# Język Python na potrzeby GIS

Katedra Geoinformacji, Fotogrametrii i Teledetekcji Środowiska



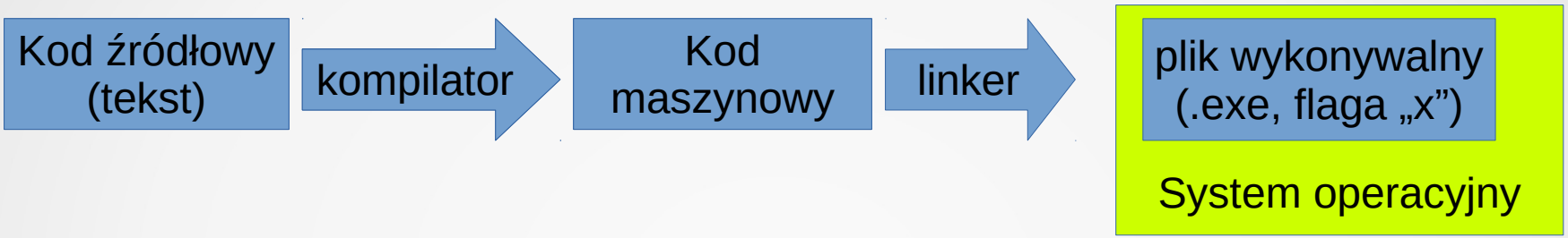
LibreOffice Impress
**http://libreoffice.org**

# Wspomaganie komputerowe GIS

- **Programy** – samodzielne aplikacje działające w określonym środowisku,
- **Skrypty** – pliki tekstowe wymagające do uruchomienia odpowiedniego interpretera,
- **Biblioteki** – zbiory funkcji nie uruchamiane bezpośrednio, ale możliwe do wykorzystania w programach lub skryptach,
- **Wtyczki** – pliki uruchamiane poprzez specyficzną aplikację dla której zostały stworzone, mogą mieć postać bibliotek lub skryptów.

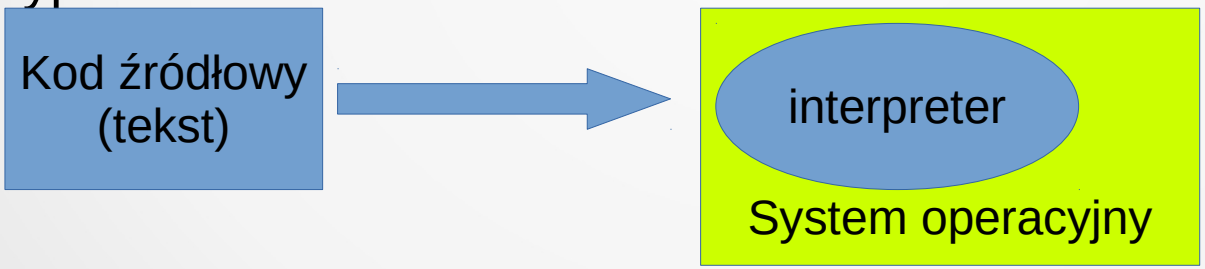
# Program vs skrypt

- Program:




  zalety – szybkość i spójność wykonania, pamięć

- Skrypt:




  zalety – wieloplatformowość, łatwe testowanie kodu

# Wtyczki python w GIS

- QGIS – http://qgis.org
  - program na licencji GPL,
  - wieloplatformowy: Windows, Linux, MacOS, BSD, Android
  - szybki rozwój w ostatnich latach,
  - coraz większa popularność w produkcji,
  - użycie biblioteki pyQT.
- ArcGIS – http://esri.com
  - licencja komercyjna, Windows,
  - standard produkcyjny,
  - wtyczki python, podobnie jak w QGIS.

# Środowisko uruchomienia

- QGIS 2.x – http://qgis.org, zawiera:
  - implementację Python i zintegrowana konsolę
  - bibliotekę pyQT do interfejsu graficznego
  - środowisko OSGeo4W

- Notepad++ – http://notepad-plus-plus.org,
  - zaawansowany edytor tekstu ASCII
  - podświetlanie składni
  - wtyczki, automatyczne uruchamianie skryptów

# Cechy języka Python

- stosunkowo krótka historia,
- wysoki poziom,
- uniwersalne zastosowanie,
- łatwość zrozumienia kodu,
- konserwatywna ilość kodu,
- programowanie strukturalne i obiektowe,
- wieloplatformowość,
- łatwy do nauczenia,
- popularność.

