFeature` należy teraz podać nowy schemat tabeli atrybutów.

```
nowy_obiekt = QgsFeature( tabela )
```

Geometria nowego obiektu jest tworzona poprzez przekształcenie geometrii obiektu iterowanego. Klasa `QgsGeometry` posiada metody pozwalające np. obliczyć centroid, otoczkę czy bufor wokół geometrii źródłowej. Do tego ostatniego przekształcenia służy metoda `buffer()`, która przyjmuje dwa argumenty: wielkość bufora oraz liczbę określającą liczbę wierzchołków do zaokrąglenia geometrii (domyślnie możemy ustawić na 5). Wielkość bufora należy podać w jednostkach danej warstwy, czyli np. w metrach dla układu PUWG 1992 lub w stopniach dla WGS 84. Nowo utworzoną geometrię można zapisać do osobnej zmiennej ponieważ później posłuży nam do obliczenia pola powierzchni (metoda `area()`).

```
geometria = obiekt.geometry().buffer( bufor, 5 )
nowy_obiekt.setGeometry( geometria )
```

Schematy tabel obu warstw różnią się od siebie, więc nie możemy przekopiować wartości bezpośrednio pomiędzy oboma obiektami. Aby to obejść można wykonać jedną z dwóch metod:

1. Wyciągnięcie listy wartości metodą `attributes()` i dodanie do niej na końcu wartości powierzchni - ten sposób jest możliwy jeśli wiemy dokładnie w jakiej kolejności są pola w nowej tabeli:

```
atrybuty = obiekt.attributes()
atrybuty.append( geometria.area() )
nowy_obiekt.seAttributes( atrybuty )
```

2. Ustawienie wszystkich wartości jedna po drugiej poprzez iterację po nazwach kolumn warstwy. Listę nazw można uzyskać za pomocą metody `names()` klasy `QgsFields`. W takim wypadku iteracja odbywa się po tabeli warstwy wejściowej, wszystkie dodatkowe pola należy ustawić poza pętlą.

```
nowy_obiekt['Powierzchnia'] = geometria.area()
for nazwa in source.fields().names():
    nowy_obiekt[nazwa] = obiekt[nazwa]
```

Pozostałe elementy skryptu się nie zmieniają.

Kod źródłowy

```
# coding: utf-8
#import użytych obiektów
from qgis.processing import alg
from qgis.PyQt.QtCore import QVariant
from qgis.core import (QgsFeature, QgsFeatureSink, QgsWkbTypes,
     QgsFields, QgsField, QgsProcessing)
```

```
#Metadane algorytmu, jego nazwa i grupa, w której się znajdzie w
panelu Narzędzi geoprocessingu
@alg( name='bufor', label='Bufor', group='szkolenie',
      group_label='Szkolenie' )
#Warstwa wejściowa
@alg.input( type=alg.SOURCE, name='WARSTWA_WEJSCIOWA',
      label='Warstwa wejściowa',
      types=[QgsProcessing.TypeVectorPoint ,
             QgsProcessing.TypeVectorLine] )
#Wielkość bufora
@alg.input( type=alg.NUMBER, name='BUFOR', label='Wielkość bufora',
      default=1000 )
#Warstwa wyjściowa
@alg.input( type=alg.SINK, name='WARSTWA_WYJSCIOWA',
      label='Warstwa wyjściowa' )
#Funkcja główna
def bufor( instance, parameters, context, feedback, inputs ):
    """ Kopiowanie obiektów z warstwy wektorowej """
    #Wyciągnięcie informacji o wybranej przez użytkownika warstwie
wejściowej
    source = instance.parameterAsSource( parameters,
'WARSTWA_WEJSCIOWA', context )
    #Wielkość bufora ustawiona przez użytkownika
    bufor = instance.parameterAsDouble(parameters, 'BUFOR',
context)
    #Stworzenie nowego schematu tabeli i wypełnienie go polami z
warstwy źródłowej
    tabela = QgsFields( source.fields() )
    #Stworzenie nowej kolumny
    kolumna_powierzchnia = QgsField('Powierzchnia',
QVariant.Double)
    #Dodanie kolumny do tabeli
    tabela.append( kolumna_powierzchnia )
    #Określenie warstwy wyjściowej, metadane kopiowane są z warstwy
wejściowej
    sink, destination = instance.parameterAsSink( parameters,
        'WARSTWA_WYJSCIOWA', context,
        tabela,                 #Nowy schemat tabeli atrybutów
        QgsWkbTypes.Polygon,    #Typ geometrii
        source.sourceCrs() )    #Układ współrzędnych
    #Liczba obiektów w warstwie wejściowej, do obliczenia postępu
algorytmu
    liczbaObiektow = source.featureCount()
    #Iteracja po obiektach warstwy wejściowej
```

```
    #Dzięki enumerate pobieramy również indeks aktualnego obiektu
do obliczenia postępu algorytmu
    for indeks, obiekt in enumerate(source.getFeatures()):
        if feedback.isCanceled():
            #Jeśli użytkownik wcisną Anuluj algorytm pętla jest
przerywana
            break
        #Stworzenie nowego obiektu przestrzennego
        nowy_obiekt = QgsFeature( tabela )
        #Stworzenie nowej geometrii poprzez obliczenie bufora o
zadanej wielkości
        geometria = obiekt.geometry().buffer( bufor, 5 )
        #Skopiowanie geometrii z obiektu wejściowego
        nowy_obiekt.setGeometry( geometria )
        #Przypisanie wartości powierzchni
        nowy_obiekt['Powierzchnia'] = geometria.area()
        #Skopiowanie wartości atrybutów z obiektu wejściowego
        #Iteracja po nazwach kolumn warstwy wejściowej i kopiujemy
wartości jedna po drugiej
        for nazwa in source.fields().names():
            #Przypisanie wartości kolumny
            nowy_obiekt[nazwa] = obiekt[nazwa]

        #Dodanie obiektu do warstwy wyjściowej
        sink.addFeature( nowy_obiekt, QgsFeatureSink.FastInsert )
        #Obliczanie postępu algorytmu
        procent = int( (indeks*100)/liczbaObiektow )
        #Ustawienie paska postępu na obliczoną wartość
        feedback.setProgress( procent )
    #Zwrócenie informacji o parametrach wyjściowych, czyli ścieżki
do stworzonego pliku
    return { 'WARSTWA_WYJSCIOWA':destination }
```

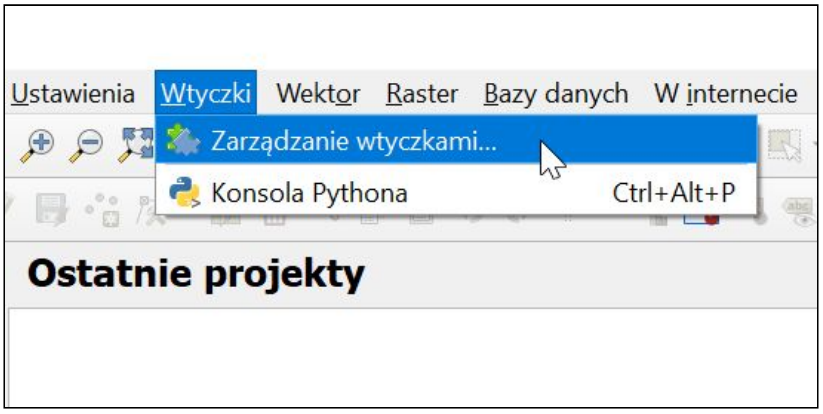
Ćwiczenie 8. Instalacja wtyczek PluginBuilder i Plugin Reloader

