



Kurs programowania w QGIS za pomocą Pythona

Hel 23.9 – 27.9 2019

poniedziałek

PLAN (zarys ogólny – możliwe przesunięcia)

Dzień 1:

- Konsola Pythona w QGIS i jej wykorzystanie
- Wykorzystanie skryptów w Pythonie (**Select by expression**, **Field Calculator**)
- Wykorzystanie PyQGIS bindings
- Wykorzystywanie wtyczki Script Runner

Dzień 2:

- Wykorzystywanie w programach funkcji lub narzędzi z QGIS i Sagi itd. (biblioteka *processing*)
- Wykorzystanie biblioteki **OGR** w QGIS - czytanie i zapisywanie danych wektorowych, tworzenie i zmienianie obiektów geometrycznych (punkty, linie, poligony),

Dzień 3:

- Wykorzystanie biblioteki **OGR** w QGIS cd. - filtry przestrzenne i geoprzetwarzanie
- Programowanie rastrowe - biblioteka **GDAL**

Dzień 4:

- Programowanie rastrowe biblioteka **GDAL** cd.
- Tworzenie wtyczek QGIS z wykorzystaniem **Plugin Builder, Qt Designer**

Dzień 5:

- Tworzenie wtyczek QGIS z wykorzystaniem **Plugin Builder, Qt Designer** cd.

