MINISTERSTWO ŚRODOWISKA

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

TRAPPA

Szkolenie
Programowanie Python w QGIS

Prowadzący

Michał Bednarczyk

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

Python - wprowadzenie

http://docs.python.org/2/tutorial/

Python

- Python – język programowania wysokiego poziomu ogólnego przeznaczenia rozbudowanym pakiecie bibliotek standardowych, którego ideą przewodnią jest czytelność i klarowność kodu źródłowego. Jego składnia cechuje się przejrzystością i zwięzłością
- Python wspiera różne paradygmaty programowania: obiektowy, imperatywny oraz w mniejszym stopniu funkcyjny.
- Podobnie jak inne języki dynamiczne jest często używany jako język skryptowy. Interpretery Pythona są dostępne na wiele systemów operacyjnych.

Krótka historia

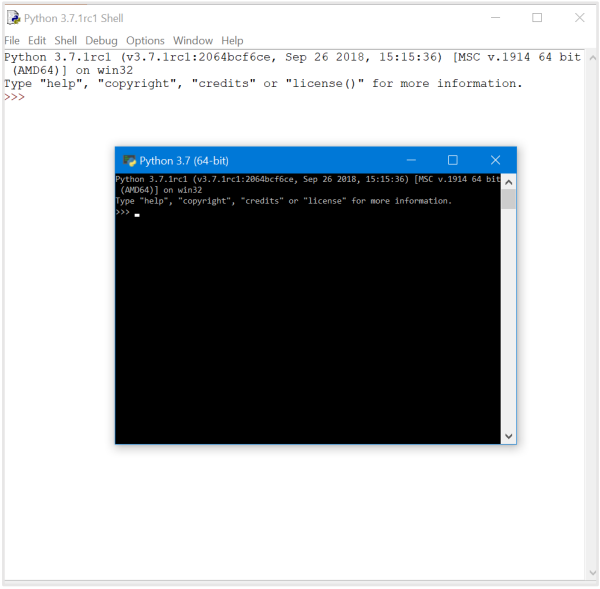
- Pythona stworzył we wczesnych latach 90. Guido van Rossum – jako następcę języka ABC, stworzonego w Centrum Matematyki i Informatyki w Amsterdamie. Van Rossum jest głównym twórcą Pythona, choć spory wkład w jego rozwój pochodzi od innych osób.
- Python rozwijany jest jako projekt Open Source zarządzany przez Python Software Foundation, która jest organizacją non-profit.
- Nazwa języka pochodzi od serialu "Monty Python's Flying Circus"

Skąd wziąć Python'a?

https://www.python.org/downloads/

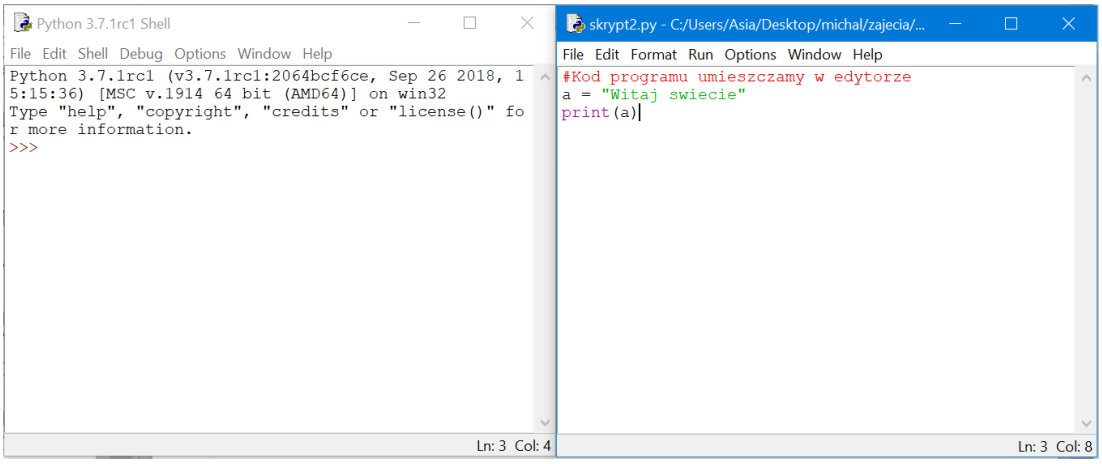
Należy wybrać wersję zgodną z systemem operacyjnym

Po zainstalowaniu dostępna będzie konsola oraz proste IDE (IDLE)



Rozpoczęcie pracy z Python IDLE

Z menu **File** należy wybrać **New**, pojawi się edytor kodu. Można z jego pomocą edytować i uruchomić program. Wyniki będą widoczne w konsoli.



Python - podstawy

Składnia Python'a

Komentarze

```
# To jest pierwszy komentarz
SPAM = 1          # a to jest drugi komentarz
                  # ... a teraz trzeci!
STRING = "# A to już nie jest komentarz."
```

Liczby i operatory

Standardowe użycie operatorów: +,-,/,*

```
>>> 2+2
4
```

Przypisanie wartości do zmiennej:

```
>>> width = 20
```

Możliwe jest przypisanie wielokrotne:

```
>>> x = y = z = 0  # Zero x, y and z
```

Liczby zespolone – funkcja `complex(real,imag)`:

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Funkcje konwertujące

```
>>> float(3)
3.0
>>> int(3.99)
3
>>> long(3.99)
3L
```

Podstawowe operatory

Operatory arytmetyczne

a=10
b=20

Operator	Opis	Przykład
+	Dodawanie - dodaje wartości znajdujące się po obu stronach	a+b da wynik 30
-	Odejmowanie - odejmuje wartości znajdujące się po obu stronach	a-b da wynik -10
*	Mnożenie - mnoży wartości z obu stron operatora	a*b da wynik 200
/	Dzielenie - wartość z lewej strony operatora przez wartość z prawej	b/a da wynik 2
%	modulo - dzieli wartość z lewej strony przez wartość z prawej i zwraca licznik części ułamowej	b%a da wynik 0
**	Potęgowanie - podstawą jest lewa strona, wykładnikiem prawa	a**2 da wynik 100
//	dzielenie bez reszty (floor division) - zwraca wynik dzielenia po opuszczeniu części ułamkowej	9//2 da wynik 4

Operatory porównania

a=10
b=20

Operator	Opis	Przykład
==	Równe - sprawdza, czy wyrażenie z lewej stwony jest równe wyrażeniu z prawej. Jeżeli tak, zwraca TRUE, w przeciwnym wypadku FALSE	(a==b) jest FALSE
!=	Różne od - sprawdza, czy wyrażenie z lewej stwony jest równe wyrażeniu z prawej. Jeżeli nie, zwraca TRUE, w przeciwnym wypadku FALSE	(a!=b) jest TRUE
<>	Różne od - sprawdza, czy wyrażenie z lewej strony jest równe wyrażeniu z prawej. Jeżeli nie, zwraca TRUE, w przeciwnym wypadku FALSE (podobnie jak operator !=)	(a<>b) jest TRUE
>	Większe od - sprawdza, czy wyrażenie z lewej strony jest większe od wyrażenia z prawej strony, jeżeli tak zwraca TRUE, w przeciwnym wypadku FALSE	(a>b) jest FALSE
<	Mniejsze od - sprawdza, czy wyrażenie z lewej strony jest mniejsze od wyrażenia z prawej strony, jeżeli tak zwraca TRUE, w przeciwnym wypadku FALSE	(a<b) jest TRUE
>=	Większe lub równe - sprawdza, czy wyrażenie z lewej strony jest większe od lub równe wyrażeniu z prawej strony, jeżeli tak zwraca TRUE, w przeciwnym wypadku FALSE	(a>=b) jest FALSE
<=	Mniejsze lub równe - sprawdza, czy wyrażenie z lewej strony jest mniejsze od lub równe wyrażeniu z prawej strony, jeżeli tak zwraca TRUE, w przeciwnym wypadku FALSE	(a<=b) jest TRUE