# Podstawy języka Python

- dokumentacja:

  https://docs.python.org/2/reference/
- komentarz:

  znak #
- każda wartość jest obiektem
- konsola Pythona – pojedyncze wyrażenia
- skrypt – połączenie wielu poleceń

# Systemy liczbowe

- dziesiętny

  cyfry 0, 1, 2, 3 … 9  rzędy wielkości: $10^0$ $10^1$ $10^2$ $10^3$

  $255 = 2 * 10^2 + 5 * 10^1 + 5 * 10^0 = 200 + 50 + 5$

- binarny/dwójkowy

  cyfry  0 i 1 – bit najmniejsza ilość informacji

  rzędy wielkości: $2^0$, $2^1$, $2^2$, $2^3$, …, $2^7=128$, $2^8=256$

  liczymy: 0,1,10,11,100,101,110,111,1000….11111111 (bajt, 255)

- szesnastkowy

  cyfry 0,1,2...9,A,B,C,D,E,F

  rzędy wielkości: $16^0=1$ (0x01), $16^1=16$ (0x10), $16^2=256$ (0x100)

  liczymy: 0,1,2,..., 9, A,…, F, 10, 11,…, 19, 1A, …, 1F, 20, …, FF (bajt)

# Podstawowe operatory

| operator | znak |
|---|---|
| przypisanie | = |
| matematyczne | + - * / ** % |
| porównania | == != > < <= >= |
| logiczne | and or not |
| przynależności | in not in |
| wyświetlanie wartości | print |

# Sekwencje ucieczki

- **** Backslash ()
- **\'** Single-quote (')
- **\"** Double-quote (")
- **\a** ASCII bell (BEL)
- **\b** ASCII backspace (BS)
- **\f** ASCII formfeed (FF)
- **\n** ASCII linefeed (LF)
- **\r** ASCII Carriage Return (CR)
- **\t** ASCII Horizontal Tab (TAB)
- **\u**xxxx Character with 16-bit hex value xxxx (Unicode only)
- **\U**xxxxxxxx Character with 32-bit hex value xxxxxxxx (Unicode only)
- **\v** ASCII vertical tab (VT)
- **\ooo** Character with octal value ooo
- **\xhh** Character with hex value hh

# Proste typy danych

- int – wartości całkowite

  **x = 3764**  # przypisanie obiektu 3764 do x

- str – ciągi znaków

  **nazwisko = „kowalski”**

- float – wartości zmiennoprzecinkowe

  **liczbapi = 3.1415**

  **wynik = 2.0**

- sprawdzenie typu zmiennej

  **type(zmienna)**

# Ciągi

- oznaczane cudzysłowami lub apostrofami
- niemutowalne, niemożliwe zmiany bezpośrednie
- mechanizm cięcia (*slicing*), pozwala na modyfikację

```
'Litwo ojczyzno moja'[6:14]      # 'ojczyzno'
nazwisko[:2]                     # 'Ko'
(nazwisko+' Jan')[-9:]           # 'alski Jan'
```

- ciągi jako obiekty

```
nazwisko.upper()          # 'KOWALSKI'
nazwisko.find('lsk')      # 4
```

# Typy złożone

- lista = **['wpis 0', 'wpis 1', 2, 'wpis 3', 2.0]**

  **lista[2] = 5** # lista = ['wpis 0', 'wpis 1', 5, 'wpis 3', 2.0]
- tupla = **('wpis 0', 'wpis 1', 2, 'wpis 3', 2.0)**

  **tupla[2:4]** # (2, 'wpis 3')
- zbior = **set(('wpis 0', 'wpis 1', 2, 'wpis 3', 2.0))**

  **print zbior** # set([2, 'wpis 3', 'wpis 1', 'wpis 0'])
- słownik = **{0: 'wpis 0', 1:'wpis 1', 'drugi':2, 'trzeci':'wpis 3'}**

  **print slownik[1]** # 'wpis 1'

  **print slownik['drugi']** # 2

  slownik['trzeci']=3 # {0: 'wpis 0', 1:'wpis 1', 'drugi':2, 'trzeci':3}

# Operatory logiczne

- sprawdzanie warunków

  10 == 10  (prawda),  10 != 10  (fałsz),  10 <=5 (fałsz)

- tabela prawdy and, or, not

```
1 and 1 = 1,  1 and 0 = 0,  0 and 1 = 0,  0 and 0 = 0
1 or 1 = 1,  1 or 0 = 1,  0 or 1 = 1,  0 or 0 = 0

1 xor 1 = 0,  1 xor 0 = 1,  0 xor 1 = 1,  0 xor 0 = 0
          not 1 = 0,  not 0 = 1
```

- łączenie warunków

```
10 == 10 or 10 >= 3      # prawda, jeden warunek spełniony
3 < 4  and 4 < 10        # prawda, oba warunki spełnione
3 < 4 < 10               # uproszczenie zapisu
```

# Skrypty i kontrola przepływu

- plik tekstowy z rozszerzeniem „.py”
- zbiór poleceń które chcemy wykonać

```
a = 3
b = 4
print 'Wynik : ', a+b
```

- wykonanie skryptu interpreterem

  python test.py

- bloki programu oznaczane są wcięciami
- dowolne nazwy zmiennych z wyjątkiem słów kluczowych

# Wyrażenia warunkowe

W zależności od wartości zmiennej możliwe są różne ścieżki wykonania programu, struktura **if … elif … else** :

```
print 'Czy chcesz kontynuowac (t/n)?'
k=raw_input()
if k == 't':
    print 'Kontynuuję program'
elif k == 'n':
    print 'Zatrzymuję program'
    exit()
else:
    print 'Zla opcja'
print 'Dalsza część programu'
```