Organizacja zajęć

Poniedziałek - Czwartek

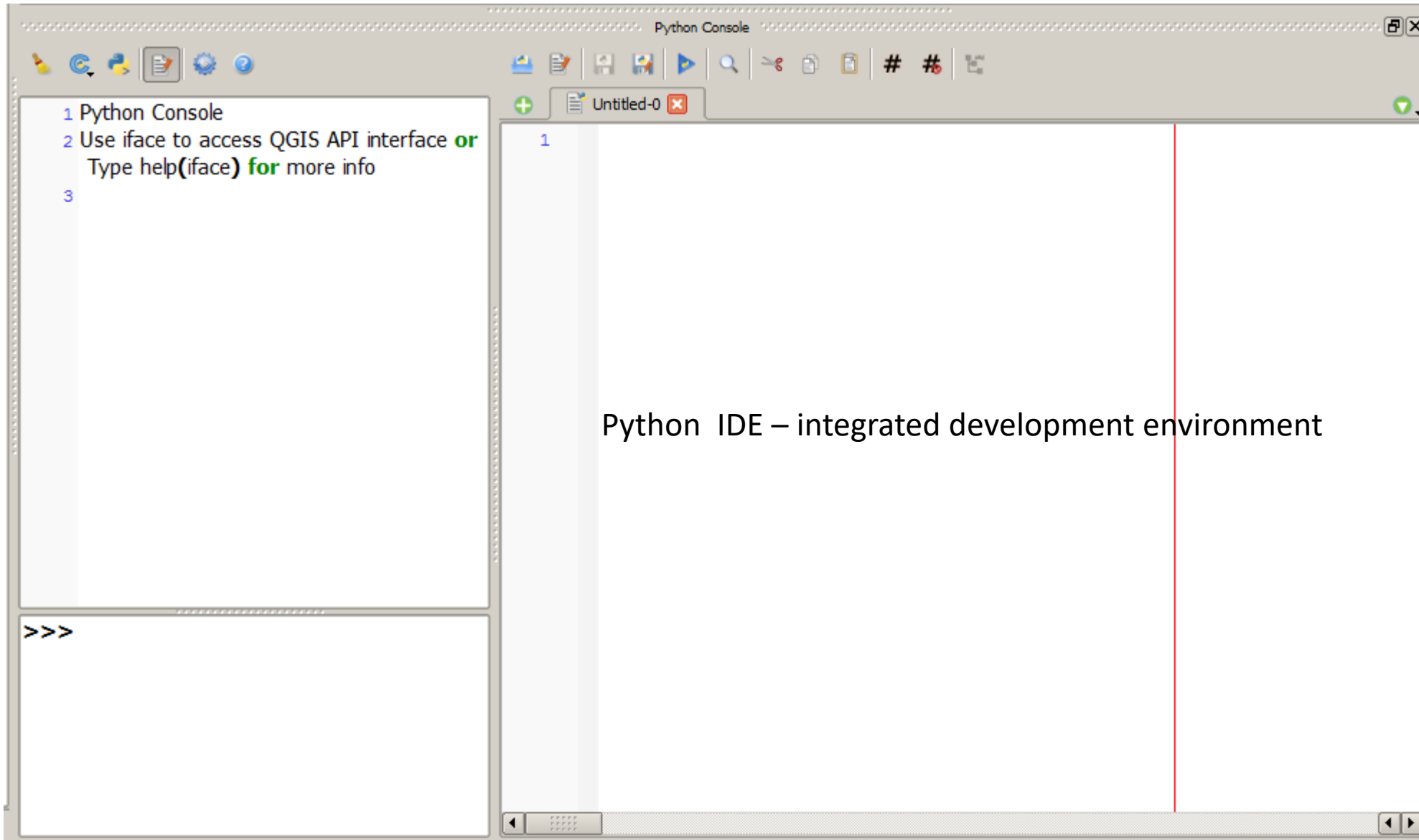
9 – 11 Warsztaty

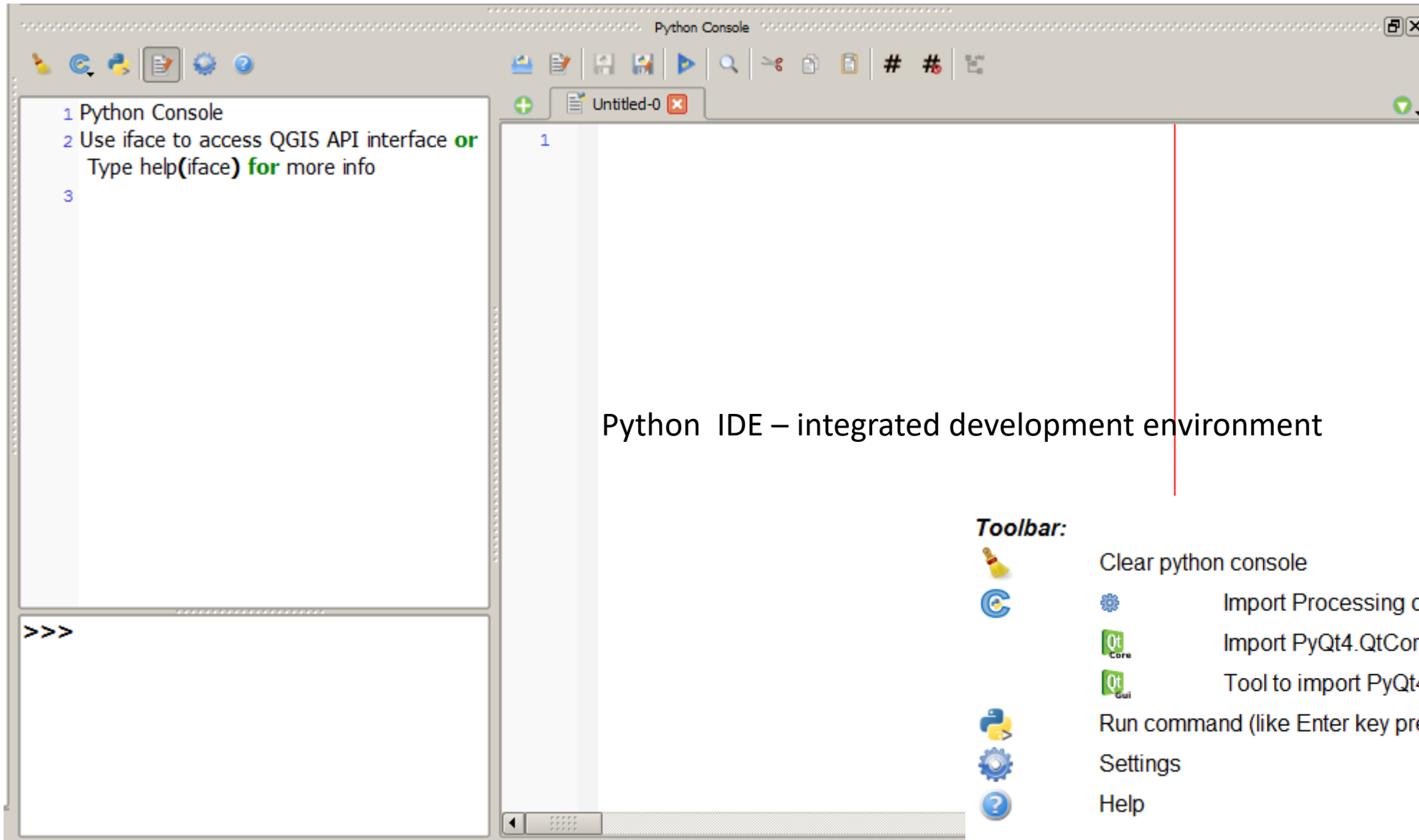
13 – 15 Warsztaty

17 – 19 Ćwiczenia - rozwiązywanie zadań

Piątek

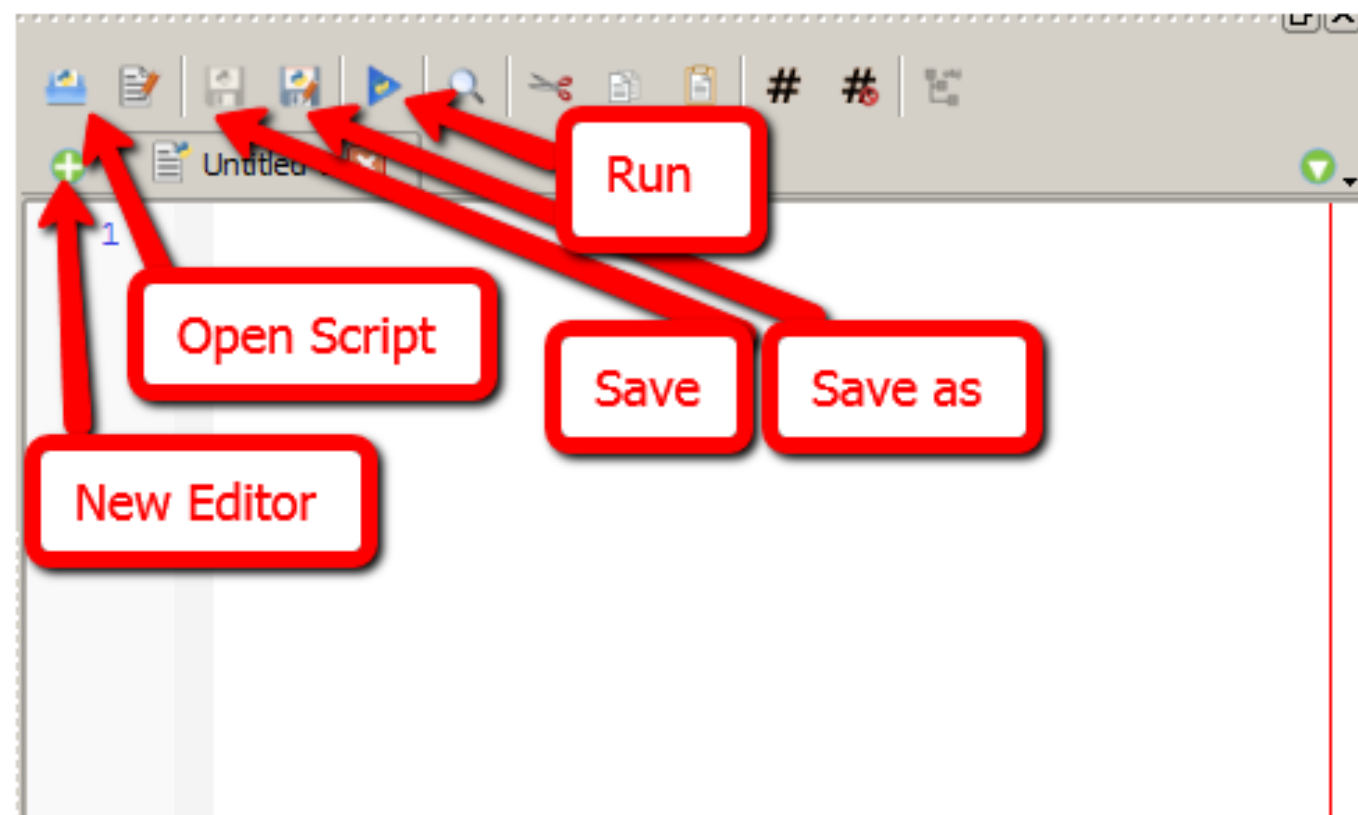
9 – 13 Warsztaty





Python IDE – integrated development environment

Zaznaczenie ikonki  wyświetla IDE Pythona.



