Programowanie komputerów
Języki programowania - Python

Repetytorium podstaw programowania w Pythonie

Marzena Nowakowska

Wydział Zarządzania i Modelowania Komputerowego
Politechnika Świętokrzyska

Budynek C, p. 3.21

spimn@tu.kielce.pl

Informacje organizacyjne

1. Budynek C, piętro III, p. 3.21.
2. Zmiana grup możliwa na zasadzie wymiany 1 za 1
3. Zaliczenie przedmiotu: zdobycie co najmniej 50% punktów możliwych do uzyskania z **egzaminu/sprawdzianu** przeprowadzonego w formie testu.
   - **Terminy: por. informacja w ogłoszeniach**
   - **Obecność na wykładach uwzględniana w ocenie**
   - Liczba egzaminów/zaliczeń przedmiotu - zgodnie z Regulaminem Studiów.
     Dodatkowych (opcjonalnych) terminów nie będzie
4. Udział w wykładach jest konieczny ze względu na duży stopień trudności przedmiotu
5. Materiały do przedmiotu w Internecie (uzupełniane na bieżąco):

   https://staff.tu.kielce.pl/spimn/

   Z menu głównego wybrać pozycję: Materiały dydaktyczne
6. Po co jest ten przedmiot?

Po co jest ten przedmiot

- *Everybody in this country should learn how to program a computer ... because it teaches, how to think.*

  Steve Jobs (założyciel firmy Apple)

- Celem jest wykształcenie umiejętności myślenia komputacyjnego (*computational thinking*), które obejmuje myślenie algorytmiczne w rozwiązywaniu problemów oraz umiejętność programowania rozszerzone na wszystkie obszary działalności człowieka.

- Język programowania jest narzędziem do zapisu algorytmów, dzięki któremu można komunikować się z komputerem. Myślenie algorytmiczne jest postrzegane jako najlepsza strategia myślenia prowadząca do myślenia komputacyjnego.

Efekty uczenia się

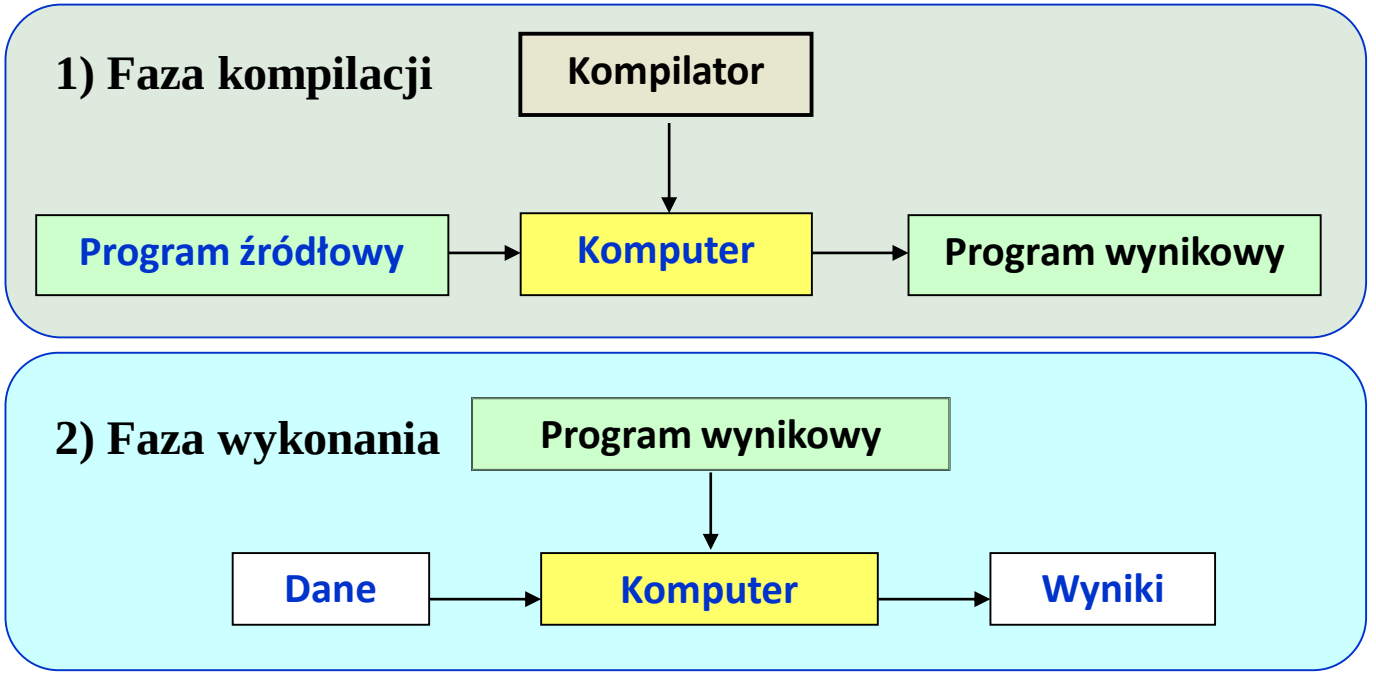
- W zakresie wiedzy
  - Struktury danych
  - Programowanie proceduralne z wykorzystaniem funkcji wbudowanych i własnych
  - Organizacja pracy programu z plikiem dyskowym
  - Podstawy programowania obiektowego
  - Budowa programu komputerowego do przetwarzania, analizy i wizualizacji danych, z wykorzystaniem bibliotek tematycznych i ich obiektów
- W zakresie umiejętności
  - Opracowanie programów w języku Python dla system Windows w paradygmacie imperatywnym i obiektowym do przetwarzania danych, w tym zapisanych w plikach dyskowych
  - Ocena przydatność narzędzi programowania do rozwiązywania zagadnień inżynieryjnych
- W zakresie kompetencji społecznych
  - Rozumienie potrzeby stałego uzupełniania wiedzy z obszaru nowoczesnych narzędzi (w tym AI) i idei informatyki