Operatory przypisnia

Operator	Opis	Przykład
=	Proste przypisanie - przypisuje wartość wyrażenia z lewej strony, wyrażeniu z prawej	c=a+b sprawi, że c przyjmie wartość sumy a i b
+=	Dodanie i przypisanie - wartość wyrażenia z lewej strony zostanie dodana do wartości wyrażenia z prawej, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c+=a jest równoznaczne z c=c+a
-=	Różnica i przypisanie - od wartości wyrażenia z lewej strony zostanie odjęta wartość wyrażenia z prawej, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c-=a jest równoznaczne z c=c-a
*=	Mnożenie i przypisanie - wartość wyrażenia z lewej strony zostanie pomnożona przez wartość wyrażenia z prawej, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c*=a jest równoznaczne z c=c*a
/=	Dzielenie i przypisanie - wartość wyrażenia z lewej strony zostanie podzielona przez wartość wyrażenia z prawej, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c/=a jest równoznaczne z c=c/a
%=	Modulo i przypisanie - zostanie obliczona wartość modulo lewej i prawej strony, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c%=a jest równoznaczne z c=c%a
**=	Potęga i przypisanie - wartość wyrażenia z lewej strony zostanie podniesiona do potęgi o wykładniku równym wartości wyrażenia z prawej, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c**=a jest równoznaczne z c=c**a
//=	Dzielenie bez reszty i przypisanie - wartość wyrażenia z lewej strony zostanie podzielona bez reszty przez wartość wyrażenia z prawej, następnie wynik zostanie przypisany do wyrażenia z lewej strony.	c//=a jest równoznaczne z c=c//a

Operatory logiczne

a=True
b=True

Operator	Opis	Przykład
and	Operator logiczny AND (koniunkcja). Zwraca TRUE, jeżeli lewa i prawa strona ma wartość TRUE.	(a and b) jest TRUE
or	Operator logiczny OR (alternatywa). Zwraca TRUE, jeżeli lewa lub prawa strona ma wartość TRUE.	(a or b) jest TRUE
not	Operator logiczny NOT (zaprzeczenie). Używany do odwrócenia wartości logicznej wyrażenia.	not(a and b) jest FALSE

Typy danych i zmienne

- Python należy do tak zwanych języków programowania typowanych dynamicznie, oznacza to, że nie ma potrzeby określania typu zmiennej. Zostanie ona ustalona podczas wykonywania kodu, w zależności od kontekstu.

```
a=12
a=”abc”
a,b=1,”abc”
```

Ciągi tekstowe - strings

```
>>> 'STRING w apostrofach'
'STRING w apostrofach'
>>> "STRING w cudzyslowie"
'STRING w cudzyslowie'
```

Dzielenie linii:

```
>>> hello = "To jest całkiem długi ciąg tekstowy zawierający\n\
... kilka linii tekstu, po to by je rozdzielić.\n\
...     Zauważ, że odstęp w tej linii (whitespace) nie jest\
...  pomijany."
>>> print hello
To jest całkiem długi ciąg tekstowy zawierający
kilka linii tekstu, po to by je rozdzielić.
    Zauważ, że odstęp w tej linii (whitespace) nie jest
pomijany.
```

Konkatenacja i formatowanie ciągów tekstowych

- Konkatenacja ciągów (operator +)

```
“tekst nr 1 ” + “tekst nr 2”
```

Pola reprezentuące inne rodzaje wartości
%s - tekst
%d - liczba całkowita

- Formatowanie ciągów (operator % oraz pole %f)

```
>>>"Wynik obliczenia to: %f" % 23.678
'Wynik obliczenia to: 23.678000'
>>>"Wynik obliczenia to: %.f" % 23.678
'Wynik obliczenia to: 24'
>>>"Wynik obliczenia to: %.2f" % 23.678
'Wynik obliczenia to: 23.68'
```

Formatowanie z użyciem kilku pól

Jeeli mamy zmienne:

```
K = 23
Z = 4
>>>"%d jest wieksze od %d" % (K,Z)
'23 jest wieksze od 4'
```

Listy

Lista w Python'nie nie wymaga określenia typu danych:

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

Odwołanie do wartości listy:

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:3]
['eggs', 100]
>>> a[1:-1]
['eggs', 100]
```

Łączenie i generowanie list:

```
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam',
'eggs', 100, 'Boo!']
```

Tablice

Utworzenie tablicy

```
A=[
['Jan',80,75,85,90,95],
['Julia',75,80,75,85,100],
['Krystyna',80,80,80,90,95]
]
```

Pobranie wartości elementu

```
>>>A[0][0]
'Jan'
```

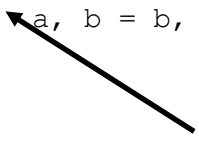
Zapis wartości do elementu

```
A[0][1] = 156
```

Sterowanie wykonaniem

Pętla while

```
>>> # Ciąg Fibonacciego:
... # suma poprzedzających elementów definiuje
następny
... a, b = 0, 1
>>> while b < 10:
...   print b
...   a, b = b, a+b
...
1
1
2
3
5
8
>>>
```



UWAGA: w języku Python należy robić wcięcia (indent) dla każdego podrzędnego bloku kodu. W przeciwnym wypadku nastąpi błąd składni.

Pętla while

```
>>> a, b = 0, 1
>>> while b < 10:
...   print b,
...   a, b = b, a+b
...
1 1 2 3 5 8
```

Aby wynik wyświetlił się w jednej linii należy użyć przecinka po zmiennej wskazanej na wyjściu.

Instrukcja warunkowa if

```
>>> x = int(raw_input("Podaj liczbe calkowita: "))
Podaj liczbe calkowita: 42
>>> if x < 0:
...     x = 0
...     print 'Ujemna, zmieniono na 0'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Jeden'
... else:
...     print 'Wiecej'
...
Wiecej
```

Pętla for

```
>>> # Zmierz ciągi znaków:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print w, len(w)
...
cat 3
window 6
defenestrate 12
```

Funkcja `len()` zwraca ilość elementów listy.

Funkcja range()

Generowanie zakresu od zera:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Generowanie dowolnego przedziału:

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
```

Generowanie przedziału z krokiem:

```
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Range() i for

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Break, continue i else w pętlach (1)

Szukanie liczb pierwszych

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

BREAK – przerwie wykonanie jeżeli warunek (reszta z dzielenia) jest spełniony
ELSE – UWAGA – dotyczy pętli FOR, a nie if. Zadziała, gdy nie nastąpi BREAK.

Break, continue i else w pętlach (2)

Szukanie liczb parzystych:

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print "Znaleziono liczbe parzysta", num
...         continue
...     print "Zanleziono liczbe", num
Znaleziono liczbe parzysta 2
Zanleziono liczbe 3
Znaleziono liczbe parzysta 4
Zanleziono liczbe 5
Znaleziono liczbe parzysta 6
Zanleziono liczbe 7
Znaleziono liczbe parzysta 8
Zanleziono liczbe 9
```