# Pętla *while*

Wielokrotne wykonanie bloku dopóki spełniony jest warunek kontrolujący pętlę:

```
print 'Zgadnij cyfrę od 1 do 9'
while input() != 7:
    print 'Zła odpowiedz!'
    print 'Wracam na początek”
print 'Trafileś'
```

# Pętla *for*

Wykonanie bloku programu z góry określoną ilość razy

```
print 'Enumeracja liczb 0-99: ',
for i in range(100):
    print i,' ',
print '\nKoniec enumeracji.'
```

- funkcja range() generuje liczby w zakresie 1 - 100

# Kontrola pętli

- pominięcie jednego cyklu pętli

  **continue**
- całkowite wyjście z pętli

  **break**
- pusta instrukcja / nie rób nic

  **pass**

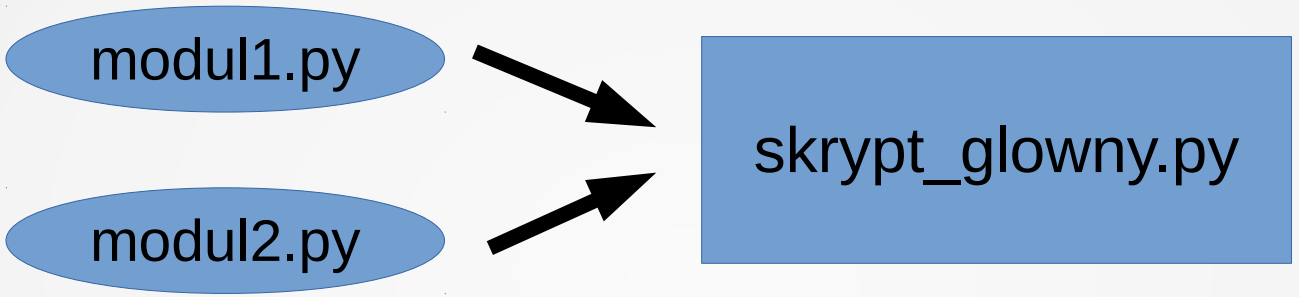
# Funkcje

Wyodrębnienie części programu z możliwością wywołania z dodatkowymi parametrami:

```
def dodaj(a,b): # definicja funkcji
      c=a+b
      return c   # zwraca wartość do programu


print dodaj(2,3)   # początek programu
wyn=dodaj          # funkcje można przypisać do zmiennej
print wyn(3,4)
```