Zad 1

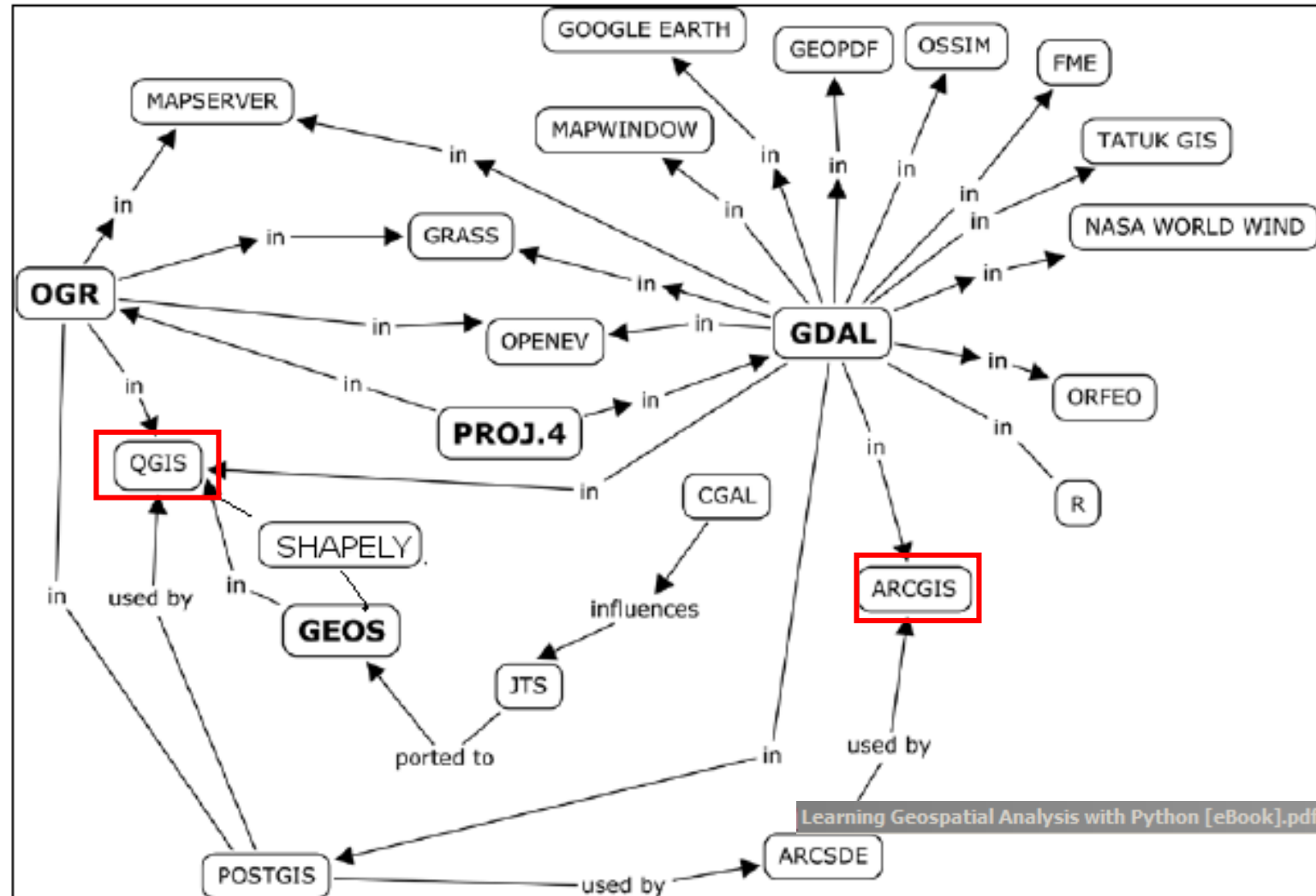
Wylosuj liczbę całkowitą od 0 do 1000, następnie iterując sprawdź jaka liczba została wylosowana.

moduł `random`, funkcja `randint(od,do)`

Zad 2

Napisz program, który korzystając z funkcji `uniform(od,do)` wylosuje 50 liczb od 0 do 1000 i wydrukuje liczbę największą i najmniejszą.

Geospatial technology ecosystem



QGIS – Python Ekosystem

QGIS > wtyczki (plugins) do dynamicznego dodawania nowych możliwości programu

Do 2007 (do wersji 0.9)

Realizacja za pomocą C++ (ograniczona liczba osób tworzących skrypty)

Od 2007 (od wersji 0.9)

Dodanie Pythona jako języka skryptowego

1. *lingua franca* dla świata skryptowego GIS
2. dobra integracja z QGIS API
3. znaczny wzrost liczby osób piszących skrypty > rozwój QGIS

QGIS – Python Ekosystem

QGIS + Python

QGIS jest napisany w C++ i zawiera ponad 400 core classes tworzących podstawowy program

Około 75% z tych klas jest dostępnych z poziomu Pythona za pomocą SIP - Session Initiation Protocol – (łączenie modułów w Pythonie z C++)
(tworząc PyQGIS bindings – interfejs do klas z poziomu Pythona – łączy kod Pythona z kodem C++ QGISa)

Technicznie dostęp z poziomu Pythona odbywa się za pomocą szeregu modułów:

- qgis.core - core classes
- qgis.gui - graphical user interface classes
- qgis.analysis –
- qgis.networkanalysis -

QGIS – Python Ekosystem

Python >> PyQGIS >> QGIS

Wykorzystanie Pythona w QGISie może się odbywać na różnych poziomach:

- Wykorzystanie funkcji napisanych w Pythonie dla poszerzenia możliwości podstawowych funkcji programu takich jak **Select by expression** i **Field Calculator**)
- Wprowadzanie komend w konsoli Pythona w QGIS.
- Tworzenie skryptów w Pythonie, które mogą być uruchamiane za pomocą wtyczki **Script Runner**
- Tworzenie skryptów (programów) uruchamianych z poziomu konsoli Pythona
- Tworzenie wtyczek w Pythonie
- Tworzenie samodzielnych aplikacji wykorzystujących QGIS API (*Application Programmer Interface*)

Python >> OGR, GDAL , numpy, scipy, pyplot >> PyQGIS >> QGIS

Zadanie 3

Napisz funkcję przypisującą klasę w zależności od roku trzęsienia Ziemi:

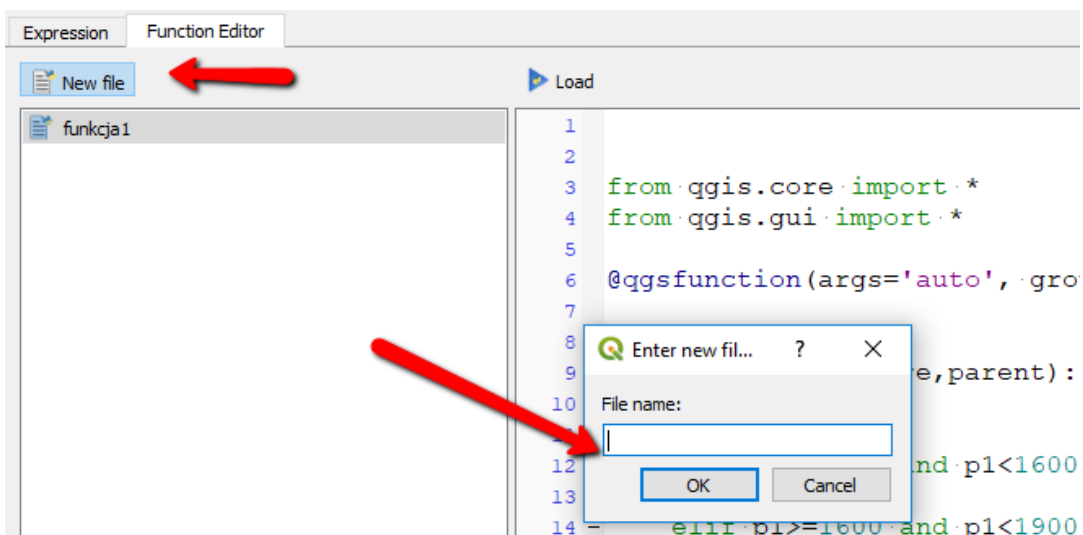
	Klasa	lata
--	-------	------

- | | |
|---|-------------|
| 1 | Do 1000 |
| 2 | 1000 – 1600 |
| 3 | 1600 – 1900 |
| 4 | 1900 – 2000 |
| 5 | Od 2000 |

Wykorzystaj funkcję do przypisania klasy do utworzonego w tabeli atrybutowej tablicy warstwy **trzęsienia** pola

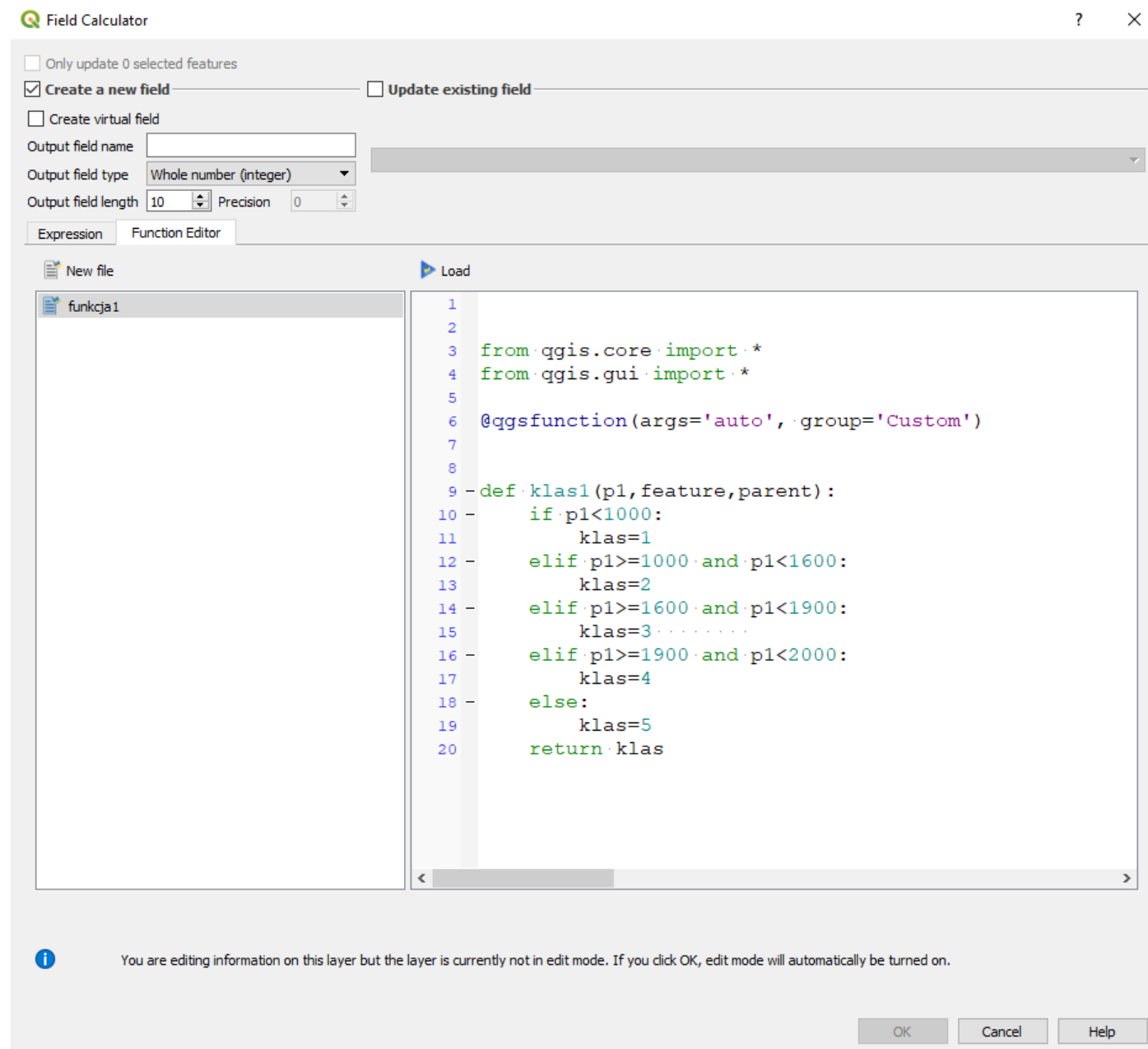
```
def klas1(p1,feature,parent):  
  
    if p1<1000:  
  
        klas=1  
    elif p1 >=1000 and p1<1600:  
  
        klas=2  
  
    elif p1>=1600 and p1<1900:  
  
        klas=3  
  
    elif p1>=1900 and p1<2000:  
  
        klas=4  
  
    else:  
  
        klas=5  
  
    return klas
```

Wykorzystywanie funkcji napisanych w Pythonie (Select by expression, Field Calculator)