CONTINUE – powoduje wykonanie kolejnej iteracji bez przechodzenia dalej

Obsługa wyjątków (2)

```
try:
  <kod do wykonania>
except:
  <kod do wykonania gdy wystąpi wyjątek>
```

Obsługa wyjątków (1)

```
a = "abc"
try:
   int(a)
   print a
except ValueError:
        print "Zly format"
```

Obsługa wyjątków (3)

```
a = "abc"
try:
   int(a)
   print a
except ValueError,x:
        print "Wystapil blad: ",x
else:
    print "ok"
```

Próba wykonania kodu

Wykonanie gdy wyjątek

Wykonanie gdy nie nastąpi wyjątek

Funkcje i klasy obiektów

Deklaracja funkcji

```
Def [nazwa_funkcji](<parametry wejściowe>):
     <kod funkcji>
    …...
    Return <zwracana wartość>
```

Przykład 1

```
>>> def suma(a,b):
...   print a+b
...
>>> suma(12,34)
46
```

Przykład 2

```
>>> def suma(a,b):
...   return a+b
...
>>> a=suma(43,34)
>>> a+7
84
```

Klasy obiektów

```
>>> class Klasa:
...     i=10
...     def daji(self):
...       return self.i
...     def daj10i(self):
...       return self.i*10
...
>>> k=Klasa()
>>> k.i
10
>>> k.daji()
10
>>> k.daj10i()
100
```

Biblioteki i moduły w python - wykorzystanie

Wbudowaną funkcjonalność języka Python można rozszerzyć korzystając z możliwości importu modułów zewnętrznych

```
import math
import numpy
import numpy as n
```

Usunięcie importu:

```
del math
```

Import wybranej funkcji lub metody

```
from datetime import timedelta
```

Przykładowe biblioteki

NumPy - rozszerza funkcjonalnośc matematyczną Pythona

Scipy - zawiera algorytmy funkcji matematycznych i naukowych

matplotlib - biblioteka do rysowania wykresów

SQLAlchemy - funkcje do wykonywania zapytań SQL

BeautifulSoup - parser języka XML i HTML

Biblioteki i moduły w python
uzyskiwanie informacji

wyświetlanie wszystkich zaimportowanych zasobów:

```
dir()
```

wyświetlanie elementów danego zasobu (np. math):

```
dir(math)
```

pomoc na temat biblioteki lub funkcji:

```
help(scipy)
help(scipy.zeros)
```

Więcej informacji o programowaniu w Python'nie:

http://docs.python.org/2/tutorial/

https://wiki.python.org

http://www.tutorialspoint.com/python/

Python w QGIS

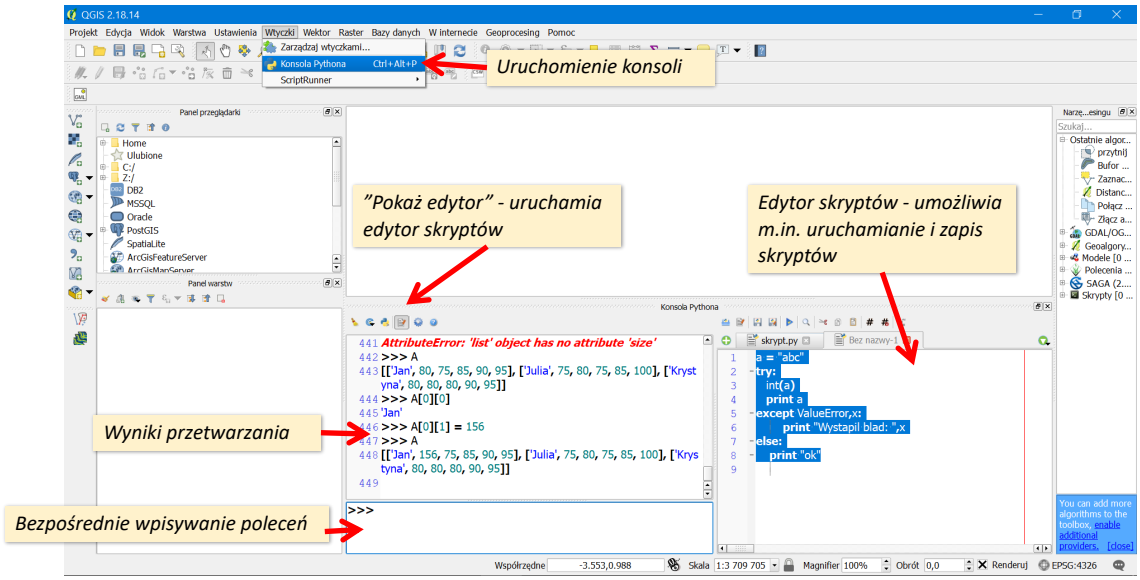
https://docs.qgis.org/2.18/en/docs/pyqgis_developer_cookbook/

Zastosowanie Python w QGIS

Język Python jest wykorzystywany w systemie QGIS w wielu miejscach:

- Konsola Python'a
- Akcje
- Skrypty geoprocessingu
- Wtyczki

Obsługa konsoli Python w QGIS



Biblioteki QGIS

- Po uruchomieniu konsoli, użytkownik ma dostęp do instancji klasy QgisInterface (poprzez polecenie iface), dzięki któremu można operować w środowisku Quantum GIS. Pozwala ona m.in. na dostęp do wczytanych warstw, obszaru mapy czy ustawień aplikacji.

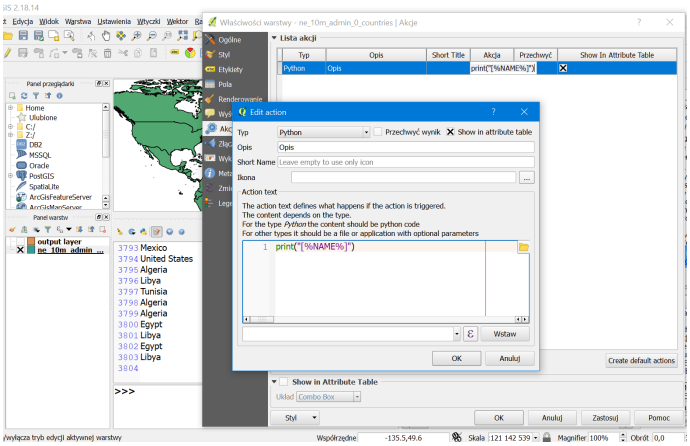
```
>>>iface
<qgis._gui.QgisInterface object at 0x00000000095E96A8>
```

- Importując moduły qgis.core i qgis.gui użytkownik ma pełny dostęp do API Quantum GIS.

```
>>>from qgis import core, gui
```