Hasła programowania – krótkie repetytorium

- **Język programowania** jest narzędziem obejmującym zbiór reguł, które określają poprawność zapisu sekwencji symboli i instrukcji w programie.
- **Program komputerowy** (oprogramowanie, aplikacja komputerowa) jest zbiorem poleceń, które zostały napisane przez programistę w wybranym języku programowania. Zawiera sekwencję symboli i ciąg instrukcji definiujących sposób przetwarzania danych (wykonywania obliczeń), kontrolowanie i obsługę urządzeń zewnętrznych przyłączonych do komputera.
- **Kod źródłowy** (program źródłowy) jest programem komputerowym (tekstem) napisanym w postaci zrozumiałej dla człowieka, zwykle z wykorzystaniem angielskich słów kluczowych. Do jego utworzenia wykorzystuje się edytor tekstowy.
- **Kod maszynowy** (kod lub program wynikowy, binarny, wykonywalny) jest programem komputerowym napisanym w postaci zrozumiałej dla komputera, prezentowanym w postaci ciągu zer i jedynek. Do jego utworzenia potrzebny jest translator.
- **Translator** jest programem komputerowym, który tłumaczy kod źródłowy na kod maszynowy. Translator jest niezbędnym narzędziem do tego aby program komputerowy się wykonał. Są dwa rodzaje translatorów: kompilatory i interpretery.