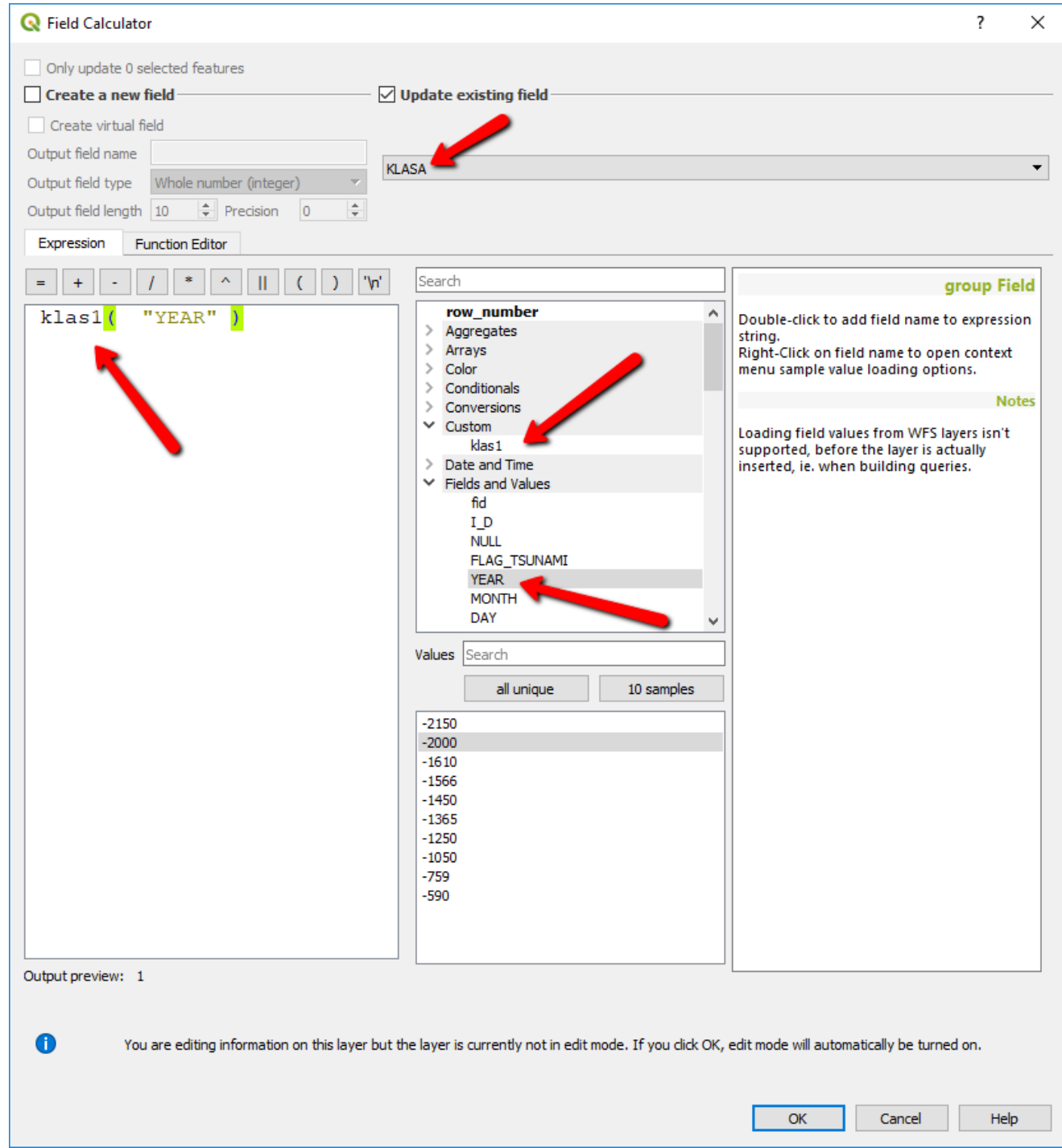


Po wpisaniu nazwy otwarty zostanie plik zawierający gotowy schemat, który musi zostać uzupełniony. Po wpisaniu funkcji naciskamy klawisz **Load**, funkcja (o ile nie pojawi się okno błędów) będzie zapisana w katalogu C:\Users\CentrumGIS\AppData\Roaming\QGIS\QGIS3\profiles\default\python\expressions .

Przy wpisywaniu funkcji do wcięć należy używać wyłącznie spacji, a najlepiej tworzyć ją i testować w dowolnym IDE (Interactive Development Environment) Pythona i następnie przekleić.



Funkcja jest następnie wywoływana w drugiej zakładce. Utworzone funkcje znajdują się pod **Custom**, a pola pod **Fields and Values**.



Zadanie 4

Napisz funkcję zaznaczającą trzęsienia dla zakresu lat w których wystąpiło tsunami:

parametry : od,do

```
1 qgis.core import *
2 qgis.gui import *
3
4 unction(args='auto', group='Custom')
5
6 -ddo_tsunami(od, do, fczas, ftsun, feature, parent):
7 -f fczas>=od and fczas<=do and ftsun=='Tsu':
8     return True
9 -lse:
10    return False
```

Select by Expression - trzesienia_ziemi

ExpressionFunction Editor

= + - / * ^ || () 'n'

oddo_tsunami(1500,1900,
"YEAR" , "FLAG_TSUNAMI")

Search

Custom

klas1
oddo_tsunami

> Date and Time

> Fields and Values

fid
I_D
NULL
FLAG_TSUNAMI
YEAR
MONTH

Values Search

all unique10 samples

group Field

Double-click to add field name to expression string.
Right-Click on field name to open context menu sample value loading options.

Notes

Loading field values from WFS layers isn't supported, before the layer is actually inserted, ie. when building queries.

PyQGIS developer cookbook

Release testing

QGIS Project

Otwarcie Konsoli powoduje wykonanie:

```
from qgis.core import *  
import qgis.utils
```

Przykłady pracy w konsoli:

```
21 >>> warstwa=iface.activeLayer()  
22 >>> print(warstwa.name())  
23 Kraje
```

Spowoduje wydrukowanie aktualnej warstwy aktywnej (zaznaczonej).

Otwieranie warstw wektorowych (shp)

The quickest way to open and display a vector layer in QGIS is the `addVectorLayer` function of the `QgisInterface`:

```
layer = iface.addVectorLayer("/path/to/shapefile/file.shp", "layer name you like",  
    ↪ "ogr")  
if not layer:  
    print("Layer failed to load!")
```

This creates a new layer and adds it to the map layer registry (making it appear in the layer list) in one step. The function returns the layer instance or *None* if the layer couldn't be loaded.

- OGR library (shapefiles and many other file formats) — data source is the path to the file:
 - for shapefile:

Istnieją „providers” dla innych warstw wektorowych: (Spatialite, GPX)

Otwieranie warstw rastrowych

Similarly to vector layers, raster layers can be loaded using the `addRasterLayer` function of the `QgisInterface`:

```
iface.addRasterLayer("/path/to/raster/file.tif", "layer name you like")
```

This creates a new layer and adds it to the map layer registry (making it appear in the layer list) in one step.