Treść zadania

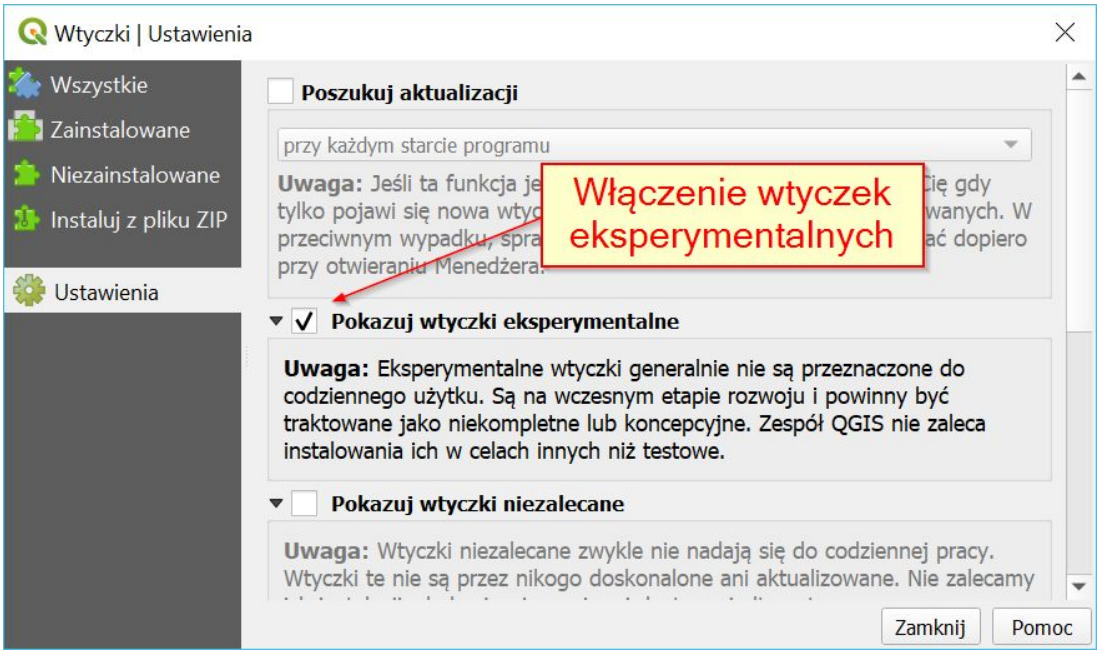
Zainstaluj w QGIS rozszerzenia Plugin Builder oraz Plugin Reloader z oficjalnego repozytorium wtyczek.

Opis

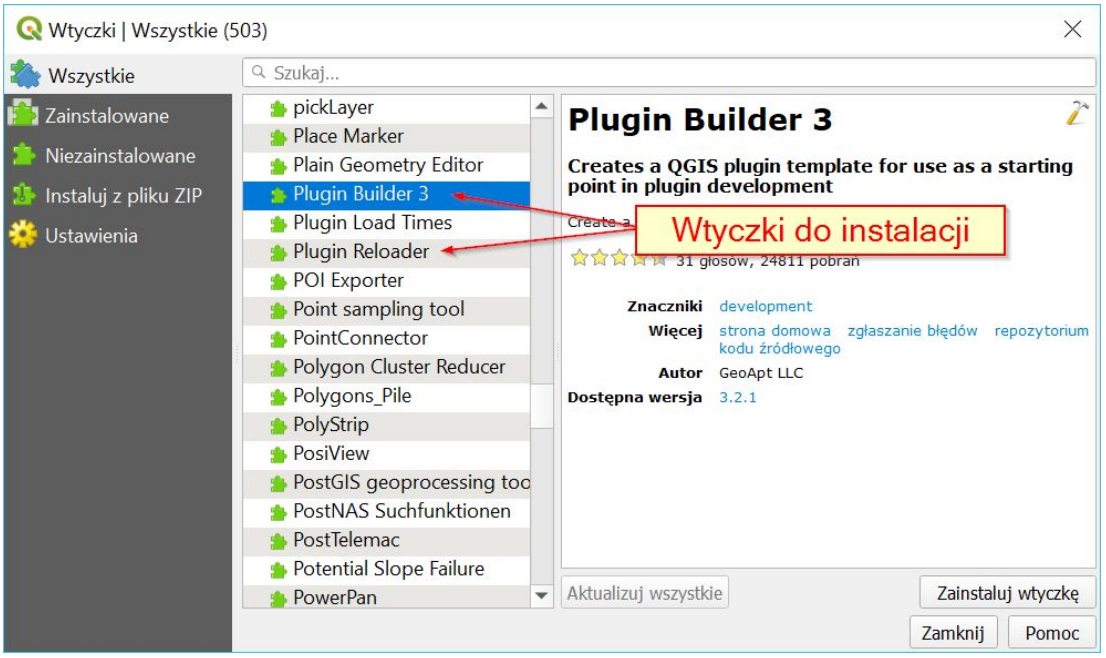
Do instalacji rozszerzeń służy *Menedżer wtyczek*. Można go otworzyć wybierając w menu *Wtyczki* pozycję *Zarządzanie wtyczkami…*



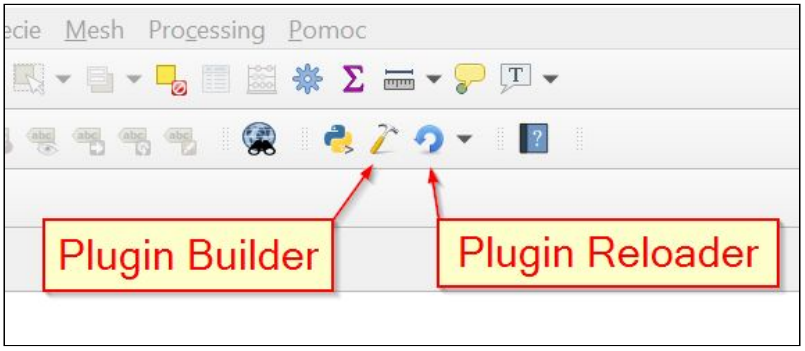
Wtyczka *Plugin Reloader* jest oznaczona jako eksperymentalna w związku z tym nie jest domyślnie widoczna na liście. Aby to zmienić należy przejść na zakładkę *Ustawienia* i zaznaczyć opcję *Pokazuje wtyczki eksperymentalne*.



Następnie można zainstalować obie wtyczki w QGIS zaznaczając je po kolei na liście i klikając przycisk *Zainstaluj wtyczkę*.



Po zakończeniu ikony wtyczek pojawią się na pasku narzędzi *Wtyczki*.



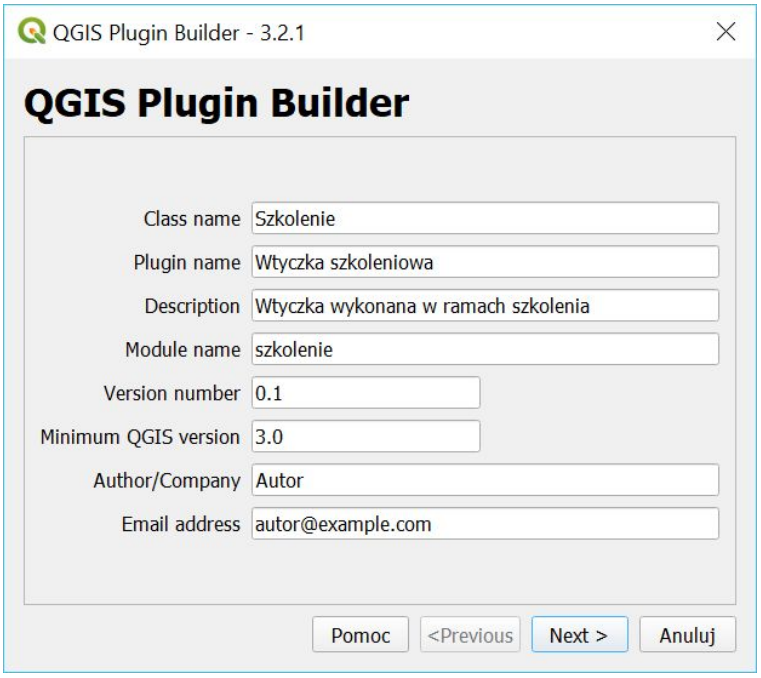
Ćwiczenie 9. Stworzenie szablonu wtyczki z wykorzystaniem wtyczki Plugin Builder