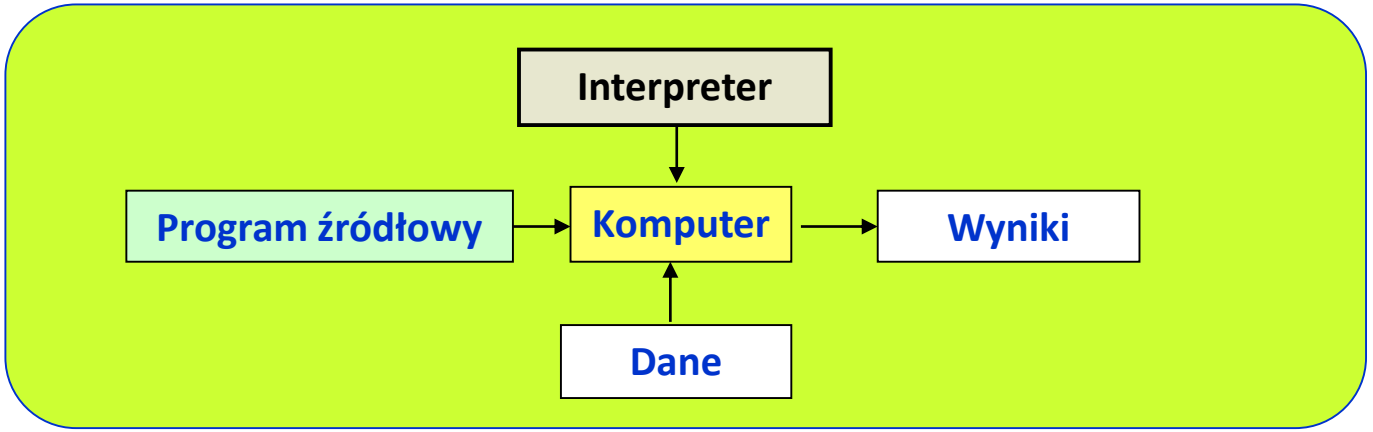
Rodzaje translatorów i ich działanie

- **Kompilator**



Przykład języka kompilowanego: C++

- **Interpreter**



Przykład języka interpretowanego: Python

Cechy języka Python

- Autor: Holender Guido van Rossum, serial angielski lat 69-74 „Monty Python's Flying Circus" (Latający Cyrk Monty Pythona).
- Zorientowany obiektowo skryptowy (interpretowany) język programowania ogólnego przeznaczenia typu *open source*, zoptymalizowany pod kątem jakości, wydajności, przenośności i integracji, dostępny za darmo.
- Struktury danych wysokiego poziomu; listy, krotki, słowniki, zbiory i szerokie spektrum ich przetwarzania.
- Dynamiczny system typów zwiększający wydajność i produktywność programistów oraz automatyczne zarządzanie pamięcią operacyjną.
- Grupowanie instrukcji przez wcięcia (**indetancja**).
- Bogata biblioteka standardowa oraz różne rozszerzenia.
- Bardzo duża i rosnąca społeczność skupiona wokół języka.

IDE dla Pythona

- IDE (*Integrated Development Environment*) - program lub zbiór programów (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.
- Funkcjonalność IDE:
  - edycja kodu źródłowego (edytor tekstowy),
  - translacja kodu źródłowego,
  - linting kodu (wskazywanie potencjalnych błędów i miejsc „podejrzanych”)
  - debugging (śledzenie),
  - linkowanie (w przypadku języków kompilowanych),
  - tworzenie zasobów programu (tzn. formatek/ekranów/okien dialogowych, menu, raportów, elementów graficznych jak ikony, obrazy),
  - tworzenie baz danych,
  - tworzenie komponentów i innych.
- Przykładowe IDE dla Pythona:
  - IDLE
  - Thony
  - Sublime Text 3
  - PyCharm
  - Jupyter
  - **Spyder** (Anaconda)

Spyder

Spyder (**https://www.spyder-ide.org/**) to IDE napisany w Pythonie i przeznaczony do obsługi programowania w tym języku. Podobnie jak Jupyter, wchodzi w skład pakietu Anaconda. Można jednak zainstalować Spydera bez konieczności instalacji Anacondy

Środowisko Spydera

- Menu główne i pasek narzędzi.
- Lewe okno. Sekcja edycji kodu (edytor tekstowy z analizą kodu i stylu w czasie rzeczywistym); tworzenie skryptów w Pythonie.
- Prawe górne okno. Zakładki dolne: Help, Variable Explorer (eksplorator zmiennych), Plots (wykresy utworzone z wykorzystaniem biblioteki mathplotlib), Files (pliki i foldery).
- Prawe dolne okno. Ipython (konsola) - interaktywne uruchamianie poleceń, przydatne: przywoływanie pomocy na wybrany temat (*help(słowo_kluczowe)*), *History* (historia poleceń wydanych w konsoli).

Każde okno posiada w prawym górny rogu menu kontekstow ☰ – wykaz opcji menu zależy od zawartości okna.

Tools/Preferences → Wybrać z lewego panelu **Working directory.**

Tryby pracy: interaktywny (konsola), skryptowy (edytor kodu).

Moduły w Pythonie

Opracowane i rozwijane przez społeczność Pythona (*open source*) oferta zasobów programistycznych jest zgrupowana tematycznie w postaci modułów (*module*) lub inaczej bibliotek (*library*). Wykaz modułów standardowych jest zestawiony w dokumentacji: http://www.python.org/doc/.