Podstawowe operacje na danych rastrowych

```
rlayer.width(), rlayer.height()  
(812, 301)  
rlayer.extent()  
<qgis._core.QgsRectangle object at 0x000000000F8A2048>  
rlayer.extent().toString()  
u'12.095833,48.552777 : 18.863888,51.056944'  
rlayer.rasterType()  
2 # 0 = GrayOrUndefined (single band), 1 = Palette (single band), 2 = Multiband  
rlayer.bandCount()  
3  
rlayer.metadata()  
u'<p class="glossy">Driver:</p>...'  
rlayer.hasPyramids()  
False
```

```
ras1=iface.activeLayer()

rxt=ras1.extent()
print(rxt.xMaximum(),rxt.xMinimum(),rxt.yMaximum(),rxt.yMinimum())
print(ras1.rasterUnitsPerPixelX())
```

Pozostałe można znaleźć za pomocą instrukcji: `help(obiekt)` np. `help(rxt)`

Ekstrakcja wartości z rastra: (rezultat jest wyprowadzany jako dictionary)

```
ident = ras1.dataProvider().identify(QgsPointXY(0, 0), \
QgsRaster.IdentifyFormatValue)
- if ident.isValid():
    print(ident.results())
```

Program do losowego pobierania wartości z rastra

Podstawowe operacje na danych wektorowych

```
# "layer" is a QgsVectorLayer instance
for field in layer.fields():
    print(field.name(), field.typeName())
```

```
vec1=iface.activeLayer()

- for field in vec1.fields():
    print(field.name(), field.typeName())
```

```
layer = iface.activeLayer()
features = layer.getFeatures()
```

Pozyskiwane za pomocą iteracji

```
for feature in features:
    # retrieve every feature with its geometry and attributes
    # fetch geometry
    geom = feature.geometry()
    print("Feature ID: ", feature.id())
    # show some information about the feature
    if geom.wkbType() == QgsWkbTypes.Point:
        x = geom.asPoint()
        print("Point:", x)
    elif geom.wkbType() == QgsWkbTypes.LineString:
        x = geom.asPolyline()
        print('Line:', x, 'points', 'length:', geom.length())
    elif geom.wkbType() == QgsWkbTypes.Polygon:
        x = geom.asPolygon()
        print("Polygon:", x, "Area: ", geom.area())
    else:
        print("Unknown")

    # fetch attributes
    attrs = feature.attributes()

    # attrs is a list. It contains all the attribute values of this feature
    print(attrs)
```


Informacja o danych wektorowych

```
6  vec1=iface.activeLayer()
7
8  - for field in vec1.fields():
9      print(field.name(), field.typeName())
10
11  features=vec1.getFeatures()
12  i=0;k=0
13  - for feature in features:
14      - if i==0:.....# informacje czytamy z geometrii, ale tylko pierwszej
15          geom=feature.geometry()
16      - if geom.wkbType() == QgsWkbTypes.Point:
17          print('Punkty')
18      - if geom.wkbType() == QgsWkbTypes.LineString:
19          print('Linie').....
20      - if geom.wkbType() == QgsWkbTypes.MultiLineString: #Create layer QGIS
21          print('LinieM')
22      - if geom.wkbType() == QgsWkbTypes.Polygon:
23          print('Poligony')
24      - if geom.wkbType() == QgsWkbTypes.MultiPolygon:.....#Create layer QGIS
25          print('PoligonyM').....
26          i+=1
27      k+=1.....
28  print('rekordow', k)
```

Dostęp do atrybutów

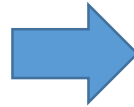
```
11 features=vec1.getFeatures()  
12  
13 - for feature in features:  
14     attrs=feature.attributes()  
15     print(attrs) ....# lista
```



```
19 [1, 'kolej']  
20 [2, 'droga']  
21 [3, 'droga']
```

Dostęp do geometrii danych wektorowych - punkty

```
6  vec1=iface.activeLayer()  
7  
8  
9  features=vec1.getFeatures()  
10  
11  - for feature in features:  
12      geom=feature.geometry()  
13      xx=geom.asPoint()  
14      print(xx)  
15      print(xx[0],xx[1])  
16
```



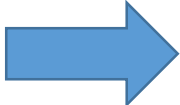
```
86 (343376,6.02974e+6)  
87 343376.35733059514 6029735.630404593  
88 (342796,6.02754e+6)  
89 342795.7234238153 6027542.89140059  
90
```

Dostęp do geometrii danych wektorowych - linie

```
vec1=iface.activeLayer()
```

```
features=vec1.getFeatures()
```

```
- for feature in features:  
    geom=feature.geometry()  
    xx=geom.asMultiPolyline()  
    print(geom.length())  
    print(1,xx) # lista (one feature)  
    print(2,'part1',xx[0]) # part 1  
    print(3,xx[0][1]) # tupla 2 punkt  
    print(4,xx[0][1][0]) # wspolrzedna
```



```
160 2 part1 [(336147,6.02566e+6), (336720,6.02585e+6), (336  
918,6.02581e+6), (337324,6.02568e+6), (337628,6.02553e+  
6), (337943,6.02539e+6), (338181,6.02535e+6), (338536,6.  
.02525e+6), (338841,6.0252e+6), (339135,6.02514e+6), (3  
39698,6.02496e+6), (340256,6.02471e+6), (340697,6.02428  
e+6), (340991,6.02398e+6), (341265,6.02371e+6), (341630  
,6.02359e+6), (342087,6.02351e+6), (342604,6.02354e+6),  
(343401,6.02366e+6), (344091,6.02388e+6), (344334,6.02  
396e+6), (344816,6.0242e+6), (345151,6.02444e+6), (3453  
64,6.02455e+6), (345739,6.02455e+6), (345952,6.02457e+6  
, (346120,6.02472e+6), (346383,6.02488e+6), (346581,6.  
02488e+6), (346754,6.02479e+6), (347002,6.02474e+6), (3  
47469,6.02458e+6), (347682,6.02451e+6), (347895,6.02443  
e+6)]
```

```
161 3 (336720,6.02585e+6)
```

```
162 4 336720.2355936768
```

```
features=vec1.getFeatures()
```