Treść zadania

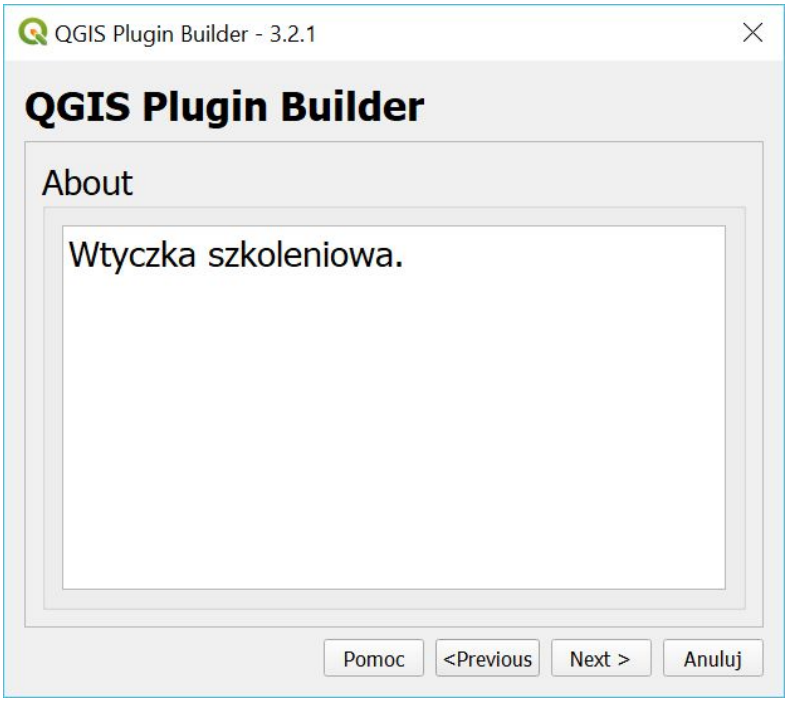
Wykorzystując Plugin builder stwórz szablon prostej wtyczki z oknem dokowanym.

Opis

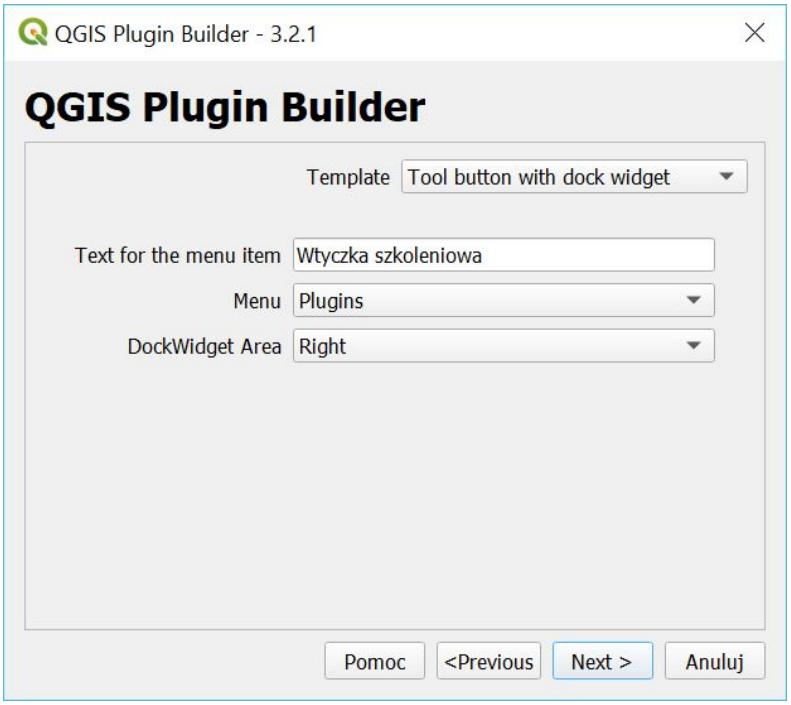
Aby uruchomić Plugin builder należy kliknąć przycisk  na pasku narzędzi lub wybrać menu *Wtyczki* → *Plugin Builder* → *Plugin Builder*. Pojawi się kreator wtyczek, w kolejnych oknach dialogowych można podać podstawowe ustawienia nowo tworzonej wtyczki. W pierwszym oknie ustawiamy podstawowe metadane wtyczki. Po wpisaniu poniższych danych należy kliknąć *Next*.



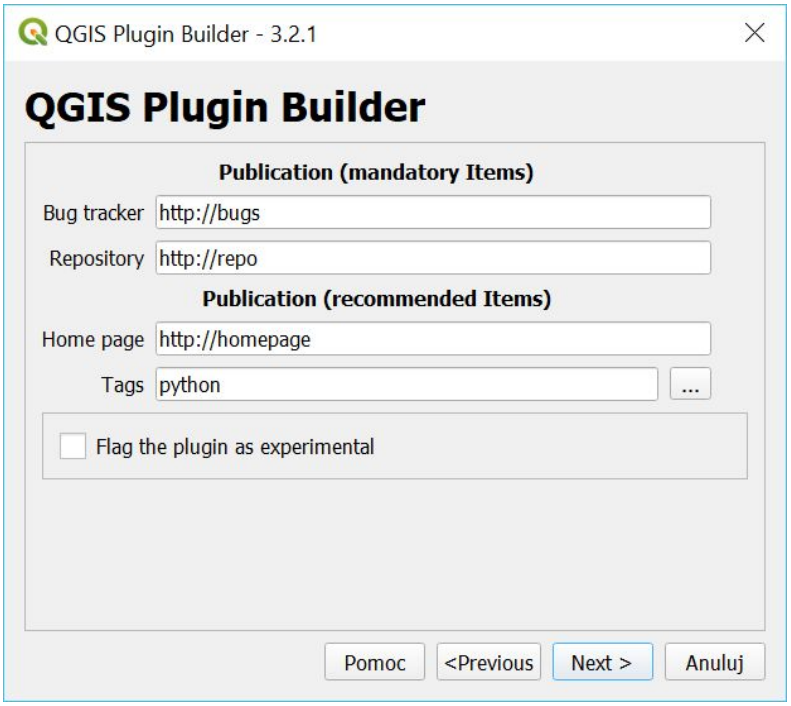
W kolejnym oknie podajemy dłuższy opis wtyczki i klikamy *Next*.