Moduł to plik tekstowy z rozszerzeniem *py*. Może zawierać definicje: zmiennych (różnych typów), funkcji oraz klas. Wykorzystanie zasobów określonego modułu jest możliwe po wykonaniu operacji importu:

import nazwa_modułu ⇔ *from nazwa_modułu import **

Przykład

```
import math
print(math.cos(30))
```

from nazwa_modułu import nazwa_zasobu1 [, nazwa_zasobu2]

Przykład

```
from math import sin, tan
print("sinus 30:", sin(30), "tangens 30:", tan(30))
```

Po zaimportowaniu modułu, można sprawdzić jego zawartość: ***dir(nazwa_modułu)***
Informacje o funkcji: ***help(nazwa_modułu.nazwa_funkcji)***

Standardowe typy danych Pythona - 1

Wszystkie dane w Pythonie są **obiektami**, co oznacza, że właściwie wszystkie zmienne (i stałe) posiadają metody (funkcje), które mogą wykonywać różne operacje na rzecz danego obiektu.

Python jest **językiem typowanym dynamicznie**, tzn. że typ danej zależy od wartości do niej przypisanej.

Liczba całkowita **int**

Stała: 12; -100

Zmienna: a = 12; a, b = 12, 100

Liczba zmiennoprzecinkowa (rzeczywista) **float**

Stała: 12.5; -100.78; 12.6e-3; 12.

Zmienna: aa = 12.5; aa, bb = 12.6, 13.

Liczba zespolona (część rzeczywista + urojona) **complex**

Stała: 2.6+6j

Zmienna: a = 2.6+6j

Typ logiczny **bool**. Jest podtypem typu całkowitego.

Stała: True; False

Zmienna: a = 12.5 > 18 (wartość a: False)

Łańcuch (napis) **str** (string) - ciąg znaków między cudzysłowami lub apostrofami. Wielolinijkowy napis umieszcza się między potrójnymi cudzysłowami.

Stała:

```
"Miał pod nosem czarny wąs"
 "Moja mama mówiła: 'Ostatnia umiera nadzieja'"
""" Wszystko to co mam
To ta nadzieja, że życie mnie poskleja"""
```

Zmienna: poczatek = "Miał pod nosem czarny wąs"

Standardowe typy danych Pythona - 2

Lista **list**. Służy do przechowywania uporządkowanej kolekcji obiektów (danych) a poszczególne elementy zmiennej tego typu **mogą** podlegać modyfikacji (dodawanie, podmiana i usuwanie elementów).

Stała: ['Python', 'WZiMK', 123, 98.45, 3.6e-3, 2.6+6j]
Zmienna: a = ['Python', 'WZiMK', 123, 98.45, 3.6e-3, 2.6+6j]

Krotka **tuple**. Służy do przechowywania uporządkowanej kolekcji obiektów (danych) a poszczególne elementy zmiennej tego typu **nie mogą** podlegać modyfikacji (brak możliwości edycji „w miejscu").

Stała: ('Python', 'WZiMK', 123, 98.45, 3.6e-3, 2.6+6j)
Zmienna: a = ('Python', 'WZiMK', 123, 98.45, 3.6e-3, 2.6+6j)

Zbiór **set**. Nieuporządkowana kolekcja prostych obiektów. Można je kojarzyć ze zbiorami w matematyce.

Cechą wyróżniającą zbiór jest unikalność elementów, to znaczy, że elementy w zbiorze nie mogą się powtarzać

Stała: {'bulgaria', 'mexico', 'poland', 'usa'}
Zmienna: VMWC2022 = {'bulgaria', 'mexico', 'poland', 'usa'}

Słownik **dict**. Tablica asocjacyjna, w której klucz jest kojarzony z przypisaną mu wartością, przy czym klucze muszą być unikalne. Słownik zapisuje się w postaci par *klucz*: *wartość*, rozdzielonych przecinkami.

Stała: {"klasa_a": 4.5, "klasa_b": 4, "klasa_c": 3.5, "klasa_d": 4.5, "klasa_e": 5 }
Zmienna: sl={"klasa_a": 4.5, "klasa_b": 4, "klasa_c": 3.5, "klasa_d": 4.5, "klasa_e": 5 }
osoba={"imię": "Dawid", "nazwisko": "Podsiadło", "rok_ur": 1993}
roma={1: "I", 2:"II", 3:"III", 4:"IV", 5:"V", 6:"VI", 7:"VII"}

Wybrane operatory

Operatory służą do definiowania operacji matematycznych, logicznych i symbolicznych. Kombinację wartości, zmiennych, operatorów i wywołań funkcji nazywa się **wyrażeniem**.