ometrii danych wektorowych - poligony

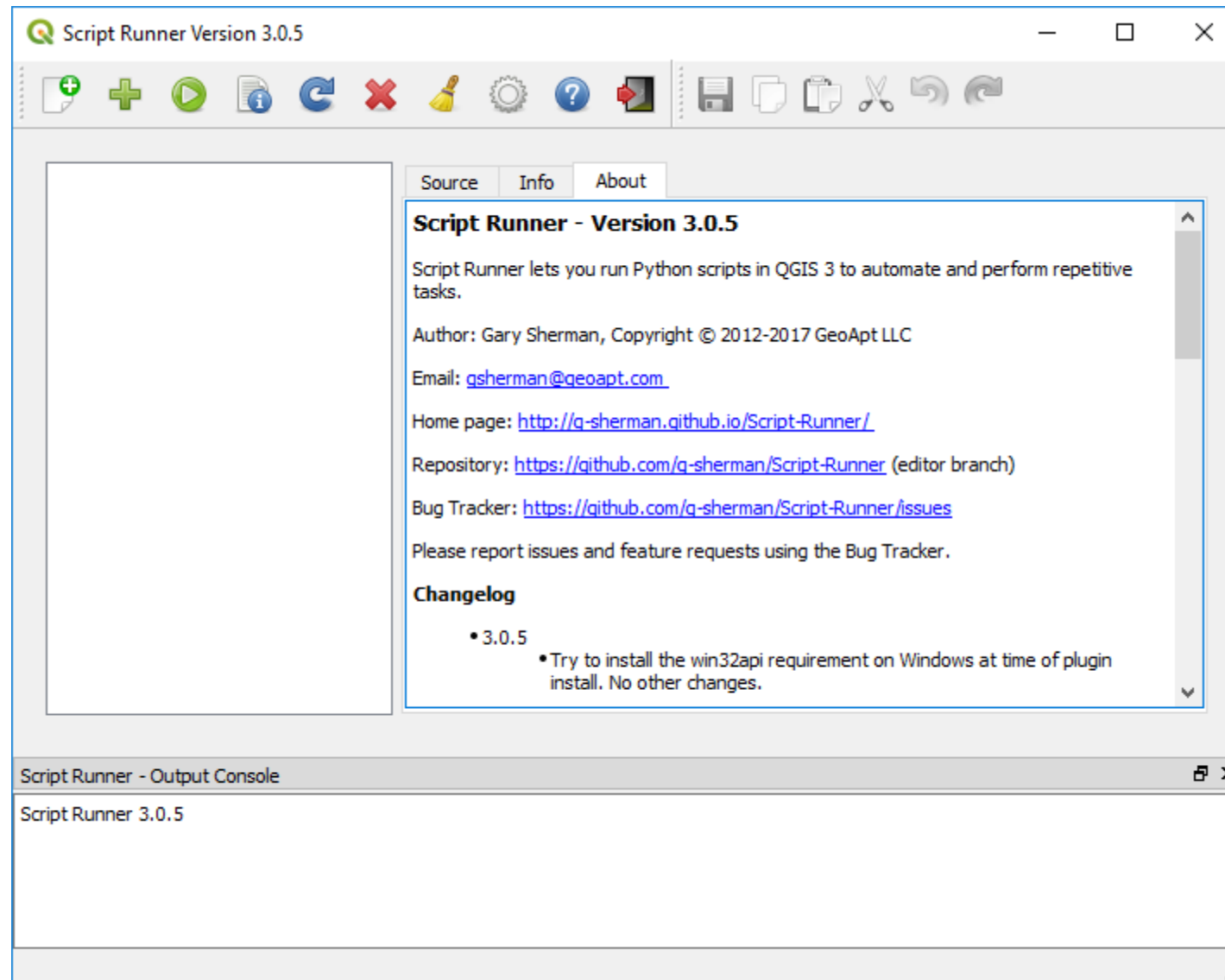
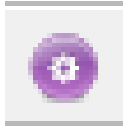
```
- for feature in features:  
    geom=feature.geometry()  
    xx=geom.asMultiPolygon()  
    print(geom.length(),geom.area())  
    print(1,xx)    # lista (one feature)  
    print(2,'part1',xx[0]) # part 1  
    print(3,xx[0][0]) # ring 1  
    print(4,xx[0][0][1]) # tupla wspolrzedne  
    print(5,xx[0][0][1][0]) # wspolrzedna
```

```
222 3 [(347901,6.02863e+6), (347914,6.02826e+6), (347831,6.02824e+6), (347886,6.02804e  
+6), (347899,6.02805e+6), (347906,6.02799e+6), (347891,6.02798e+6), (347901,6.0279  
e+6), (347919,6.0279e+6), (347922,6.02739e+6), (347873,6.02741e+6), (347852,6.0274  
7e+6), (347849,6.02754e+6), (347807,6.02761e+6), (347721,6.02746e+6), (347672,6.02  
747e+6), (347854,6.0278e+6), (347784,6.028e+6), (347664,6.02782e+6), (347617,6.027  
75e+6), (347544,6.02763e+6), (347491,6.02766e+6), (347734,6.02808e+6), (347724,6.0  
2813e+6), (347591,6.02812e+6), (347567,6.02816e+6), (347695,6.02819e+6), (347719,6  
.02823e+6), (347708,6.02827e+6), (347588,6.02825e+6), (347585,6.02821e+6), (347541  
,6.0282e+6), (347531,6.02824e+6), (347549,6.02827e+6), (347745,6.02834e+6), (34775  
0,6.02843e+6), (347536,6.02838e+6), (347403,6.02837e+6), (347238,6.02824e+6), (347  
199,6.02817e+6), (347152,6.02816e+6), (347071,6.02796e+6), (347259,6.02781e+6), (3  
47243,6.02779e+6), (347037,6.02792e+6), (347024,6.02785e+6), (346899,6.02789e+6),  
(346967,6.0281e+6), (347100,6.02829e+6), (347324,6.02846e+6), (347337,6.02844e+6),  
(347382,6.02846e+6), (347369,6.02848e+6), (347408,6.02849e+6), (347515,6.02851e+6  
) , (347687,6.02856e+6), (347901,6.02863e+6)]
```

```
223 4 (347914,6.02826e+6)
```

```
224 5 347914.27221116686
```

Script Runner plugin



Zadanie 5

Napisz skrypt do wykonywania w SR, który dla warstwy poligonów dla każdego rekordu poda ID, obwód poligonu i jego powierzchnię (pole ID jest pierwszym polem)

```

1  # Skrypt wyprowadza id obwody i powierzchnie poligonow dla danej warstwy
2  # Pole id jest pierwszym polem atrybutowym
3
4  from PyQt5.QtGui import *
5  from PyQt5.QtCore import *
6  from qgis.core import *
7  from qgis.utils import iface
8
9  war_name='Strz_woda'
10
11  mapa=iface.mapCanvas()
12  warstwy=mapa.layers()
13  brak=1
14  - for ww in warstwy:
15  -     if ww.name()==war_name:
16  -         pol_vec=ww
17  -         brak=0
18  - if brak==1:
19  -     QMessageBox.warning(iface.mainWindow(),"Informacja","Brak takiego pliku")
20  - else:
21  -     features=pol_vec.getFeatures()
22  -     poly=0
23  -     for feature in features:
24  -         gPoly=feature.geometry()
25  -         if gPoly.wkbType() == QgsWkbTypes.MultiPolygon:
26  -             attrs=feature.attributes()
27  -             print(attrs[0],gPoly.length(),gPoly.area())
28  -             poly=1
29  - if poly==0:
30  -     QMessageBox.warning(iface.mainWindow(),
31  -         "Informacja","To nie jest warstwa poligonow")