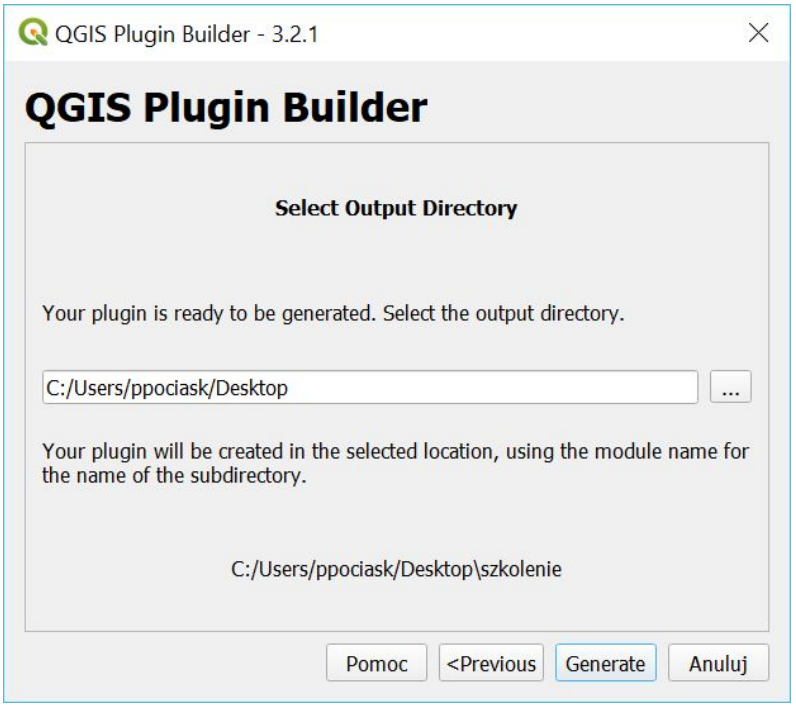
Teraz można wybrać jakie okna będzie miała wtyczka. W treści zadania jest mowa o panelu dokowanym więc należy wybrać szablon *Tool button with dock widget*.



W kolejnym kroku ustawiane są kolejne metadane wtyczki, tym razem związane ze źródłem wtyczki. Dane te są wymagane jeśli wtyczka będzie dodawana do oficjalnego repozytorium QGIS, jeśli nie można pozostawić te pola niezmienione.



Ostatnim krokiem jest wskazanie katalogu gdzie powstanie wtyczka. Po jego ustawieniu należy kliknąć przycisk *Generate*. Kolejne komunikaty, które mogą się pojawić należy zamknąć.



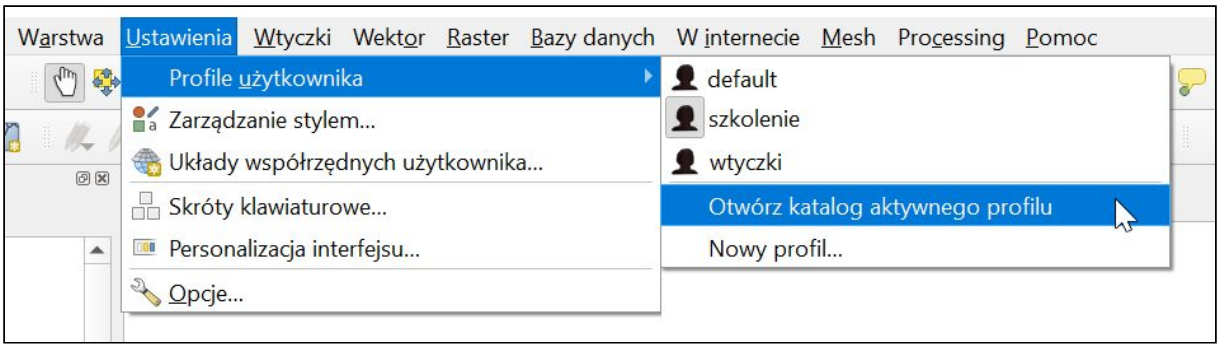
Ćwiczenie 10. Uruchomienie wtyczki w QGIS

Treść zadania

Dodaj i uruchom stworzoną wtyczkę w QGIS.

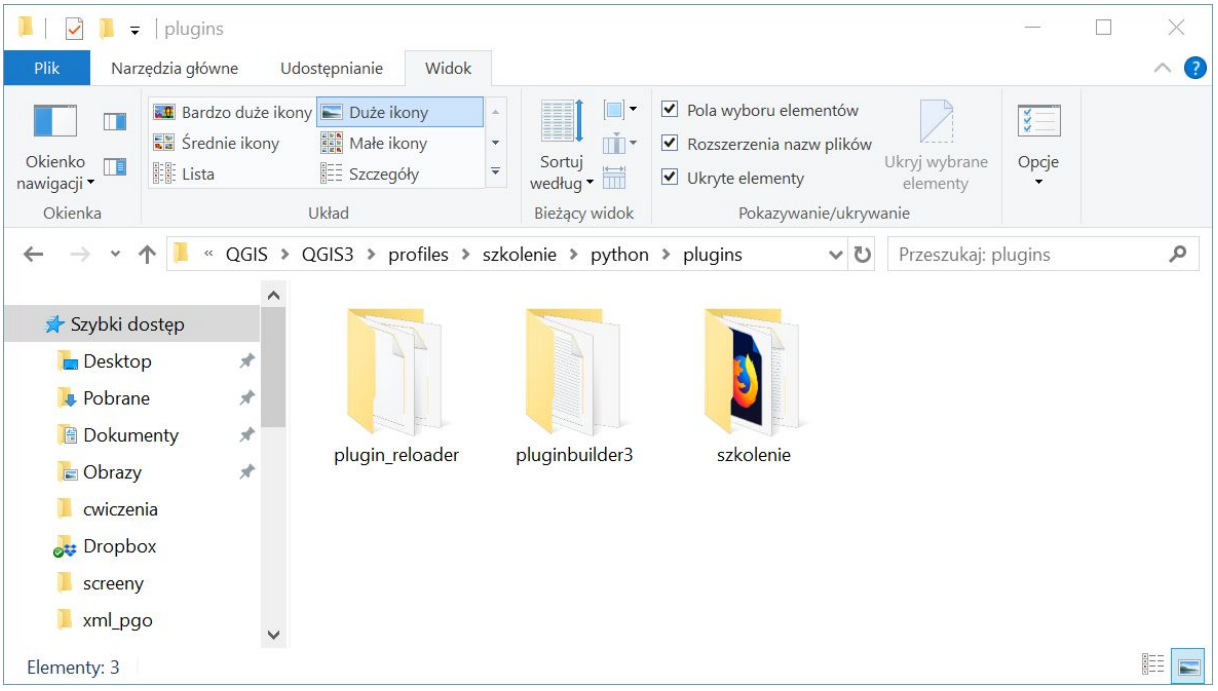
Opis

Stworzoną wtyczkę należy ją skopiować do katalogu QGIS, w którym przechowywane są wszystkie zainstalowane wtyczki. Najprościej się do niego dostać wybierając z menu *Ustawienia → Profile użytkownika* opcję *Otwórz katalog aktywnego profilu*.



W otworzonym katalogu przechowywane są wszystkie ustawienia aktywnego profilu. Wtyczki przechowywane są w podkatalogu *python/plugins* (jeśli folder *plugins* nie istnieje należy go

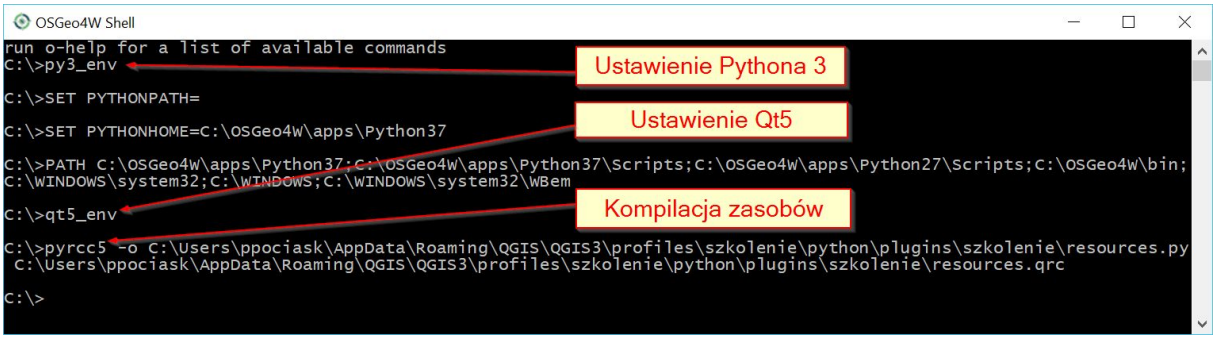
utworzyć). Po wejściu do niego powinny być widoczne wszystkie zainstalowane dotychczas wtyczki. Folder wtyczki utworzonej przez *Plugin Builder* należy przenieść w to miejsce.