O kolejności obliczeń w wyrażeniu decyduje **priorytet** (ważność) i **łączność** (asocjatywność) operatora.

Wykaz operatorów jest zestawiony w pliku ***NotatkiDlaStudentów.docx***.

- Operator **warunkowy** (trójargumentowy):

  wynik_gdy_prawda* if *warunek* else *wynik_gdy_fałsz

  Przykład: maxNapis = adres if len(adres) >= len(telefon) else telefon
- Operatory **arytmetyczne**:

  +, -, *, /, % (modulo), // (dzielenie całkowite), **
- Operatory **porównania**: **<, <=, >, >=, !=, ==** (dwa znaki = obok siebie)

  a* < *b* < *c jest równoważny ***a* < *b* and *b* < *c***
  a* >= *b* > *c jest równoważny ***a* >= *b* and *b* > *c***
  a* ==*b* == *c jest równoważny ***a* == *b* and *b* == *c***

  Przykład: 5 < 6 < -1 → False
- Operator **przypisania** – znak **=**

  Stosowany w najprostszej instrukcji języka – przypisania

Kompendium języka Python

- Program w Pythonie jest plikiem tekstowym ze standardowym rozszerzeniem *py*.
- Bloki instrukcji w Pythonie wyróżniane są za pomocą odpowiedniego wcięcia (tabulatora) - **indetancja**. Edytor Spydera wprowadza indetancję w trakcie pisania skryptu. Brak lub nieprawidłowa indetancja są sygnalizowane jako błąd składni.
- Nazwy zmiennych, klas ani funkcji nie mogą być nazwami zastrzeżonymi. Uzyskanie wykazu w trybie interaktywnym: (1) wywołać w oknie konsoli funkcję help: *help()*, (2) w trybie helpa wywołać: *keywords* . Zamknięcie trybu helpa: *quit*
- Nazwa zawiera litery alfabetu łacińskiego, cyfry, znaki podkreślenia (podkreślnik występujący na początku ma specjalne znaczenie). Nie może zaczynać się od cyfry. Python rozróżnia wielkość liter w nazwie. Stąd np. identyfikatory: *ident* oraz *Ident* odnoszą się do dwóch różnych elementów.

Instrukcja przypisania
Instrukcja warunkowa
Instrukcje pętli:
- pętla *for*
- pętla *while*

Instrukcja wiążąca *with*

Wszystkie dane w Pythonie są obiektami

Znak kontynuacji linii (instrukcji): \

Przerwanie wykonywania programu (instrukcji): Ctrl+C

Instrukcja przypisania

- Przypisanie proste: ***nazwa = wyrażenie***

  Przykład: c = [1, 2, 3]; b = c;

- Przypisanie złożone: ***nazwa*** *operator* **= *wyrażenie***

  Wartości elementu *operator*: **+, -, *, /, %, //, **, &, |, ^, >>, <<**

  Przykład: a+ = b; c = [1, 2, 3]; c += [100]; c += [1000, 2000, 3000]

- Przypisane wielokrotne ***z1 = z2 = z3 = wyrażenie***

  z1, z2, z3 = wyrażenie

  z1, z2, z3 = wyrażenie1, wyrażenie2, wyrażenie3

  Przykład:

  a1 = a2 = a3 = 12**2

  c1 = c2 = c3 = 34//5

  e1, e2, e3 = 15, "ala", [1000, 2000, 3000]

- Sekwencje w przypisaniu

  z1, z2 = ['a', 'b'] → z1:'a', z2: 'b'

  z1, z2 = ('a', 'b') → z1:'a', z2: 'b'

  z1, z2 = "ab" → z1:'a', z2: 'b'

Komunikacja z użytkownikiem: operacja wejścia

Funkcja wbudowana do wprowadzania danych ze standardowego strumienia wejścia (klawiatury):

input ('*prompt*')

gdzie *prompt* jest treścią zachęty (może nie występować). Funkcja zwraca wprowadzony z klawiatury ciąg znaków (łańcuch).

Zazwyczaj instrukcja wprowadzania danych wykorzystywana jest w postaci:

zmienna = input ('*prompt*')

która przypisuje do podanej zmiennej wprowadzony z klawiatury **tekst**.