# Moduły

Python pozwala na dzielenie programu na mniejsze pliki




skrypt główny będzie zaczynał się:

```
import modul1
import modul2
```

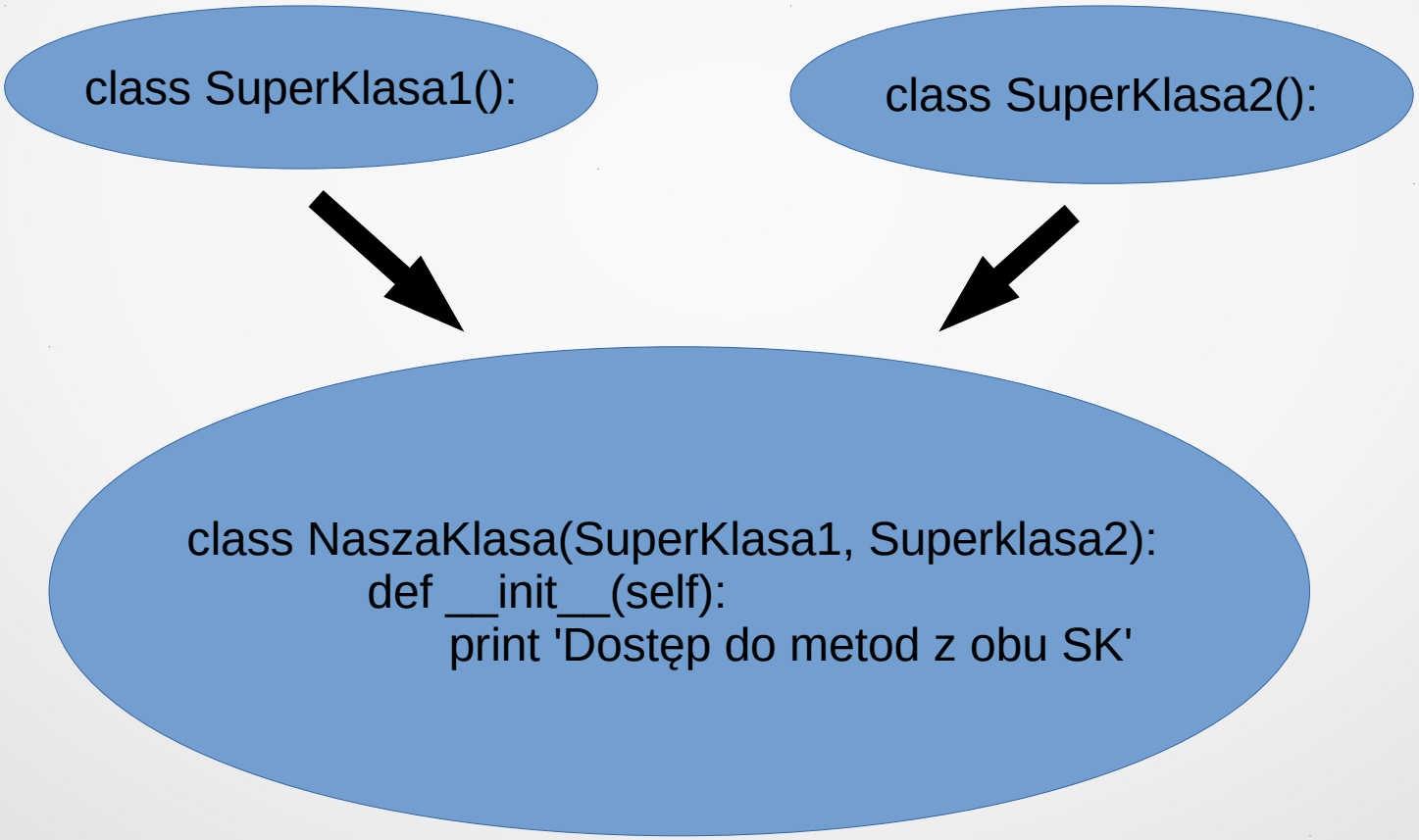
# Klasy i metody

- klasa jako kolejny poziom abstrakcji

```
class Kalk(object):
        def __init__(self, s='Inicjalizuje klase kalkulatora'):
                self.zaps = s   # przypisanie wartości do zmiennej lokalnej
                print self.zaps     # wyswietlenie napisu
        def dodaj(self, a, b)
                return a+b


nowyobiekt1=Kalk()                                  # początek programu
nowyobiekt2=Kalk('Tworze klase z parametrem')   # drugi obiekt
```

# Dziedziczenie klas

Najważniejsza cecha programowania obiektowego to możliwość korzystania z już istniejących klas.



# Dziedziczenie klas

Przykład dziedziczenia klasy Kalk:

```
class PodKalk(Kalk):
        def __init__(self):
            print 'Inicjalizuję podklasę'
        def odejmij(a, b):
             return a-b

nowypodkalk=PodKalk()
print nowypodkalk.odejmij(8 ,2)
print nowypodkalk.dodaj(3, 5)
```

# Obsługa wyjątków

W miejscach wrażliwych na nieprzewidziane błędy (np. przy otwieraniu pliku), możemy dodać test na ich wystąpienie używając struktury ***try… except…else***

```
try:
    plik=open('NazwaPliku.txt', w)
except:
    print('Błąd dostepu do pliku')
else:
    plik.read()
```

Bez tej struktury w przypadku błędu odczytu/zapisu program zakończył by działanie bezwarunkowo.

# Biblioteki standardowe

- możemy korzystać z wielu bibliotek standardowych, które zawierają często używane funkcje

  **import math, sys**

- zamiast całej biblioteki możemy importować wybrane klasy

  **from PyQt4 import ***

  **from PyQt4 import QtWidget**

- możemy sprawdzić jakie funkcje zostały zaimportowane

  **dir(math)**

# Funkcje rekurencyjne

Przykładowy problem sumowania liczb można rozwiązać iteracyjnie:

```
def liczsume(listaLiczb):
    suma = 0
    for liczba in listaLiczb:
        suma = suma + liczba
    return suma


print( liczsume([1,3,6,8,12]) )
```

# Funkcje rekurencyjne

Ten sam problem można rozwiązać rekurencyjnie ( ang. *recursive*):

```
def liczsume(listaLiczb):
    if len(listaLiczb) == 1:
        return listaLiczb[0]
    else:
        return listaLiczb[0] + liczsume(listaLiczb[1:])


print( liczsume([1,3,6,8,12]) )
```