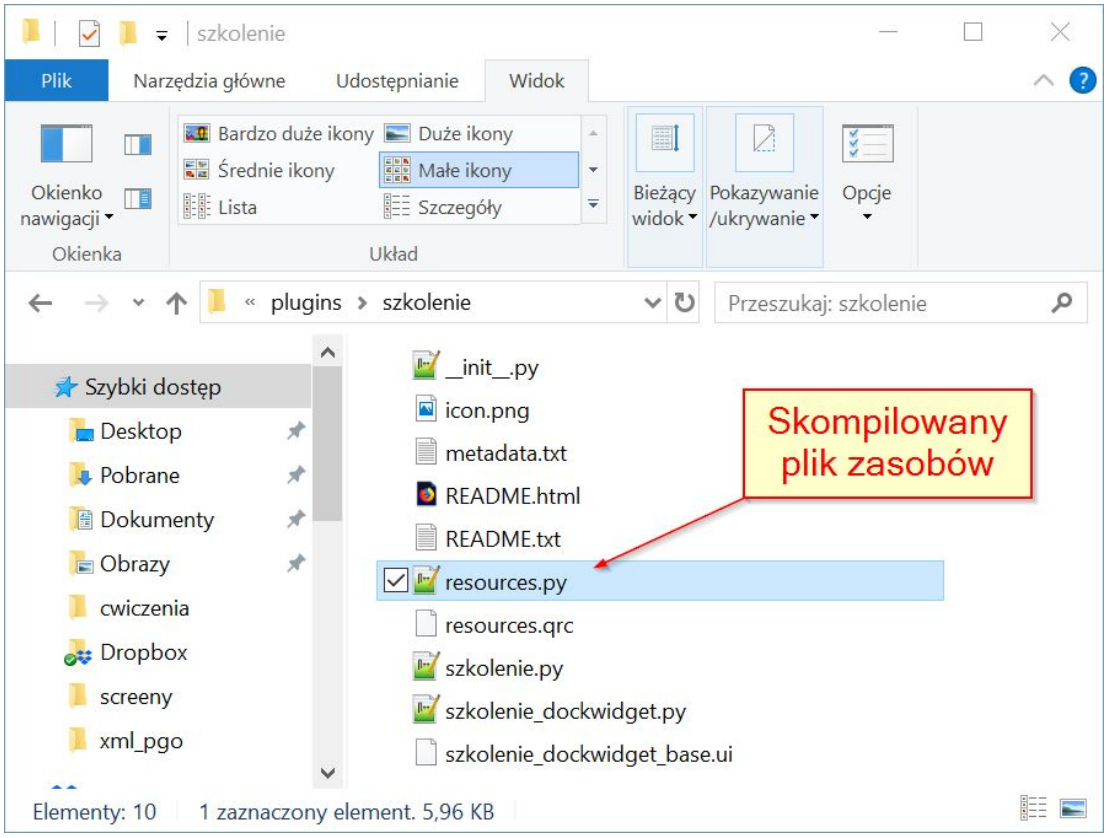
Aby uruchomić wtyczkę w QGIS należy skompilować zasoby wtyczki, czyli plik *resources.qrc*. W tym celu należy uruchomić konsolę *OSgeo4W* i uaktywnić Python 3 i Qt 5 za pomocą poleceń `py3_env` i `qt5_env`. Następnie należy wykonać kompilację pliku .qrc poleceniem:

```
pyrcc5 -o <plik_wynikowy>.py <plik_wejsciowy>.qrc
```

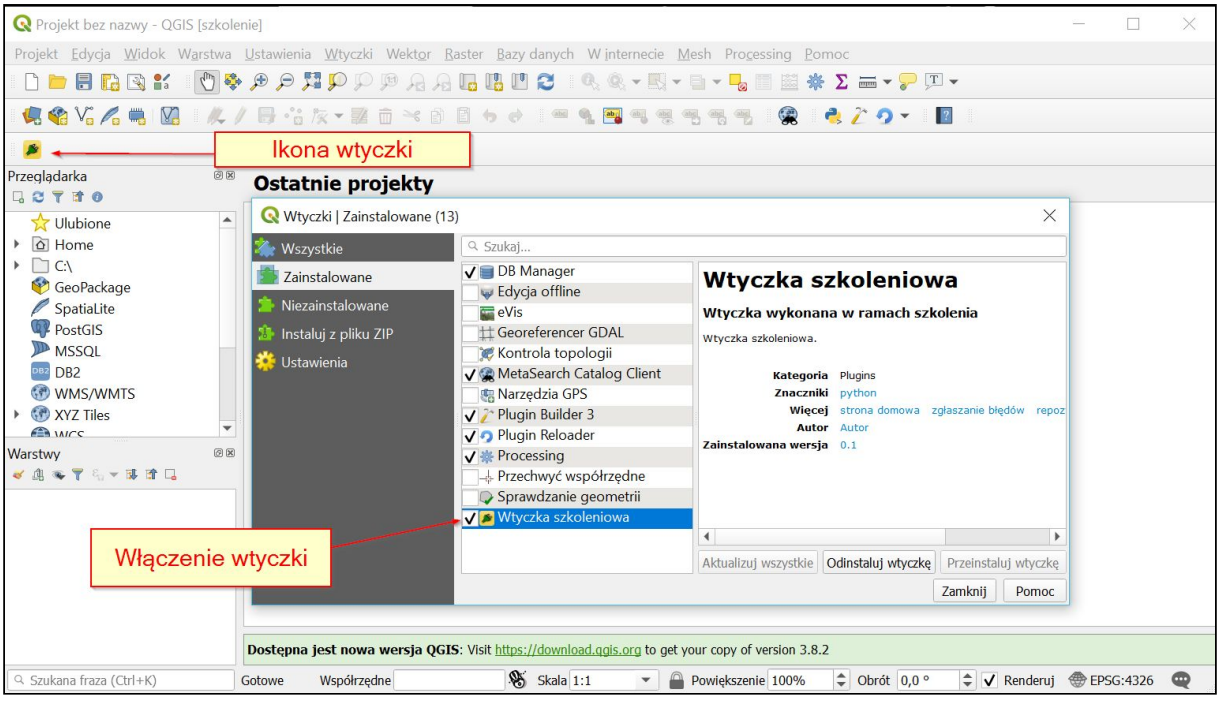
Na konsolę można upuścić plik .qrc co ułatwi wpisanie pełnych ścieżek do plików.



W katalogu wtyczki powinien powstać nowy plik *resources.py*.



Aby uruchomić wtyczkę należy zrestartować aplikację QGIS, wejść do Menedżera wtyczek i z jego poziomu włączyć ją klikając pole obok nazwy na liście. Ikona wtyczki pojawi się w oknie głównym QGIS.