Akcje - omówienie - przykład

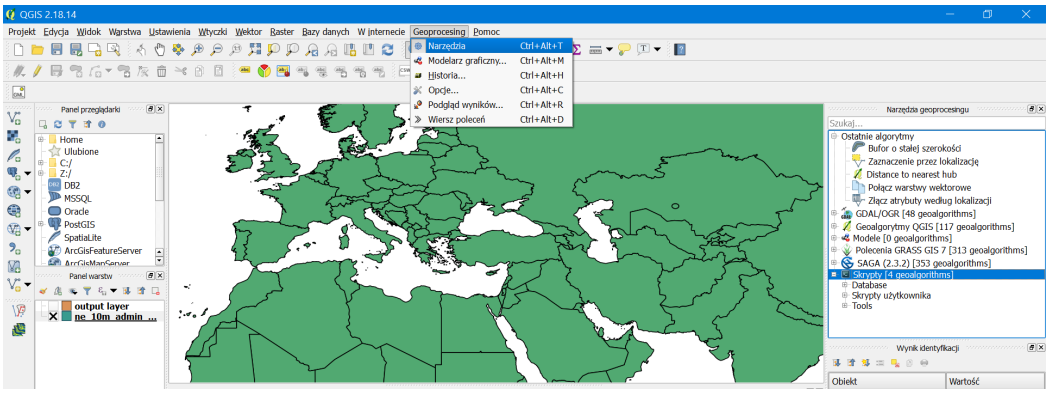
- Akcje są zdarzeniami wykonywanymi w kontekście obiektów na mapie
- Jako akcję można zaprogramować skrypt w Pythonie



Skrypty - omówienie (1)

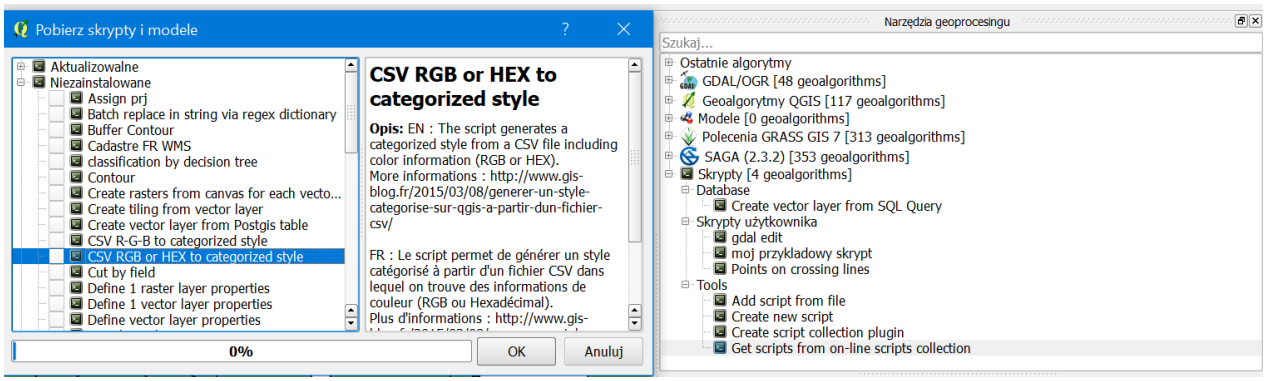
Menu: Geoprocessing -> Narzędzia

- Użytkownik ma możliwość wykorzystywania tzw. narzędzi geoprocessingu (geoalgorytmy)



Skrypty - omówienie (2)

- Istnieją gotowe skrypty, które można doinstalować z sieci i używać
- Istnieje możliwość tworzenia własnych skryptów z wykorzystaniem języka Python

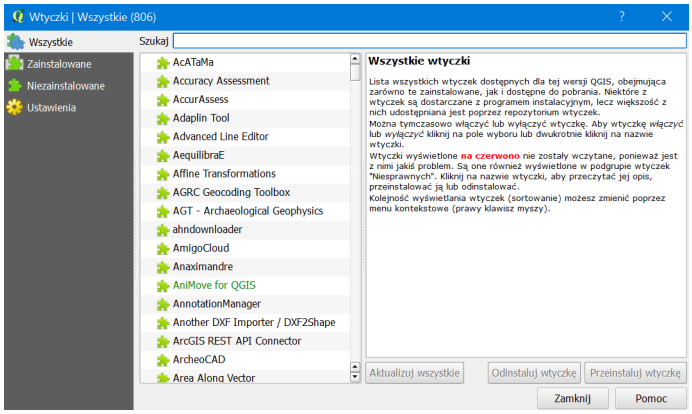


Wtyczki - omówienie

Menu: Wtyczki -> Zarządzaj wtyczkami

Są bardziej zaawansowaną formą rozszerzania funkcjonalności QGIS

Można je integrować z interfejsem graficznym QGIS



Tworzenie własnych modułów (skryptów) z użyciem edytora konsoli

Należy utworzyć plik ze skryptem *.py i umieścić go w lokalnym katalogu QGIS'a, np:

```
C:\Users\janusz\.qgis2\python\skrypt.py
```

Teraz można zaimportować ten skrypt jako moduł:

```
import skrypt
```

i wywoływać jego funkcje, np.:

```
skrypt.suma(2,5)
```

jak również uzyskać pomoc:

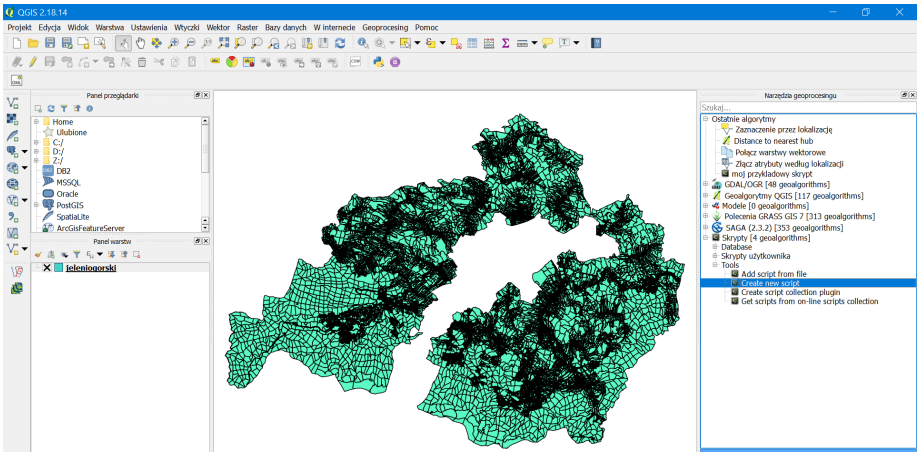
```
help(skrypt)
```

Tworzenie modułu geoprocessingu jako własnego narzędzia do przetwarzania danych

Do QGIS należy wczytać dane: warstwa działek ewidencyjnych - jeleniogorski.shp

Należy wybrać: **Menu -> Geoprocesing -> Narzędzia** aby uruchomić narzędzia geoprocessingu

Z drzewa narzędzi geoprocessingu wybrać: **Skrypty -> Tools -> Create new script**



Rozpoczęcie tworzenia modułu geoprocessingu

Stworzymy skrypt pobierający wszystkie unikalne wartości wskazanego pola (atrybutu) warstwy.

- Aby skrypt mógł być rozpoznany przez QGIS jako moduł geoprocessingu należy na początku umieścić specyfikację wejścia i wyjścia:

```
##warstwa_wejsciowa=vector
##pole_wejsciowe=field warstwa_wejsciowa
```

Składnia ogólna:

```
##[nazwa_parametru]=[typ_parametru] [wartości_opcjonalne]
```

Typy parametrów specyfikacji wejścia i wyjścia modułów geoprocessingu (1)

- **raster** - Warstwa rastrowa
- **vector** - Warstwa wektorowa
- **table** - Tabela
- **number** - Wartość numeryczna. Należy dodatkowoa podać wartość domyślną np. wysokosc=number 2.4
- **string** - ciąg tekstowy, podobnie jak w przypadku number należy podać wartość domyślną np. nazwa=string Warszawa
- **longstring** - podobnie jak string, lecz zapisywany z użyciem większej ilości pamięci.
- **boolean** - wartość logiczna TRUE/FALSE. Należy podać wartość domyślną np. drukuj=boolean True.