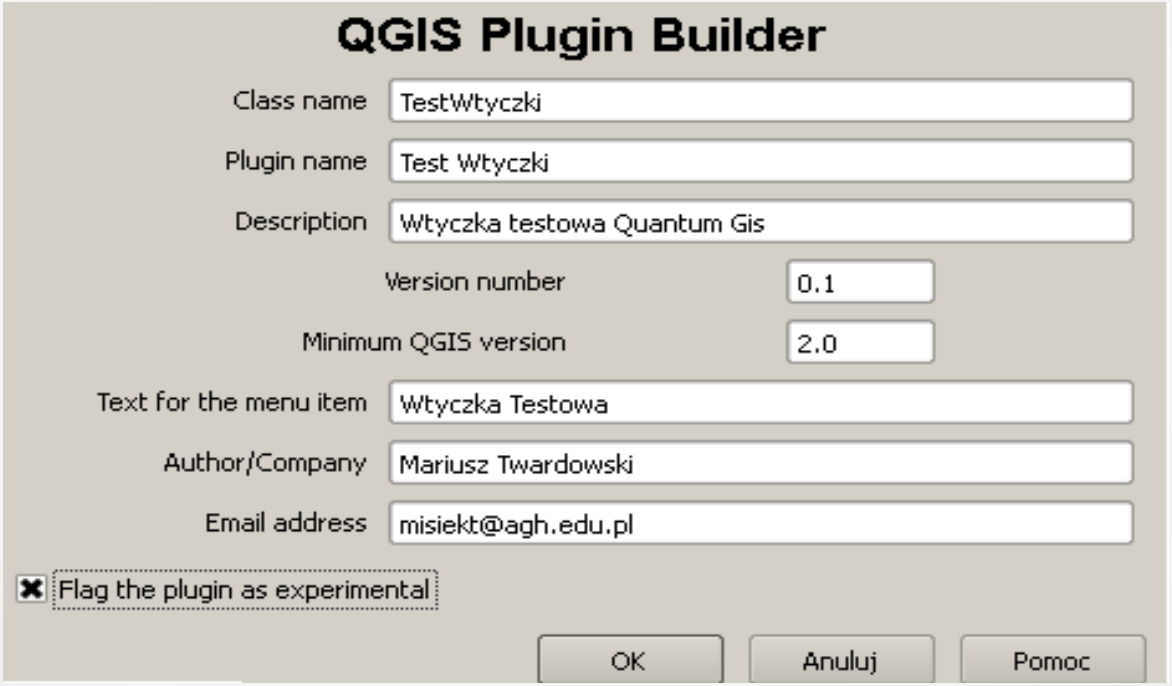
# Python w QGIS

- integracja Python i PyQt w instalacji
- środowisko uruchomienia OSGeo4W
- realizacja wtyczki dla QGis
- automatyczne generowanie kodu
- instalacja wtyczki w programie
- katalog wtyczek

  %USERPOFILE%/.qgis/python/plugins

# Generowanie bazy wtyczki

- zarządzanie wtyczkami w QGIS
- wtyczka Plugin Builder




- translatory **pyuic4** i **pyrcc4**
- wtyczka Plugin Reloader

# Zawartość plików wtyczki

- inicjalizacja w pliku **__init__.py**

```
def classFactory(iface):
    from testwtyczki import TestWtyczki
    return TestWtyczki(iface)
```

- plik klasy **testwtyczki.py**

```
import resources_rc
from testwtyczkidialog import TestWtyczkiDialog
class Testwtyczki:
    (...)
    self.dlg=TestWtyczkiDialog()
```

# Zawartość plików

- ikona wtyczki a plik **resources_rc.py**
- reprezentacja szesnastkowa obrazu PNG