```

Krok 1: program, testowanie

Krok 2 - Budowa klasy według schematu

```
class Skrypt_u:  
    def __init__(self, iface):  
        self.iface=iface  
  
    def funkcja1(self, x):
```

```
fs=Skrypt_u(iface)
```

```
x=
```

```
fs.funkcja1(x)
```

```

1  from PyQt5.QtGui import *
2  from PyQt5.QtCore import *
3  from qgis.core import *
4  from qgis.utils import iface
5
6  class ObwodPole:
7      def __init__(self,iface):
8          self.iface=iface
9      def poly_info(self,warstwa):
10         war_name=warstwa
11
12         mapa=iface.mapCanvas()
13         warstwy=mapa.layers()
14         brak=1
15         for ww in warstwy:
16             if ww.name()==war_name:
17                 pol_vec=ww
18                 brak=0
19         if brak==1:
20             QMessageBox.warning(iface.mainWindow(),"Informacja","Brak takiego pliku")
21         else:
22             features=pol_vec.getFeatures()
23             poly=0
24             for feature in features:
25                 gPoly=feature.geometry()
26                 if gPoly.wkbType() == QgsWkbTypes.MultiPolygon:
27                     attrs=feature.attributes()
28                     print(attrs[0],gPoly.length(),gPoly.area())
29                     poly=1
30             if poly==0:
31                 QMessageBox.warning(iface.mainWindow(),
32                                     "Informacja","To nie jest warstwa poligonow")
33
34 op=ObwodPole(iface)
35
36 war='Strz_woda'
37 op.poly_info(war)

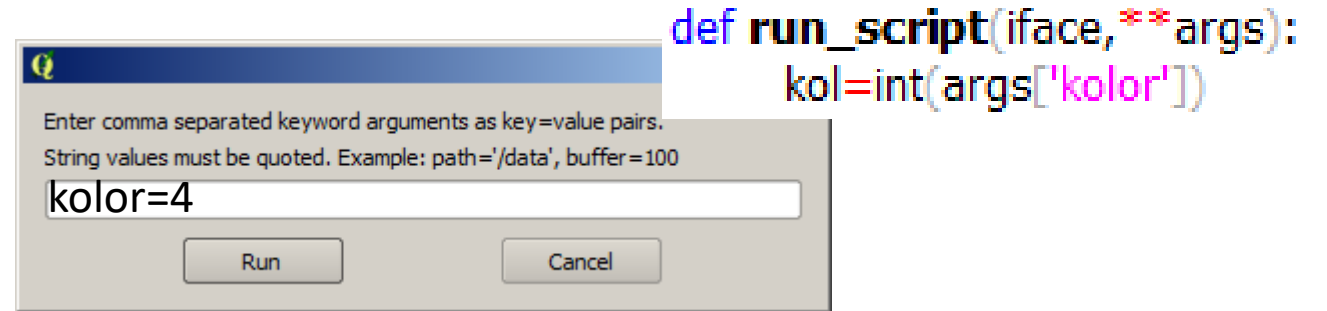
```

Krok 2: utworzenie klasy, w której nasz program jest funkcją

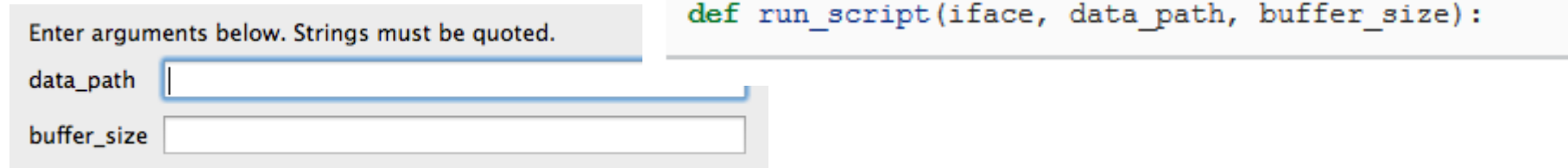
1. Aby wykorzystać **Script Runner** musi być utworzona funkcja:

– `def run_script(iface, **args):`

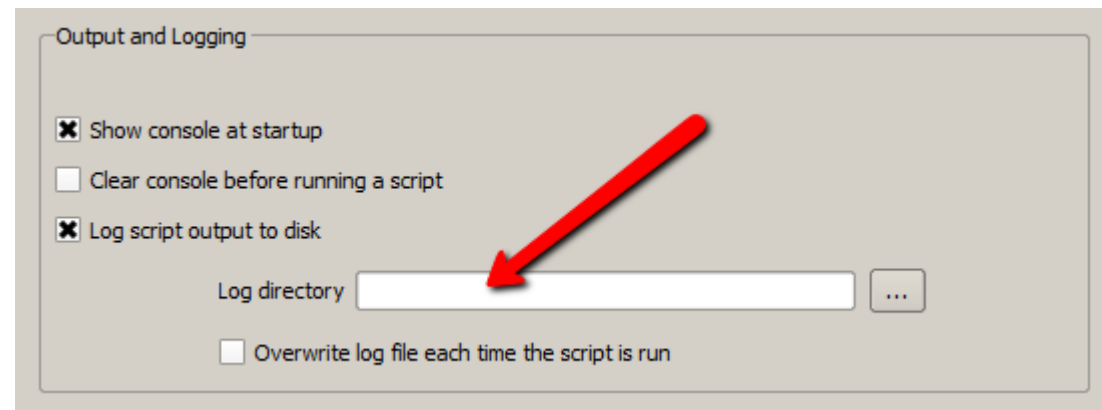
- 2a. Argumenty - Key words arg.



- 2b. Argumenty



3. Output może być dodatkowo logowany na dysk, wykorzystanie opcji preferencji Script Runer
Zapis do **scriptrunner.log**



```

1  from PyQt5.QtGui import *
2  from PyQt5.QtCore import *
3  from qgis.core import *
4  from qgis.utils import iface
5
6  class ObwodPole:
7      def __init__(self,iface):
8          self.iface=iface
9      def poly_info(self,warstwa):
10         war_name=warstwa
11
12         mapa=iface.mapCanvas()
13         warstwy=mapa.layers()
14         brak=1
15         for ww in warstwy:
16             if ww.name()==war_name:
17                 pol_vec=ww
18                 brak=0
19         if brak==1:
20             QMessageBox.warning(iface.mainWindow(),"Informacja","Brak takiego pliku")
21         else:
22             features=pol_vec.getFeatures()
23             poly=0
24             for feature in features:
25                 gPoly=feature.geometry()
26                 if gPoly.wkbType() == QgsWkbTypes.MultiPolygon:
27                     attrs=feature.attributes()
28                     print(attrs[0],gPoly.length(),gPoly.area())
29                     poly=1
30             if poly==0:
31                 QMessageBox.warning(iface.mainWindow(),
32                                     "Informacja","To nie jest warstwa poligonow")
33
34 -def run_script(iface,**args):
35     war=str(args['Warstwa'])
36     op=ObwodPole(iface)
37     op.poly_info(war)

```

Krok 3: utworzenie skryptu
Zawierającego funkcję

```
def run_script(iface,**args):
```

QMessageBox nie działa w SR
Użyć print

Doc string - info i instrukcja obsługi skryptu (na początku kodu)

```
''' Informacja o ID, obwodzie i polu poligonu.
```

```
Parametr : Warstwa='nazwa_warstwy' '''
```