Typy parametrów specyfikacji wejścia i wyjścia modułów geoprocessingu (2)

- **multiple raster** - Zestaw (zbiór) wejściowych warstw rastrowych.
- **multiple vector** - Zestaw (zbiór) wejściowych warstw wektorowych.
- **field** - pole w tabeli atrybutów warstwy wektorowej. Należy podać nas]zwę warstwy z którą jest związane. Np. jeżeli mamy warstwę wejściową mojawarstwa=vector, to aby stworzyć parametr z pola tej warstwy użyjemy zapis: mojepole=field mojawarstwa
- **folder** - folder
- **file** - nazwa pliku
- **crs** - system odniesień przestrzennych (Coordinate Reference System)

Typy parametrów specyfikacji wejścia i wyjścia modułów geoprocessingu (3)

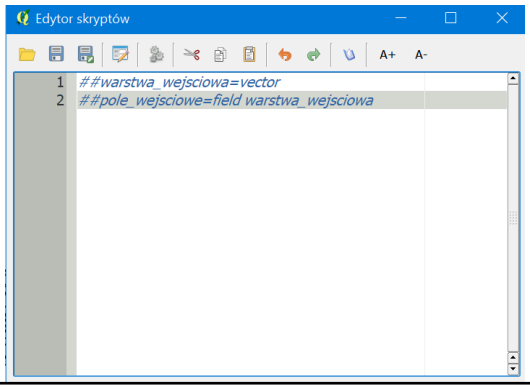
Dane wyjściowe parametryzowane są w podobny sposób

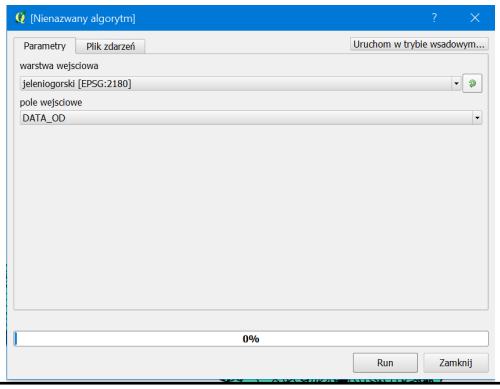
- **output raster**
- **output vector**
- **output table**
- **output html**
- **output file**
- **output number**
- **output string**
- **output extent**

Więcej informacji na ten temat:
https://docs.qgis.org/2.8/en/docs/user_manual/processing/scripts.html

Pierwsze uruchomienie modułu geoprocesingu

- Po zapisaniu i uruchomieniu skryptu automatycznie zostanie wygenerowany graficzny interfejs użytkownika, zgodny ze specyfikacją
- Zamknięcie edytora skryptów spowoduje odświeżenie drzewa skryptów, nazwa nowego skryptu pojawi się na liście. Można go uruchomić lub edytować ponownie wybierając pozycję z listy.



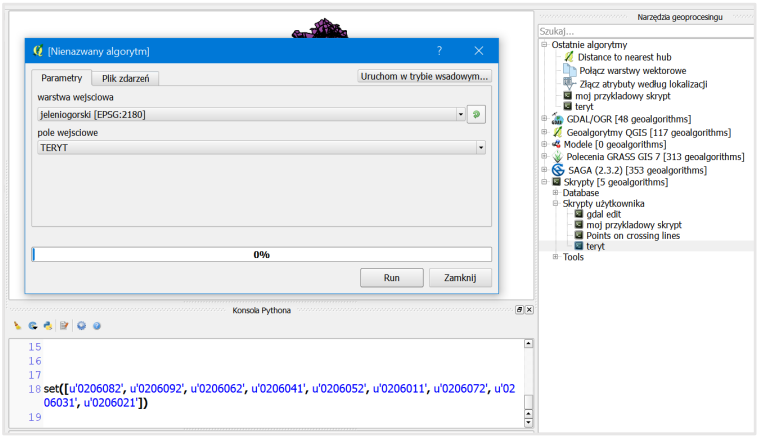


Dalsza część kodu źródłowego modułu

```
from qgis.core import *
from PyQt4.QtCore import *

inlayer = processing.getObject(warstwa_wejsciowa)

#Znajdz unikalne wartosci w polu pole_wejsciowe
unique_values = set([f[pole_wejsciowe] for f in
processing.features(inlayer)])

print unique_values
```

Użycie stworzonego modułu

- Posługując się utworzonym narzędziem wybierzmy warstwę “**jeleniogorski**” oraz pole “**TERYT**”
- Unikalne wartości **TERYT** będą widoczne w konsoli



Tworzenie wtyczki

- W QGIS możlwe jest tworzenie wtyczek w języku Python. Wtyczka po utworzeniu powinna znaleźć się w katalogu przeszukiwanym przez QGIS podczas ładowania wtyczek:

~/.qgis2/python/plugins

- Katalog domowy oznaczony ~ to przypadku Windows:

C:\Documents and Settings\(user) (Windows XP i starszy) lub C:\Users\(user)

- Utworzona wtyczka będzie widoczna w menu wtyczek razem z pozostałymi wtyczkami QGIS.

Tworzenie wtyczek - przydatne materiały

https://www.qgis.org/en/docs/

https://docs.qgis.org/2.18/en/docs/pyqgis_developer_cookbook/

https://qgis.org/api/2.18/classQgisInterface.html

Tworzenie wtyczki - kroki

1. Pomysł: Do czego będzie służyła wtyczka? Jaki problem będzie rozwiązywać? Czy istnieje wtyczka rozwiązująca ten problem?
2. Utworzenie plików: Plik startowy (__init__.py). Utworzenie pliku metadanych (metadata.txt). Plik główny z kodem źródłowym wtyczki (mainplugin.py). Formatka utworzona w QT-Designer (form.ui), oraz plik zasobów resources.qrc.
3. Napisanie programu: (ułożenie algorytmu i zapisanie kodu źródłowego): w pliku głównym mainplugin.py
4. Testowanie: Zamykanie i ponowne uruchamianie QGIS oraz ponowny import wtyczki. Sprawdzanie poprawności wykonania.

Pliki wtyczki QGIS

- **Oto pliki skladające się na przykładową wtyczkę**

SCIEZKA_WTYCZEK_PYTHONA/
 MojPlugin/
  __init__.py --> *wymagany*
  mainPlugin.py --> *wymagany*
  metadata.txt --> *wymagany*
  resources.qrc --> *przydatny*
  resources.py --> *wersja skompilowana, przydatny*
  form.ui --> *przydatny*
  form.py --> *wersja skompilowana, przydatny*

Pliki wtyczki QGIS - znaczenie

__init__.py - punkt startowy wtyczki. Musi zawierać metodę classFactory(), może zawierać dodatkowy kod inicjalizacyjny.

mainPlugin.py - główny plik wtyczki, zawierający kod źródłowy.

resources.qrc - dokument XML wytwarzany przez QT Designer. Zawiera ścieżki do plików z zasobami dla formatek.

resources.py - translacja pliku .qrc wspomanego powyżej do Pythona.

form.ui - graficzny interfejs uzytkownika, stworzony w QT Designerze.

form.py - translacja pliku form.ui opisanego wyżej do Pythona.

metadata.txt - wymagane dla QGIS wersji nowszej niż 1.8.0. Zawiera ogólne informacje opisowe na temat wtyczki. Od wersji QGIS 2.0 metadane z pliku __init__.py nie są obsługiwane natomiast metadata.txt stało się wymagane.