Ćwiczenie 11. Tworzenie interfejsu graficznego w aplikacji Qt Designer

Treść zadania

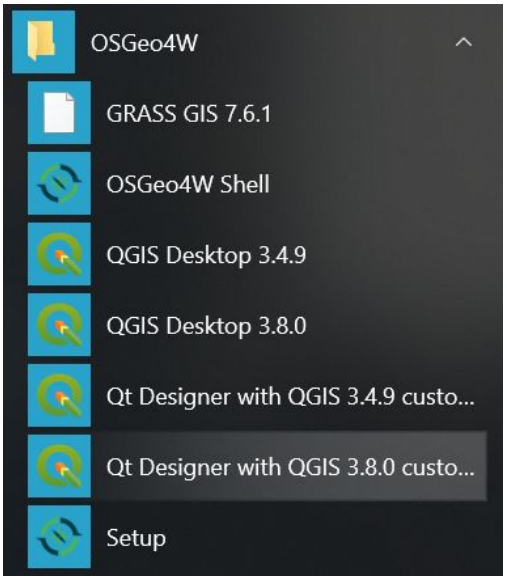
Do okna dokowanego dodaj dwie kontrolki:

- listę warstw wczytanych do QGIS - QgsMapLayerComboBox
- przycisk Eksportuj - QPushButton

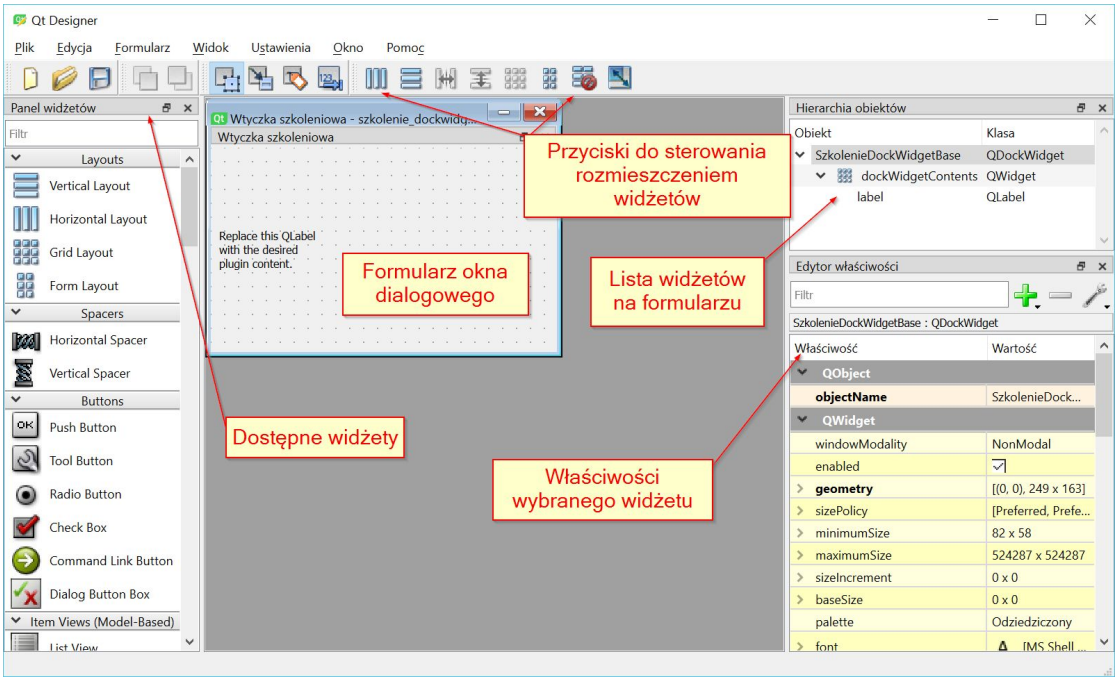
Kontrolki powinny znajdować się w górnej części okna, na górze lista warstw, a pod nią przycisk dociśnięty do prawej strony okna.

Opis

Do edycji okien dialogowych służy aplikacja *Qt Designer*, która jest instalowana razem z QGIS. Jest ona dostępna w menu Start. Jeśli jest dostępnych kilka wersji należy wybrać tą, która jest zgodna z docelową wersją QGIS.

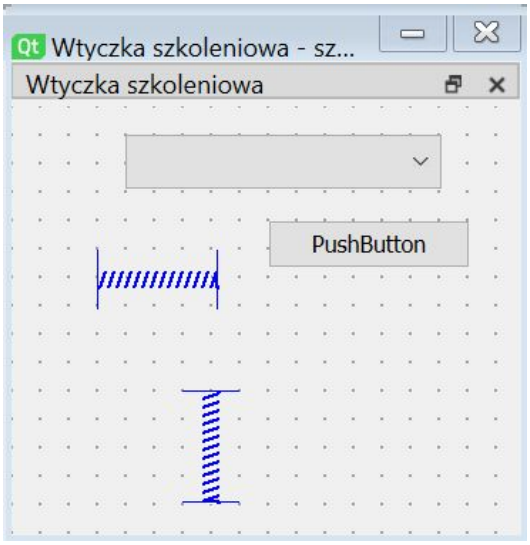


Po jej uruchomieniu pojawi się okno powitalne. Po kliknięciu przycisku Otwórz.. można wskazać plik *szkolenie_dockwidget_base.ui* z katalogu wtyczki. Alternatywnie można zamknąć okno powitalne i przeciągnąć plik na okno aplikacji bezpośrednio z folderu.

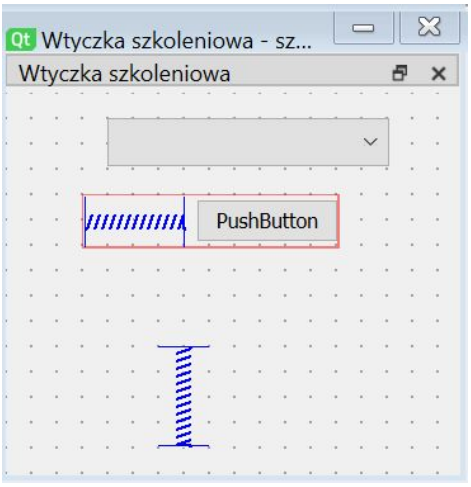


W pierwszej kolejności należy wyczyścić formularz z istniejących elementów. Aby widżety nie były automatycznie rozmieszczane (co może utrudniać na początku pracy układanie elementów) należy wcisnąć przycisk *Usuń rozmieszczenie*  . Następnie należy zaznaczyć etykietę i usunąć ją klawiszem *Del*.

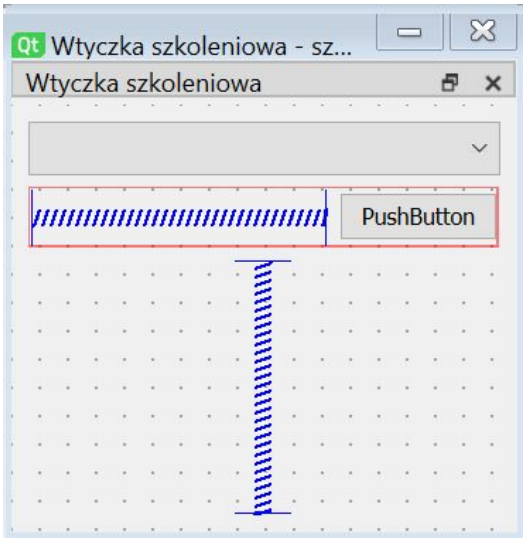
W kolejnym kroku należy dodać nowe kontrolki do formularza. Na liście z lewej strony okna, w grupie *QGIS custom widgets*, znajduje się klasa `QgsMapLayerComboBox`, należy ją złapać kursorem i przesunąć na formularz. W podobny sposób należy dodać przycisk `QPushButton` (grupa *Buttons*) oraz dwa *Spacer'y* - horyzontalny i wertykalny (grupa *Spacers*). Powinny one być rozmieszczone mniej więcej w ten sposób:



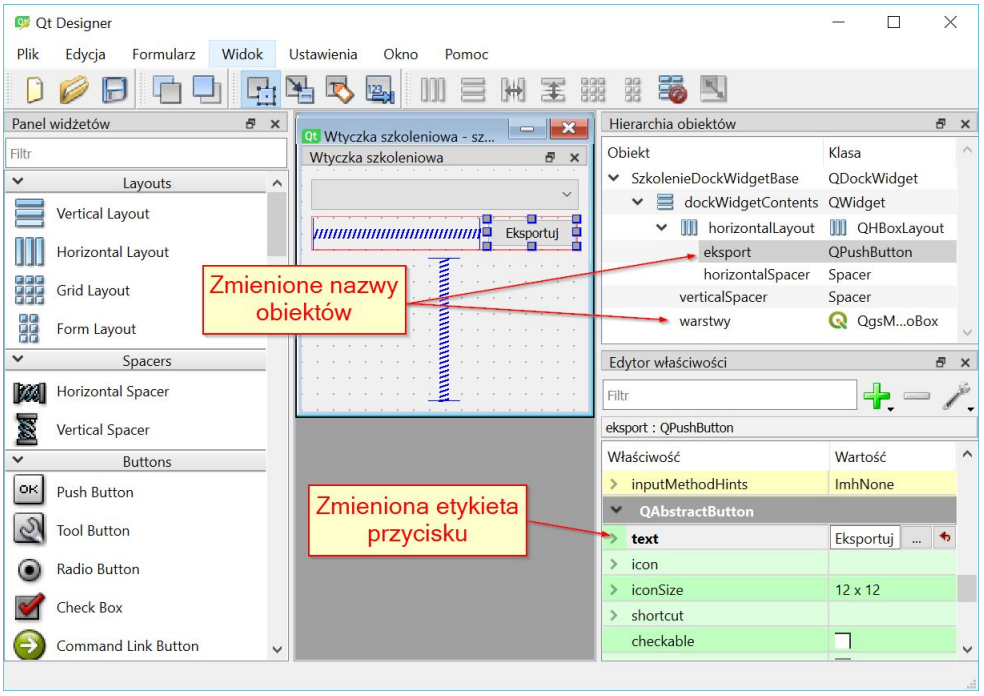
Następnie zaznacz poziomy Spacer oraz przycisk i naciśnij przycisk *Rozmieść w poziomie*  . Dzięki temu oba widżety zostaną zgrupowane i będą traktowane na formularzu jako pojedynczy obiekt.



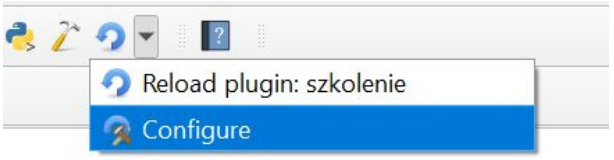
Teraz odznacz wszystkie widżety klikając na czysty formularz i kliknij przycisk *Rozmieść w pionie*  . Wszystkie kontrolki zostały automatycznie rozmieszczone w formularzu jedna pod drugą, będa również dynamicznie reagowały na zmianę rozmiaru okna.



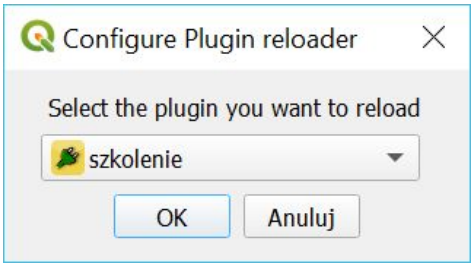
Na koniec należy zmienić napis etykietę przycisku na *Eksportuj* oraz zmienić nazwę jego i listy warstw na wygodniejsze w użyciu w kodzie źródłowym. W tym celu należy zaznaczyć daną kontrolkę i we właściwościach zmienić wartości dla pozycji *text* (wyświetlany tekst przycisku) i *objectName* (nazwa obiektu).



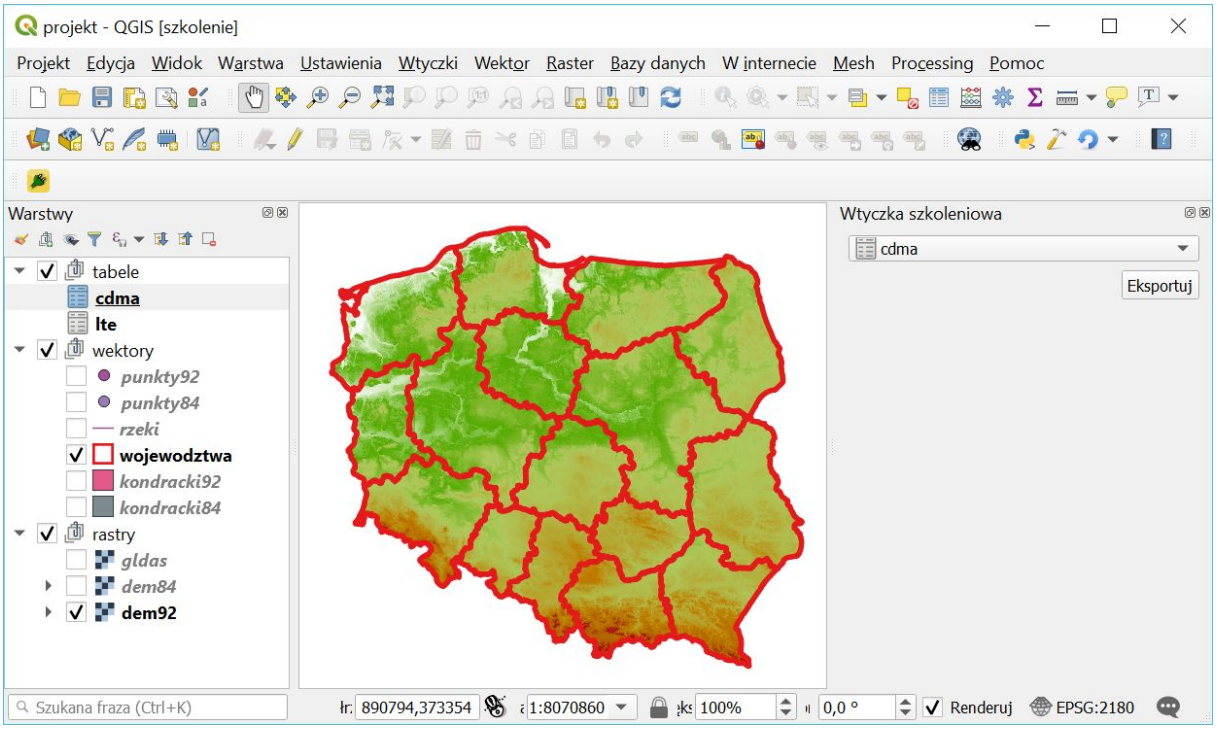
Po zakończeniu należy zapisać zmiany w pliku *szkolenie_dockwidget_base.ui*. Aby odświeżyć wtyczkę w QGIS można zresetować całą aplikację, wyłączyć i włączyć wtyczkę w *Menedżerze wtyczek* albo wykorzystać *Plugin Reloader*. Ostatnia opcja jest najwygodniejsza, przed pierwszym użyciem należy ją jednak skonfigurować. W tym celu należy kliknąć na strzałkę obok przycisku wtyczki i wybrać z menu Configure.



W oknie, które się pojawi należy wskazać wtyczkę, która ma być powiązana z tym narzędziem i kliknąć OK.



Od teraz każdorazowe wciśnięcie przycisku wtyczki Plugin Reloader na pasku narzędzie spowoduje przeładowanie całej wtyczki.



Ćwiczenie 12. Oprogramowanie wtyczki - eksport danych do CSV

Treść zadania

Dodaj funkcjonalności do stworzonego okna dialogowego. Po wybraniu przez użytkownika warstwy wektorowej z listy i naciśnięciu przycisku Eksportuj użytkownik może wskazać zapisania pliku CSV z danymi tej warstwy. Eksportowane mają być atrybuty wszystkich obiektów.