**qt_resource_data = "**

**\x00\x00\x04\x0a**

**\x89**

**\x50\x4e\x47\x0d\x0a\x1a\x0a\x00\x00\x00\x0d\x49\x48\x44\x52\x00**

**\x00\x00\x17\x00\x00\x00\x18\x08\x06\x00\x00\x00\x11\x7c\x66\x75**

**\x00\x00\x00\x01\x73\x52\x47\x42\x00\xae\xce\x1c\xe9\x00\x00\x00**

**\x06\x62\x4b\x47\x44\x00\xff\x00\xff\x00\xff\xa0\ (....) „**

# Zawartość plików

- inicjalizacja klasy dialogu **testwtyczkidialog.py**

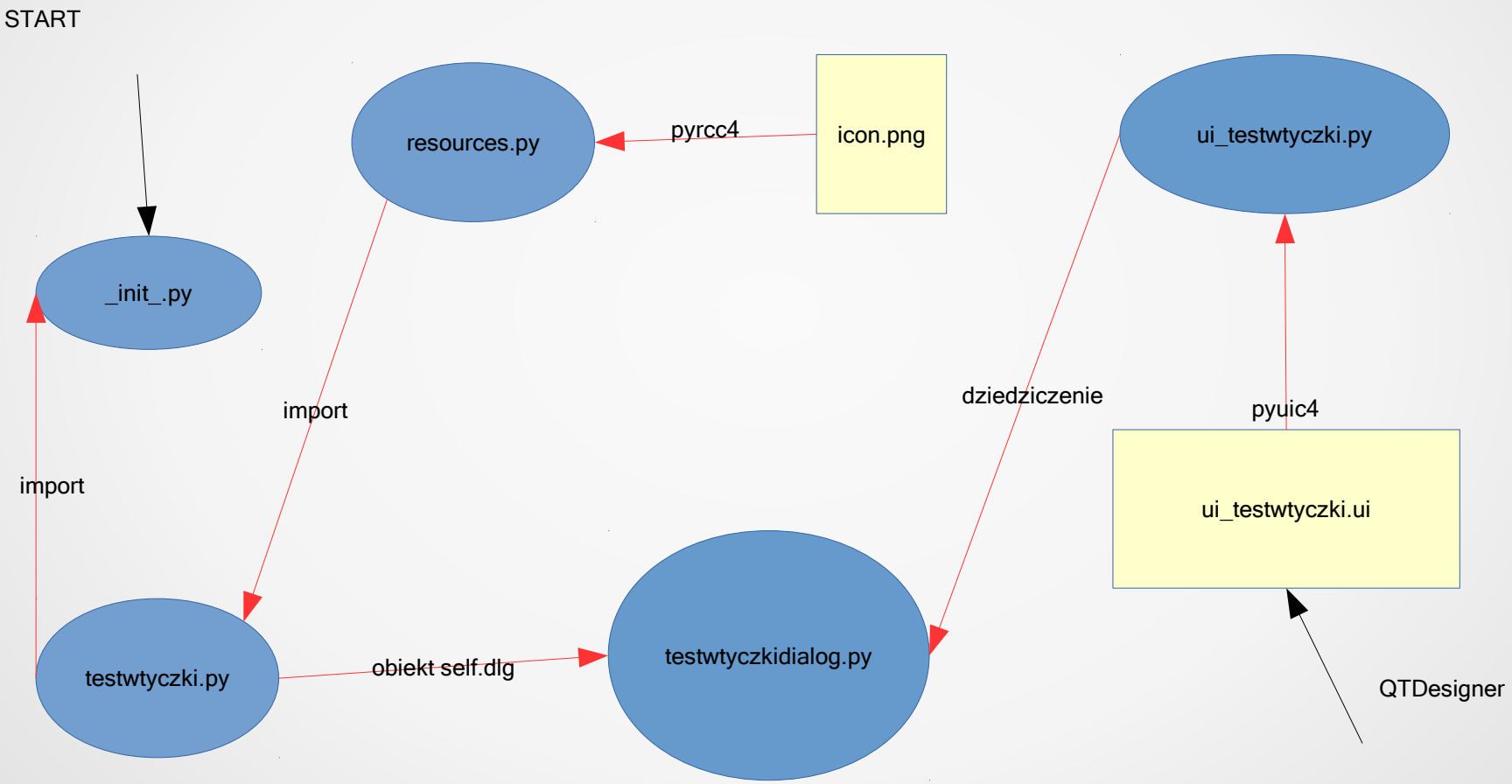
```
from ui_testwtyczki import Ui_TestWtyczki

class TestWtyczkiDialog(QtGui.QDialog, Ui_TestWtyczki):

        def __init__(self):

            self.setupUi(self)
```