Metadane w pliku metadata.txt (1)

Nazwa	wymagane	opis
name	Tak	Nazwa wtyczki
qgisMinimumVersion	Tak	minimalna wersja QGIS
qgisMaximumVersion	Nie	maksymalna wersja QGIS
description	Tak	Krótki opis wtyczki
about	Tak	Dłuższy opis wtyczki
version	Tak	Wersja wtyczki
author	Tak	Autor
email	Tak	Email autora

Metadane w pliku metadata.txt (2)

Nazwa	wymagane	opis
changelog	Nie	log dotychczasowych zmian
experimental (true/false)	Nie	Czy wtyczka jest eksperymentalna - wartośc logiczna prawda/fałsz
deprecated	Nie	Wartość prawda/fałsz określająca, czy wtyczka podlega rozwojowi
tags	Nie	lista słów kluczowych oddzielonych przecinkami
homepage	Nie	URL strony domowej
repository	Tak	URL repozytorium kodu źródłowego
tracker	Nie	URL monitoringu i raportowania błędów
icon	Nie	nazwa pliku lub ścieżka do obrazu (ikony) w formacie PNG lub JPG
category	Nie	Kategoria określona jako: Raster, Vector, Database albo Web

Plik __init__.py

Ten plik jest wymagany przez system importu interpretera Pythona. QGIS wymaga, aby plik ten zawierał funkcję classFactory(), która jest wywoływana podczas ładowania wtyczki do QGIS. Pobiera ona referencję do instancji klasy Qgisinterface i musi zwrócić instancję klasy reprezentującej wtyczkę (z pliku głównego - np. mainplugin.py, gdzie klasa nazywa się np. TestPlugin). Plik __init__.py może wyglądać tak:

```
def classFactory(iface):
  from mainPlugin import TestPlugin
  return TestPlugin(iface)

## pozostałe parametry inicjalizacji umieszczamy w tym miejscu
```

Plik główny wtyczki (np. mainPlugin.py)

- Plik zawiera implementację klasy reprezentującej wtyczkę. W wersji minimalnej powinien zawierać metody:

`__init__` –> daje dostęp do interfejsu QGIS'a

`initGui()` –> wywoływana podczas ładowania wtyczki

`unload()` –> wywoływana podczas usuwania wtyczki z pamięci

Aby dodać pozycję menu (grupa:Wtyczki) ładującą wtyczkę można posłużyć się metodą interfejsu QGIS:

`iface.addPluginToMenu()`

Plik zasobów - resources.qrc

- Plik definiujący zasoby interfejsu GUI ma postać np.:

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

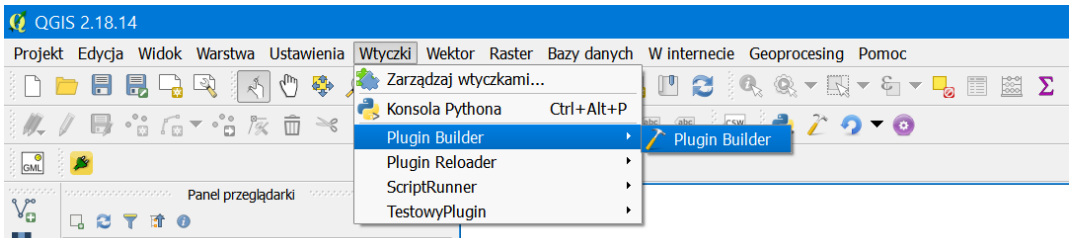
Zanim uruchomimy wtyczkę, należy z jego pomocą wygenerować plik z zasobami w formie skryptu Pythona. Można tego dokonac poleceniem w konsoli OSGeo4W Shell:

```
pyrcc4 -o resources.py resources.qrc
```

Stworzenie wtyczki z wykorzystaniem Plugin Buildera (1)

- Zadanie stworzenia wszystkich potrzebnych plików wtyczki można sobie ułatwić wykorzystując wtyczkę Plugin Builder.
- W tym celu uruchamiamy ją poleceniem:

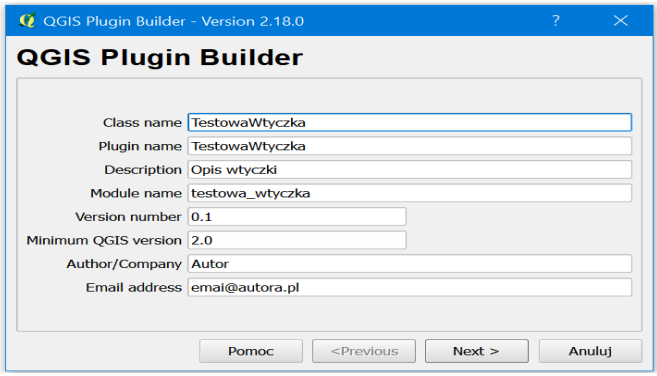
Wtyczki -> Plugin Builder -> Plugin Bulder

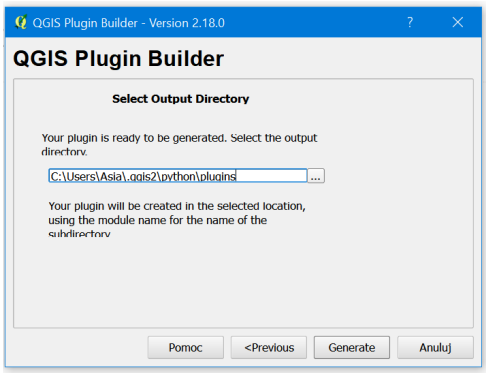


Stworzenie wtyczki z wykorzystaniem Plugin Buildera (2)

- Uzupełniamy wymagane dane w poszczególnych oknach naciskając **NEXT >**.
- W ostatnim oknie podajemy katalog, w którym po naciśnięciu "**Generate**" zapiszemy wtyczkę. Najlepiej podać katog z wtyczkami QGIS, będzie on podobny do poniższego:

  C:\Users\Asia\.qgis2\python\plugins





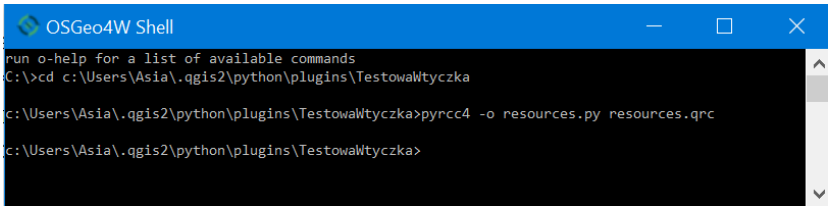
Uruchamianie wtyczki (1)