Aby wprowadzony ciąg znaków był dalej wykorzystywany jako liczba należy zastosować jedną z funkcji konwersji typów: *int* - dla liczby całkowitej, *long* – dla dużej liczby całkowitej, *float* – dla liczby rzeczywistej oraz *complex* – dla liczby zespolonej.

Przykład

- *zmienna = input ('Podaj liczbę całkowitą: '); l1 = int(zmienna); l2 = int(input("Podaj druga liczbę całkowitą: "))*
- *zm = input ('Podaj liczbę rzeczywistą: '); r1 = float(zm); r2 = float(input("Podaj drugą liczbę rzeczywistą: "))*
- *zmc = input ('Podaj liczbę zespoloną: '); cx1 = complex(zmc); cx2 = complex(input("Podaj drugą liczbę zespoloną: "))*

  # format wprowadzanej danej: *re*{+/-}*im*j, np. 2+5j, -5-7j

Komunikacja z użytkownikiem: operacja wyjścia

Funkcja wbudowana do wyprowadzania wyników do standardowego strumienia wyjścia (ekran): **print (*lista-wyjścia*)**

gdzie *lista-wyjścia* zawiera rozdzielone przecinkami elementy, których wartości mają być zaprezentowane.

Funkcja *print* ma opcjonalny parametr *end*, który definiuje, jaki znak ma być dodany na końcu wyprowadzanego wiersza. Domyślnie tym znakiem jest znak nowego wiersza ('\n'). Aby go nie wyprowadzać, należy nadać parametrowi wartość np. pustego tekstu → dwa apostrofy obok siebie ('').

Wyprowadzane wyniki można zorganizować poprzez operację formatowania napisów:

print ("*łańcuch formatujący*" % (*lista elementów oddzielonych przecinkami*))

Dla każdego elementu w liście wyników należy umieścić w łańcuchu formatującym znak % poprzedzony literą definiującą wyprowadzaną wartość, zgodnie z poniższym wykazem:

s – gdy wyprowadzany jest ciąg znaków (zamienia dowolny obiekt Pythona na napis)
c – gdy wyprowadzany jest jeden znak
i – gdy wyprowadzana jest liczba całkowita
f – gdy wyprowadzana jest liczba rzeczywista
e – gdy wyprowadzana jest liczba w postaci wykładniczej

Przykład

```
print("lista: %s\nliczba całkowita: %i\nliczba rzeczywista: %+8.2f\nliczba zespolona: %s"
%([1,2,3], -100, -11.11, 5*math.sin(30)-6j)) # nalezy przyłączyć pełen moduł math
```

Instrukcja warunkowa - 1

```
if warunek1:
    instrukcja11
    …
```

```
if warunek: instrukcja
```

Przykład

```
print ("Badanie czy liczba dodatnia")
x= float(input("Podaj liczbę:"))
if x>0:
   print ("Liczba dodatnia")
```

```
if warunek1:
    Instrukcja11
    …
else:
    instrukcja21
    …
```

Przykład

```
print("Zabawa w zgadywankę\n")
liczba = 23
dana = int(input('Podaj liczbę całkowitą: '))
if dana == liczba:
   print('Gartulacje, liczba odgadnięta.')
   print('(ale niestety nie ma żadnej nagrody!)')
else:
   print('Nie, liczba ma inną wartość')
print('\nKoniec instrukcji warunkowej')
```

Instrukcja warunkowa - 2

```
if warunek1:
    instrukcja11
    …
elif warunek2:
    instrukcja21
    …
else:
    instrukcja31
    …
```

Przykład

```
print ("Prosty kalkulator")
x= float(input("Podaj pierwszą liczbę x: "))
y= float(input("Podaj drugą liczbę y: "))
print("Wybierz rodzaj operacji arytmetycznej")
oper = int(input("\t1 - dodawania\n\t2 - odejmowania\n\t3 - mnożenie\n\t4 - dzielenie\n\t:"))
if oper == 1:
    print ("Wynik dodawania: ", x+y)
elif oper == 2:
    print ("Wynik odejmowania: ", x-y)
elif oper == 3:
    print ("Wynik mnożenia: ", x*y)
elif oper == 4:
    print ("Wynik dzielenia: ", x/y)
else:
    print ("Wprowadzony znak nie odpowiada żadnej zaproponowanej operacji arytmetycznej")
```