# Zawartość plików

- **ui_testwtyczki.py ….**

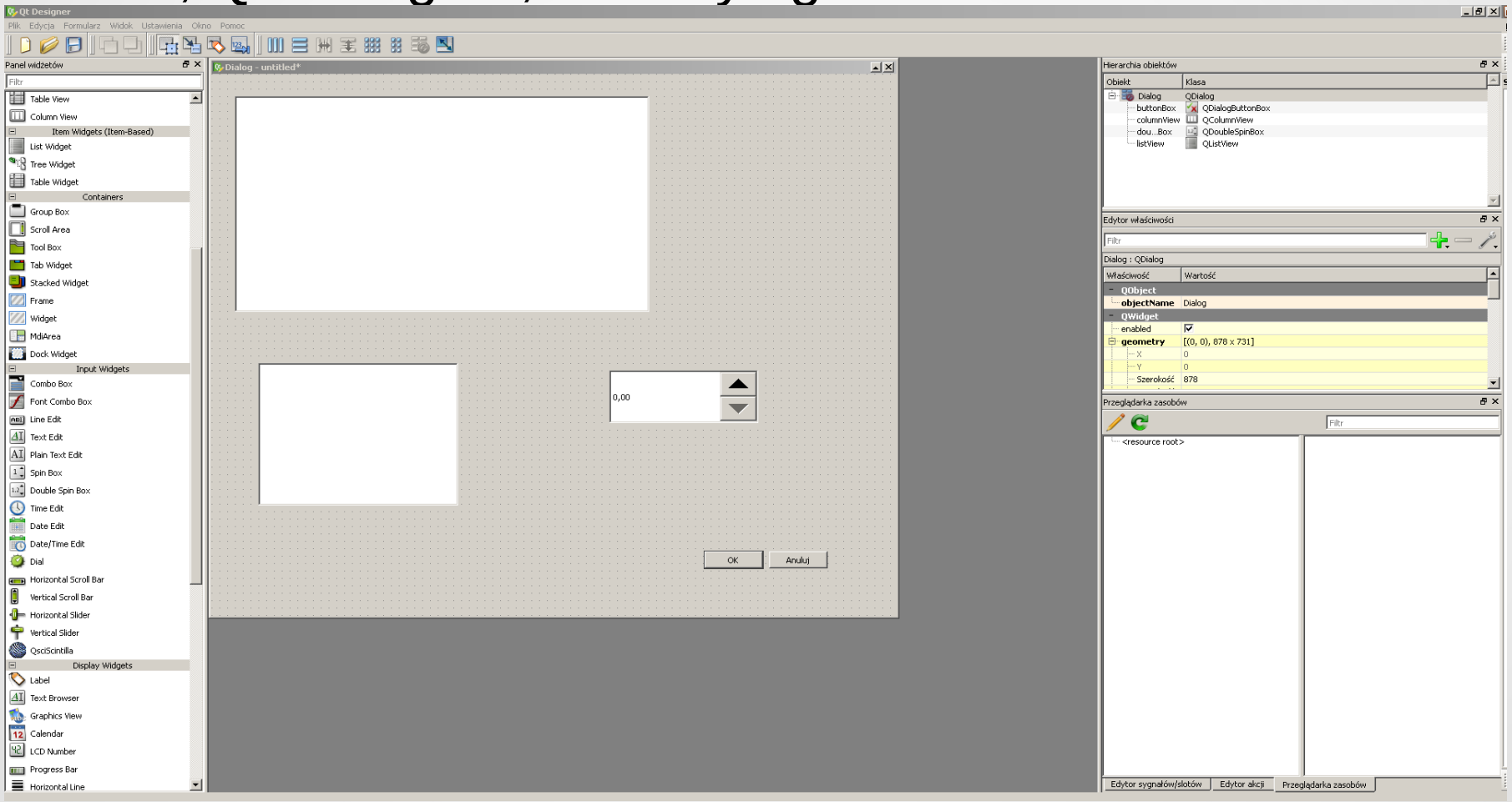
```
class Ui_TestWtyczki(object):
    def setupUi(self, TestWtyczki):
        TestWtyczki.setObjectName(_fromUtf8("TestWtyczki"))
        TestWtyczki.resize(600, 250)
        self.bOpenKUL = QtGui.QPushButton(TestWtyczki)
        self.bOpenKUL.setGeometry(QtCore.QRect(10, 10, 75, 23))
        self.bOpenKUL.setObjectName(_fromUtf8("bOpenKUL"))
        self.crdTable = QtGui.QTableWidget(TestWtyczki)
        self.crdTable.setGeometry(QtCore.QRect(0, 40, 601, 171))
        self.crdTable.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.crdTable.setAutoFillBackground(False)
        self.crdTable.setObjectName(_fromUtf8("crdTable"))
        self.crdTable.setColumnCount(5)
        self.crdTable.setRowCount(0)
        item = QtGui.QTableWidgetItem()
        self.crdTable.setHorizontalHeaderItem(0, item)
        item = QtGui.QTableWidgetItem()
        self.crdTable.setHorizontalHeaderItem(1, item)
        item = QtGui.QTableWidgetItem()
        self.crdTable.setHorizontalHeaderItem(2, item)
        item = QtGui.QTableWidgetItem()
        self.crdTable.setHorizontalHeaderItem(3, item)
        item = QtGui.QTableWidgetItem()
        self.crdTable.setHorizontalHeaderItem(4, item)
        self.bSave2000 = QtGui.QPushButton(TestWtyczki)
        self.bSave2000.setGeometry(QtCore.QRect(260, 220, 75, 23))
        self.bSave2000.setObjectName(_fromUtf8("bSave2000"))
        self.bCalc = QtGui.QPushButton(TestWtyczki)
        self.bCalc.setGeometry(QtCore.QRect(130, 220, 75, 23))
        self.bCalc.setObjectName(_fromUtf8("bCalc"))
        self.pathKUL = QtGui.QLineEdit(TestWtyczki)
        self.pathKUL.setGeometry(QtCore.QRect(90, 10, 511, 20))
        self.pathKUL.setReadOnly(True)
        self.pathKUL.setObjectName(_fromUtf8("pathKUL"))
        self.bGenSHP = QtGui.QPushButton(TestWtyczki)
        self.bGenSHP.setGeometry(QtCore.QRect(390, 220, 75, 23))
        self.bGenSHP.setObjectName(_fromUtf8("bGenSHP"))
```

# Schemat programu



# Interfejs wtyczki

- RAD, QTDesigner, widżety i generowanie kodu




- QTDesigner → plik .UI → pyuic4 → plik .PY

# Sygnały i sloty w PyQt4

- element interfejsu wysyła sygnał

  **clicked()  pushed()  released() ….**

- metoda connect() łączy sygnał ze slotem (metodą)