- Zanim użyjemy wtyczki należy wygenerować plik resources.py na podstawie pliku resources.qrc
- W tym celu uruchamiamy konsolę **OSGeo4WShell**, przechodzimy do katalogu z wtyczką, np.:

cd c:\users\.qgis2\python\plugins\TestowaWtyczka

i wydajemy polecenie:

```
pyrcc4 -o resources.py resources.qrc
```



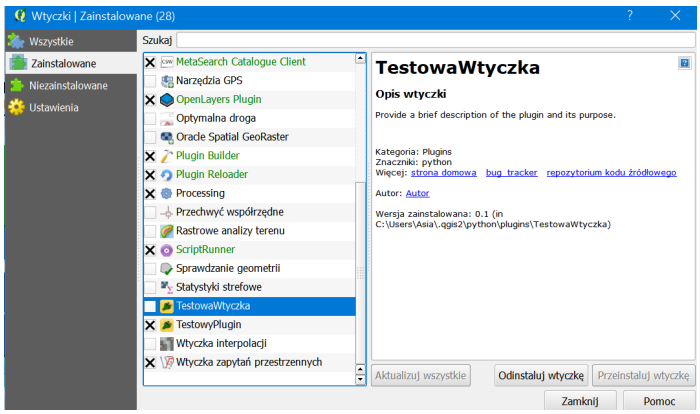
Uruchamianie wtyczki (2)

- Teraz należy zamkąć i ponownie otworzyć QGIS.
- Wtyczka powinna być dostępna w menu

Wtyczki -> Zarządzaj wtyczkami

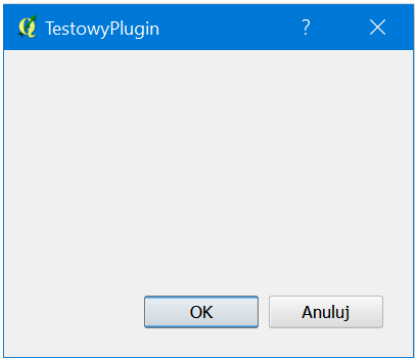
jako zainstalowana i nieaktywna

- Należy ją aktywować



Uruchamianie wtyczki (3)

- Po aktywacji wtyczki można ją uruchomić, wybierając pozycję z menu **Wtyczki**



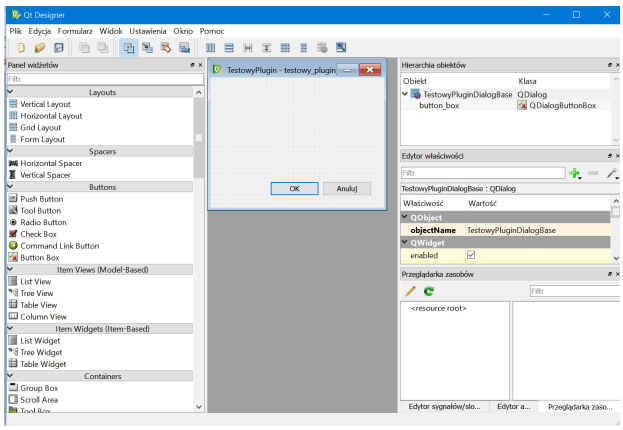
Modyfikacja interfejsu użytkownika wtyczki (1)

- Elementy interfejsu graficznego można edytować z wykorzystaniem programu QTDesigner, dostarczonego z instalacją OSGeo4W

menu Start -> Qgis -> Qt Designer

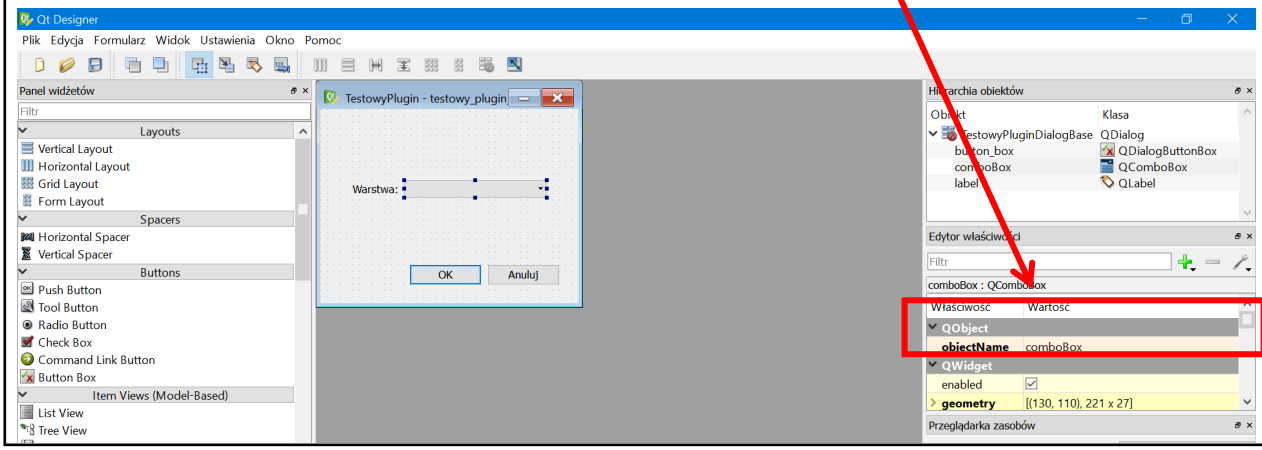
W nim otwieramy plik **.ui** zawierający definicję okna tworzonej wtyczki

C:\Users\Asia\.qgis2\python\plugins\TestowyPlugin



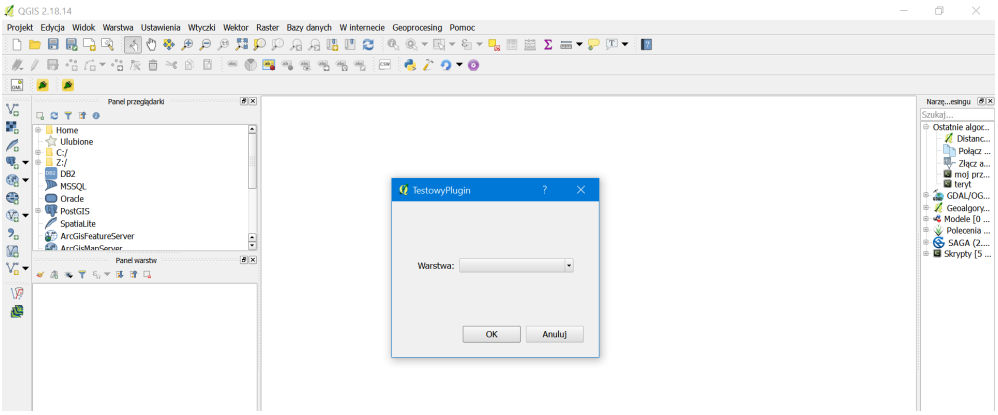
Modyfikacja interfejsu użytkownika wtyczki (2)

- Z pomocą edytora dodajemy **etykietę** i pole wybieralne (**combobox**)
- Zapamiętujemy, że nazwa dodanego pola to “**comboBox**”
- Zapisujemy zmiany (**Plik -> Zachowaj**)



Modyfikacja interfejsu użytkownika wtyczki (3)

- W Qgis przeładowujemy wtyczkę (wtyczką Plugin Reloader), lub zamykamy i otwieramy Qgis ponownie