Opis

W celu oprogramowania panelu dokowanego i jego widżetów należy zmodyfikować klasę reprezentującą to okno. W naszym przypadku znajduje się ona w pliku *szkolenie_dockwidget.py* i nosi nazwę `SzkolenieDockWidget`. Nad definicją tej klasy, za pomocą uic następuje kompilacja pliku .ui do obiektu Pythona (`FORM_CLASS`), który następnie jest dziedziczony przez klasę `SzkolenieDockWidget`. W metodzie `__init__()` wywoływana jest funkcja `setupUi` - tworzy ona wszystkie widżety znajdujące się w pliku .ui. Po wywołaniu tej funkcji z poziomu tej klasy mamy dostęp do widżetów za pomocą nazw nadanych w *Qt Designerze*.

W pierwszej kolejności należy ograniczyć listę wyświetlanych warstw tylko do wektorów, ponieważ aktualnie wyświetlane są również rastry. W tym celu należy ustawić odpowiedni filtr danych kontrolki `warstwy`. Ustawienie filtru odbywa się za pomocą metody `setFilters()`, w której należy podać odpowiedni atrybut klasy

`QgsMapLayerProxyModel`. Aby wyświetlić warstwy wektorowe (w tym nieprzestrzenne tabele) należy podać `QgsMapLayerProxyModel.VectorLayer`. Ustawienie filtra powinno odbyć się jak najwcześniej, najlepiej w momencie tworzenia instancji klasy (metoda `__init__()`), ale po utworzeniu wszystkich kontrolek, czyli po wywołaniu metody `setupUi()`. Należy również pamiętać o dodaniu importu każdej nowej klasy używanej w kodzie.

```
from qgis.core import QgsMapLayerProxyModel
...
def __init__(self, parent=None):
    ...
    self.setupUi(self)
    self.warstwy.setFilters( QgsMapLayerProxyModel.VectorLayer )
```

Po przeładowaniu wtyczki można zweryfikować czy zmiany zostały poprawnie wprowadzone.

Kolejnym krokiem jest podpięcie kliknięcia w przycisk z metodą, która zostanie wtedy wywołana. Przyciski mają sygnał `clicked`, który należy powiązać z nową metodą `eksportDanych()`.

```
def __init__(self, parent=None):
    ...
    self.eksport.clicked.connect( self.eksportDanych )

...

def eksportDanych(self):
    pass
```

Zgodnie z treścią zadania użytkownik ma możliwość wybrania miejsca gdzie powstanie nowy plik CSV. Najprostszym rozwiązaniem jest wykorzystanie klasy `QFileDialog` i jej metody `getSaveFileName()`, dzięki której zostanie wywołane systemowe okno dialogowe, za pomocą którego można wskazać nowy plik do utworzenia. Przyjmuje ona jako argumenty następujące elementy:

- widżet główny - można ustawić `None`
- tekst na belce okna
- katalog startowy - można wpisać pusty tekst
- filtr widocznych plików

Wynikiem jest dwuelementowa tupla zawierająca ścieżkę do wskazanego pliku oraz filtr, jaki był wybrany przez użytkownika. Warto również zabezpieczyć się przed zamknięciem przez użytkownika okna bez wskazywania pliku. W takim wypadku obie zmienne będą pustymi tekstami.

```
def eksportDanych(self):
    #Otworzenie okna do wskazania pliku
    plik, filtr = QtWidgets.QFileDialog.getSaveFileName(
            None, 'Zapisz w CSV', '', 'Pliki CSV (*.csv)' )
    if not plik:
        #Użytkownik nie wskazał żadnego pliku
```

```
        return
```

Mając informacje o pliku można przygotować informacje do zapisu. Do tego potrzebna jest warstwa wybrana przez użytkownika na liście. Można ją uzyskać wywołując metodę `currentLayer()`, która zwróci instancję klasy `QgsVectorLayer`. Mając tą klasę możemy iterować po obiektach i zbadać jej schemat tabeli.

```
warstwa = self.warstwy.currentLayer()
```

Następnie należy otworzyć plik do zapisu za pomocą polecenia open() podając ścieżkę do pliku wynikowego. Plik będzie miał następującą strukturę:

- każdy wiersz jest pojedynczym obiektem z warstwy wejściowej
- separatorem kolumn będzie średnik

```
with open( plik, 'w' ) as f:
```

W pierwszej linii pliku należy umieścić nazwy kolumn. W tym celu można wykorzystać metodę names() kasy `QgsFields`. Mając listę nazw można ją złączyć w pojedynczy tekst za pomocą metody `join()`. Na końcu dodatkowo trzeba wstawić znak nowej linii `\n` aby kolejny tekst zapisywany do pliku był w nowej linii. Mając wiersz w formie tekstu można go zapisać do pliku.

```
tekst = ';'.join( kolumna.name() ) + '\n'
f.write( tekst )
```

Ostatnim krokiem jest iteracja po wszystkich obiektach warstwy i dodawanie ich wartości do pliku. Odbywać będzie się to na podobnej zasadzie jak nagłówki kolumn, wartości atrybutów obiektów można wyciągnąć metodą attributes() klasy QgsFeature i złączyć je za pomocą join(). Należy jednak pamiętać, że join() wymaga aby wszystkie atrybuty były tekstem więc należy je przekonwertować na ten typ.

```
for obiekt in warstwa.getFeatures():
    wiersz = []
    for wartosc in obiekt.attributes():
        wiersz.append( str(wartosc) )
    tekst = ';'.join( wiersz ) + '\n'
    f.write( tekst )
```

Po zakończeniu operacji warto wyświetlić użytkownikowi informację za pomocą klasy `QgsMessageBar`. Do tego wymagany jest import obiektu `iface`.

```
from qgis.utils import iface

...

iface.messageBar().pushSuccess( 'Eksport', 'Poprawnie
wyeksportowano
     dane do pliku {}'.format( plik ) )
```

Pełny kod źródłowy

```
import os

from qgis.PyQt import QtGui, QtWidgets, uic
from qgis.PyQt.QtCore import pyqtSignal
from qgis.core import QgsMapLayerProxyModel, NULL
from qgis.utils import iface

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


class SzkolenieDockWidget(QtWidgets.QDockWidget, FORM_CLASS):

    closingPlugin = pyqtSignal()

    def __init__(self, parent=None):
        """Constructor."""
        super(SzkolenieDockWidget, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots -
see
        # http://doc.qt.io/qt-5/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.setupUi(self)
        self.warstwy.setFilters( QgsMapLayerProxyModel.VectorLayer
)
        self.eksport.clicked.connect( self.eksportDanych )

    def closeEvent(self, event):
        self.closingPlugin.emit()
        event.accept()

    def eksportDanych(self):
        #Otworzenie okna do wskazania pliku
        plik, filtr = QtWidgets.QFileDialog.getSaveFileName(
                None, 'Zapisz w CSV', '', 'Pliki CSV (*.csv)' )
        if not plik:
            #Użytkownik nie wskazał żadnego pliku
            return
        #Warstwa wybrana przez użytkownika
        warstwa = self.warstwy.currentLayer()
```

```
        #Otworzenie pliku w trybie do zapisu
        with open( plik, 'w' ) as f:
            #Połączenie nazw kolumn w jeden tekst
            tekst = ';'.join( warstwa.fields().names() ) + '\n'
            #Zapis tekstu do pliku
            f.write( tekst )
            #Iteracja po obiektach warstwy
            for obiekt in warstwa.getFeatures():
                #Metoda join () wymaga listy tekstów więc trzeba
skonwertować wszystkie elementy
                wiersz = []
                for wartosc in obiekt.attributes():
                    wiersz.append( str(wartosc) )
                #Złączenie wartości w jeden tekst
                tekst = ';'.join( wiersz ) + '\n'
                #Zapis do pliku
                f.write( tekst )
        #Wyświetlenie komunikatu o sukcesie
        iface.messageBar().pushSuccess( 'Eksport', 'Poprawnie
wyeksportowano dane do pliku {}'.format( plik ) )
```