  **self.bPrzycisk.clicked.connect(self.naszaMetoda)**

- połączenie automatyczne **connectSlotsByName()**

  **on_bPrzycisk_clicked()**

  **on_bPrzycisk_pushed()**

  **on_bPrzycisk_released()**

# Funkcjonalność programu

- transformacja współrzędnych KUL do EPSG:2178

**xw = xbw + a1 + a3*x - a4*y + a5*(x*x - y*y) - 2*a6*x*y**

**yw = ybw + a2 + a3*y + a4*x + 2*a5*x*y + a6*(x*x – y*y)**

- współczynniki

**xbp=-3.044193761904 76×10^4 ybp= 2.890308785714 29×10^5**

**xbw= 5.546593135000 00×106 ybw= 7.428523595761 90×106**

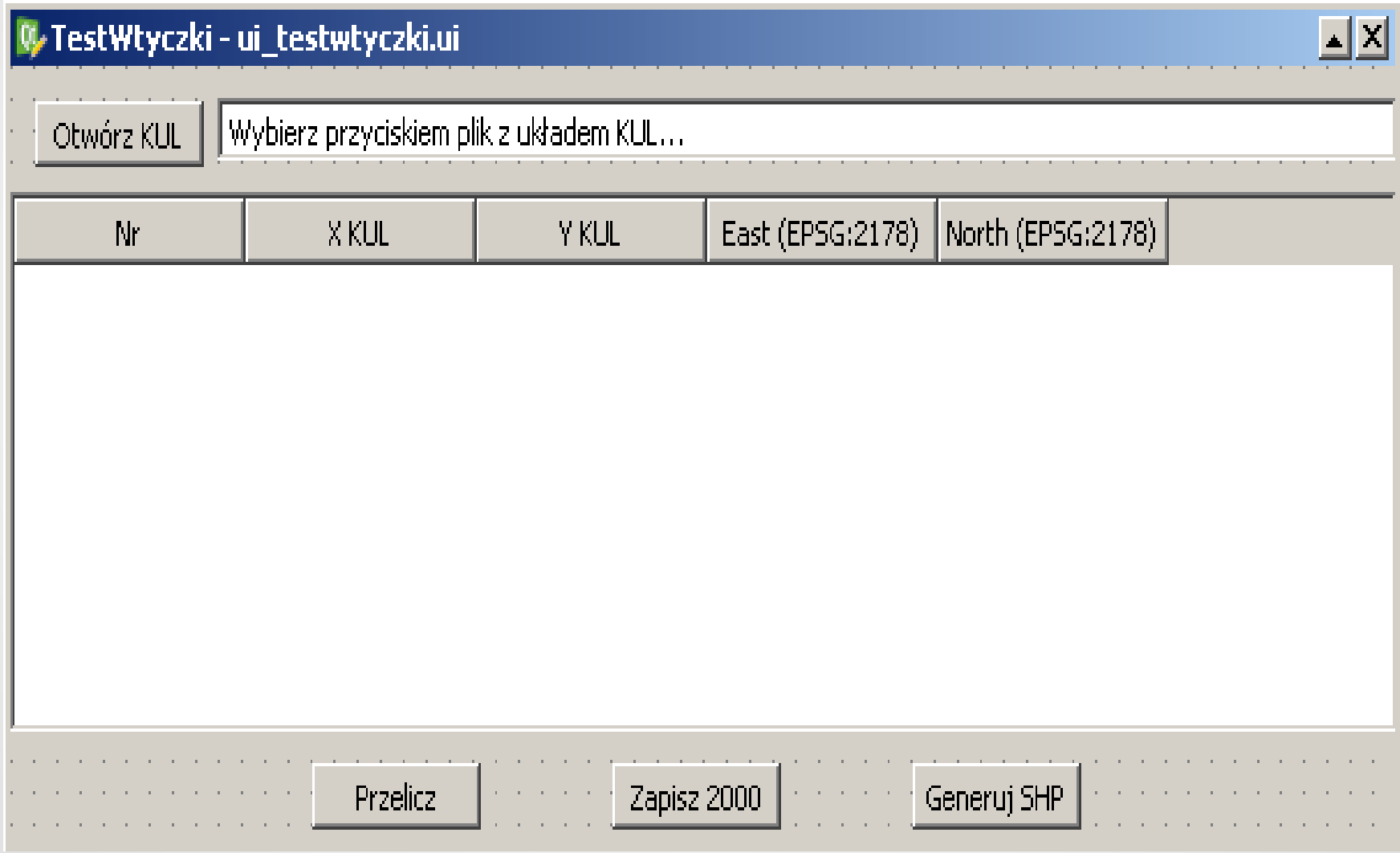
**x=x-xbp y=y-ybp**

**a1 = 2.736604507138 79×10^-22 a2 = 7.692156914323 81×10^-2**

**a3 = -9.991162253043 04×10^-1 a4 = 4.054786463616 01×10^-2**

**a5 = 6.775480432325 47×10^-11 a6 = 9.087247780980 26×10^-10**

# Interfejs programu



# Implementacja kodu

- otwacie pliku KUL - **on_bOpenKUL_released()**
- funkcja przeliczenia współrzędnych - **calcK2k()**
- obsługa przycisku przelicznia - **on_bCalc_released()**
- zapisanie pliku wyjściowego - **on_bSave2000()**
- tworzenie pliku graficznego SHP - **on_bGenSHP()**

# Implementacja kalkulatora PyQT4

- użycie QTDesigner do stworzenia interfejsu

  widżety QT:

  **Line Edit**

  **Push Button**

  **Label**


- implementacja metod dla działań

  **on_bDodaj_released()**, **on_bMinus_released()**....

- komentarze do kodu, wysyłanie przez formularz zaliczeń