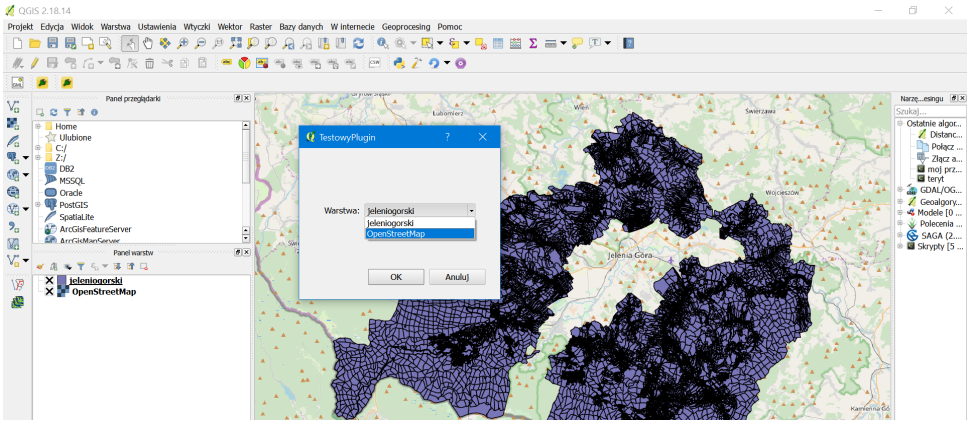
Dodanie funkcjonalności do wtyczki - ładowanie listwy warstw do comboBox (1)

- W głównym module wtyczki, za pomocą edytora tekstowego dodajemy poniższy kod do metody **run(self)**:

```
warstwy = self.iface.legendInterface().layers()
lista_warstw = []
for warstwa in warstwy:
    lista_warstw.append(warstwa.name())
self.dlg.comboBox.addItems(lista_warstw)
```

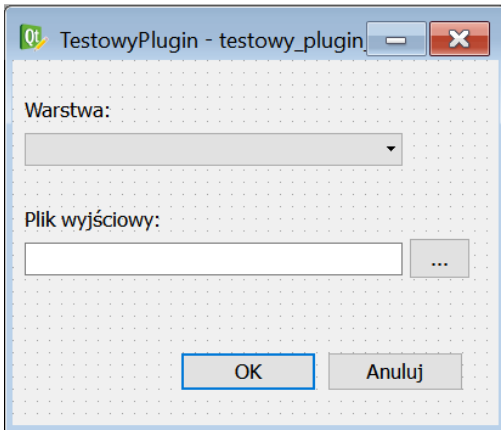
Dodanie funkcjonalności do wtyczki - ładowanie listwy warstw do comboBox (2)

- Po przeładowaniu i dodaniu kilku warstw wtyczka będzie wyświetlała ich nazwy w polu wyboru



Dodanie funkcjonalności do wtyczki - zapis atrybutów warstwy do pliku (1)

- rozbudowa GUI o elementy: **Label, LineEdit i PushButton**



Dodanie funkcjonalności do wtyczki - zapis atrybutów warstwy do pliku (2)

Obsługa zdarzenia naciśnięcia przycisku PushButton - otwarcie okna dialogowego wyboru plików

1. W głównym pliku należy zaimportować klasę QFileDialog, w linii:

```
from PyQt4.QtGui import QAction, QIcon, QFileDialog
```

2. Dodać nową metodę odpowiedzialną za wyświetlene dialogu i pobranie nazwy pliku (umieścić ją przed metodą **run()**):

```
def wybierz_plik_wyjsciowy(self):
    nazwaPliku = QFileDialog.getSaveFileName(self.dlg,
                    "Wybierz plik ","", '*.txt')
     self.dlg.lineEdit.setText(nazwaPliku)
```

Dodanie funkcjonalności do wtyczki - zapis atrybutów warstwy do pliku (3)

Powiązanie zdarzenia kliknięcia przycisku z metodą wybierz_plik_wyjsciowy().

3. W metodzie **__init__()** - na końcu - dodać kod:

```
self.dlg.lineEdit.clear()
self.dlg.pushButton.clicked.connect(self.wybierz_plik_wyjsciowy)
```

UWAGA: deklaracja obiektu reprezentującego główne okno wtyczki, np.:

```
self.dlg = TestowyPluginDialog()
```

Powinna znajdować się **wewnątrz metody __init__**. W przeciwnym wypadku **należy ją tam przenieść.**

Dodanie funkcjonalności do wtyczki - zapis atrybutów warstwy do pliku (4)

4. Dodanie do metody **run()** obsługi zapisu atrybutów warstwy do pliku tekstowego.

```
if result:
    nazwaPliku = self.dlg.lineEdit.text()
    plik_wyjsciowy = open(nazwaPliku, 'w')

    wybranaWarstwaIndex = self.dlg.comboBox.currentIndex()
    wybranaWarstwa = warstwy[wybranaWarstwaIndex]
    pola = wybranaWarstwa.pendingFields()
    nazwyPol = [pole.name() for pole in pola]

    for f in wybranaWarstwa.getFeatures():
        linia = ','.join(unicode(f[x]) for x in nazwyPol) + '\n'
        linia_unicode = linia.encode('utf-8')
        plik_wyjsciowy.write(linia_unicode)
    plik_wyjsciowy.close()
```

Kompilacja do exe - przygotowanie systemu

Instalacja Pyinstaller. W konsoli systemowej (najwygodniej jest uprzednio dodać ścieżkę Pythona do zmiennej systemowej PATH):

python -m pip install pyinstaller

W przypadku systemu Windows trzeba również zainstalować:

PyWin32

dostępne na: https://github.com/mhammond/pywin32/releases

Kompilacja do exe -wytworzenie zbioru

Kompilacja skryptu **witaj.py** do pliku exe

python -m PyInstaller --clean -F -d -y witaj.py

W wyniku działania modułu PyInstaller powstają dwa katalogi:

- **build** - zawiera pliki związane z kompilacją
- **dist** - zawiera utworzony plik .exe

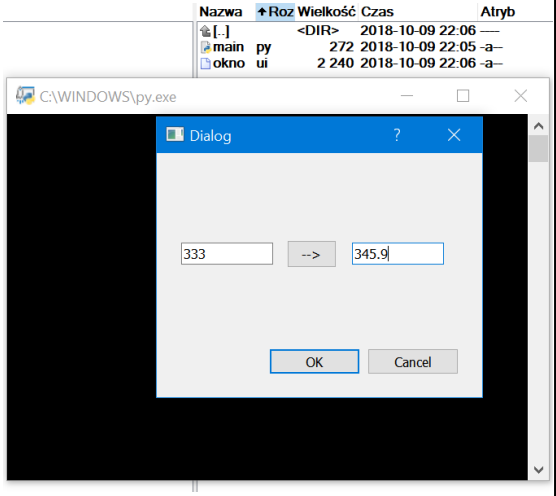
Kompilacja do exe - uwagi (1)

- Kompilacja programu Python do pliku wykonywalnego sprawia, że program staje się zależny od platformy, na której dokonano kompilacji.
- Ponadto wytworzony w ten sposób program uruchamia się i działa wolniej.
- Programy zawierające wiele zależności (jak wtyczki QGIS) mogą po kompilacji zajmować bardzo dużo pamięci, lub w ogóle nie będą działać.

Kompilacja do exe - uwagi (2)

- Ideą skryptowych języków, do których zalicza się Python, jest możliwość uruchamiania programów na dowolnym komputerze wyposażonym w interpreter danego języka. To sprawia, że są one uniwersalne i niezależne od platformy.
- Mając zainstalowany i skonfigurowany interpreter używanie programów Python jest równie wygodne jak programów kompilowanych do exe.
- Każdy skrypt konsolowy można uruchomić poleceniem:

  python nazwa_skryptu.py
- Ponadto każdy skrypt(konsolowy lub z GUI) można uruchomić klikając na plik .py tak samo jak na uruchamialny .exe



MINISTERSTWO ŚRODOWISKA

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

TRAPPA

Szkolenie
Programowanie Python w QGIS

Dziękuję za uwagę

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej