Informatyka
Wykład 2

Plan

- M-pliki
  - Skrypty
  - Funkcje
  - Skrypty vs funkcje
- Instrukcje sterujące
  - Instrukcja warunkowa **if**
  - Instrukcja wyboru **switch**
  - Pętle: **for**, **while**
- Zmienna liczba argumetnów funkcji.
- Obsługa błędów.

M-pliki

- Pliki tekstowe ASCII - *.m

- Dwa rodzaje m-plików:
  - M-pliki skryptowe
  - M-pliki funkcyjne

- Z M-plików korzystamy przy wsadowym trybie pracy z MATLABEM

M-pliki skryptowe – *skrypty (1)*

- Sekwencja komend wykonywanych jedna po drugiej

```
%przykladowy skrypt
%
clear all          %wyczyszczenie calej przestrzeni roboczej
clc                %wyczyszczenie ekranu
close all          %zamkniecie wszystkich okien wykresów

                   %na pkt z aktywności
N=100;             %
x = randn(1,N);    %
hist(x);           %
av=mean(x);        %
```

M-pliki skryptowe – *skrypty (2)*

- Skrypty działają w głównej przestrzeni roboczej MATLABA
- W trakcie wykonywania zmienne utworzone w skrypcie pojawiają się w przestrzeni roboczej
- Pojawia się problem konfliktu nazw zmiennych
  - Nadpisanie zmiennych
  - Wykorzystanie zmiennych z błędnymi danymi

Plus stosowania skryptów (w odniesieniu do interaktywnej pracy w linii poleceń)

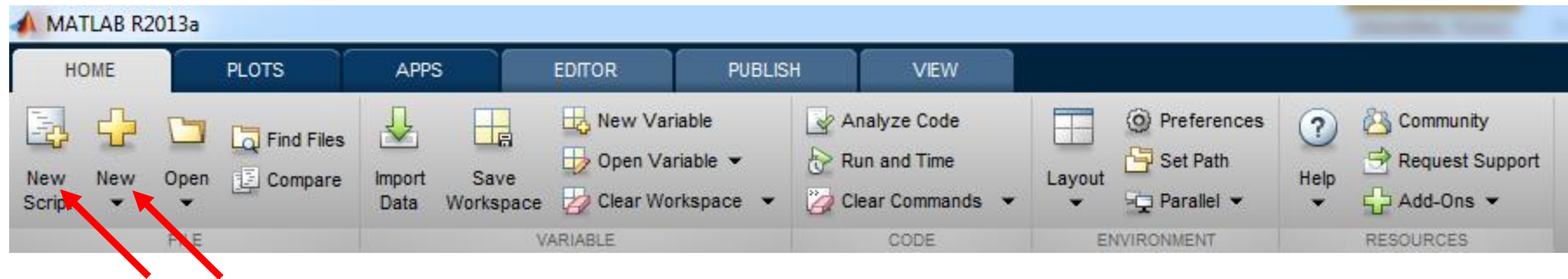
- Łatwiejsze wczytywanie dużej ilości danych
- Lepsza kontrola wprowadzanych komend
- Możliwość umieszczania komentarzy
- Zapis przebiegu obliczeń

WAŻNE!

- Przy budowie większych aplikacji skrypt to tylko etap wstępny tworzenia funkcji.

Tworzenie i uruchamianie skryptów

- Tworzenie skryptu:



- Uruchomienie skryptu:

  >> nazwa_skryptu % jeżeli skrypt znajduje się w folderze roboczym lub na ścieżce dostępu

M-pliki funkcyjne – funkcje (1)

```
function [ arg_wyjsciowe ] = nazwaFunkcji( arg_wejsciowe )
%Krotki opis funkcji
%Pelny opis

polecenia % ciało funkcji
end
```

Nazwa M-pliku powinna być taka sama jak nazwa funkcji

- Funkcja posiadają własną, lokalną przestrzeń roboczą
  - Funkcja działa na zmiennych lokalnych i globalnych
  - Funkcja może przyjmować argumenty wejściowe oraz zwracać argumenty wyjściowe
  - Zmienne utworzone w funkcji, są kasowane, jeżeli nie zostaną zwrócone

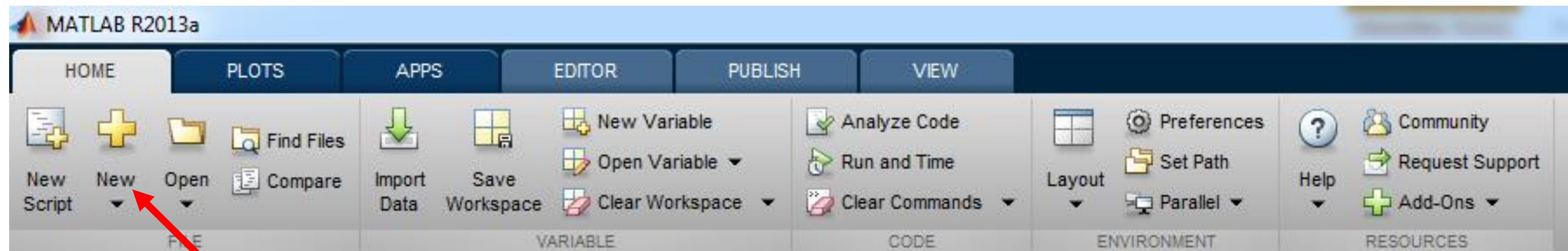
M-pliki funkcyjne – funkcje (2)

```
function [x,srednia]=losowyWektor(N) %przykladowa funkcja
%SKLADNIA
%[x,srednia]=losowyWektor(dlugoscWektora)
%DZIALANIE
%Funkcja losuje itd..


x = randn(1,N); % wygenerowanie wektora liczb z rozkladu normalnego
hist(x);              %wyrysowanie histogramu
srednia=mean(x);% wyliczenie sredniej, zmienna „srednia” będzie zwrocona
```

Tworzenie i uruchamianie funkcji

- Tworzenie funkcji



Z menu wybrać „New Function”

- Uruchamianie

```
>> [x,srednia]=losowyWektor(N)
>> [x,srednia]=losowyWektor(100)
>> [~,srednia]=losowyWektor(N)
```

- Help funkcji/skryptu

```
>>help losowyWektor
```

Funkcja główna i subfunkcje

- M-plik może zawierać definicje, więcej niż jednej funkcji
  - Pierwsza funkcja – funkcja główna
  - Kolejnej funkcje – subfunkcje
- Subfunkcje posiadają własną, lokalną przestrzeń roboczą
  - Zmienne muszą być przekazywane za pomocą argumentów
  - Mogą być deklarowane zmienne globalne (Nie zalecane!)
- Subfunkcje są dostępne tylko dla funkcji zdefiniowanych tym samym M-pliku

Funkcja główna i subfunkcje – przykład (1)

```
function [ pole, obwod ] = trojkatParametry( a, b, c, ha)

pole=poleTrojkata(a, ha) ;        %obliczanie pola i obwodu za pomoca funkcji uzytkownika
obwod=obwodTrojkata(a, b, c);
end

function [pole] = poleTrojkata (a, h) %subfunkcja otrzymuje zmienne przez argumenty wejsciowe
pole = 1/2 * a *h;
end

function [ obwod ] = obwodTrojkata(a, b, c)
obwod = a+b+c;
end
```

M-plik musi nazywać się ***trojkatParametry***

Funkcja główna i subfunkcje – przykład (2)

- Wywolanie funkcji *trojkatParametry*

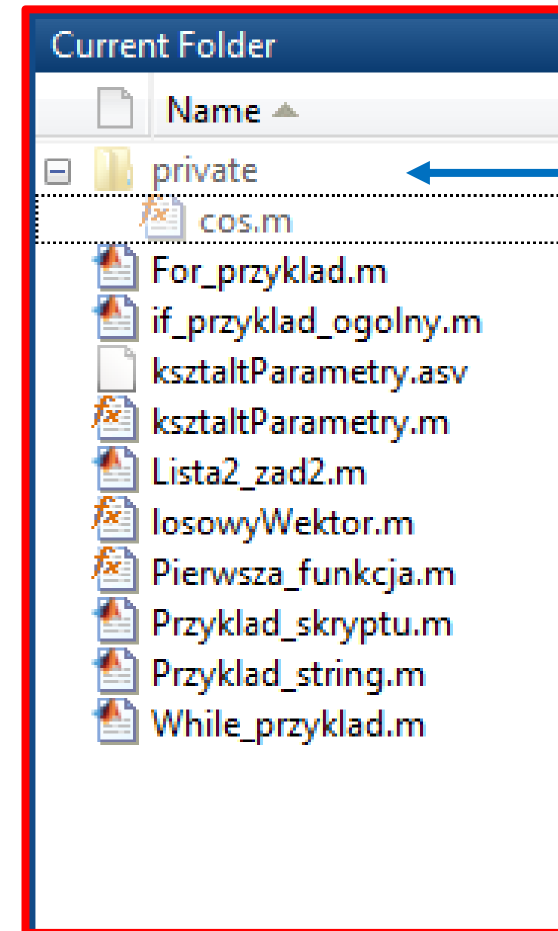
```
>> [p,o] = trojkatParametry(3,4,5,4) % uruchamiamy funkcje tak żeby zwrocila obie zmienne wyjsciowe
p =
     6                                %funkcja trojkatParametry wylicza pole za pomocą subfunkcji PoleTrojkata
o =
    12

>> poleTrojkata(3,4)                  %subfunkcja PoleTrojkata NIE JEST dostepna z linii polecen
Undefined function ‚poleTrojkata' for input arguments of type 'double'.

>>
```

Funkcje prywatne

- *Funkcje prywatne* – funkcje znajdujące się w podfolderze „private” wybranego folderu roboczego
- *Funkcje prywatne* są dostępne tylko dla funkcji znajdujących się w folderze macierzystym danego folderu „private”



Folder roboczy i ścieżka przeszukiwania

HOME | PLOTS | APPS

New Script | New | Open | Find Files | Compare | Import Data | Save Workspace | New Variable | Open Variable | Clear Workspace | Analyze Code | Run and Time | Clear Commands | Layout | Preferences | Set Path | Parallel | Help | Community | Request Support | Add-Ons

FILE | VARIABLE | CODE | ENVIRONMENT | RESOURCES

C: ▸ Users ▸ Bob ▸ Documents ▸ MATLAB ▸

Current Folder

Name

- Inf_2012_2013
- Inf_2013_2014
- Inf_2014_2015
- smietnik
- SPD
- cos.fig
- cos.mat
- dane.mat
- diary
- dist1_DIST.txt
- dist2_DIST.txt
- dist3_DIST.txt
- dist4_DIST.txt
- dist5_DIST.txt
- dodawanie.m
- dopasuj_wielomian.m
- first_script.asv
- first_script.m
- GUI_try.asv
- GUI_try.fig
- GUI_try.m

Folder roboczy

Editor - C:\Users\Bob\Documents\MATLAB\Inf_2014_2015\

Lista2_zad2.m | Przyklad_string.m | losowyWektor.

```
1  function [x,srednia]=losowyWektor(N
2  %SKLADNIA
```

Ustawianie ścieżki przeszukiwania

Set Path

All changes take effect immediately.

Add Folder... | Add with Subfolders... | Move to Top | Move Up | Move Down | Move to Bottom | Remove

MATLAB search path:

- C:\Users\Bob\Documents\MATLAB
- C:\Users\Bob\Documents\MATLAB\Inf_2014_2015
- C:\Users\Bob\Documents\MATLAB\smietnik
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\general
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\ops
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\elmat
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\randfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\elfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\specfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\matfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\datafun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\polyfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\funfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\sparfun
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\scribe
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\graph2d
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\graph3d
- C:\Program Files\MATLAB\R2013a\toolbox\matlab\specgraph

Save | Close | Revert | Default | Help

Dostępność funkcji

- Wbudowane funkcje MATLABA (np. *size*, *length*, *reshape*, *cos*, *sin*, *log*, itd. …)
- Funkcje użytkownika znajdujące się w plikach **w folderze roboczym**
- Funkcje użytkownika w folderach umieszczonych **na ścieżce przeszukiwania**
- Wyjątki:
    - Subfunkcje – dostępne dla funkcji zdefiniowanych w ramach jednego M-pliku
    - Funkcje prywatne – dostępne dla funkcji w folderze macierzystym podfolderu „private”
    - Funkcje zagnieżdżone (patrz *Mrozek & Mrozek, 2010, s. 78-79*)

Priorytet wywoływania funkcji

- W przypadku, gdy nazwy funkcji lub zmiennych są identyczne MATLAB sprawdza:

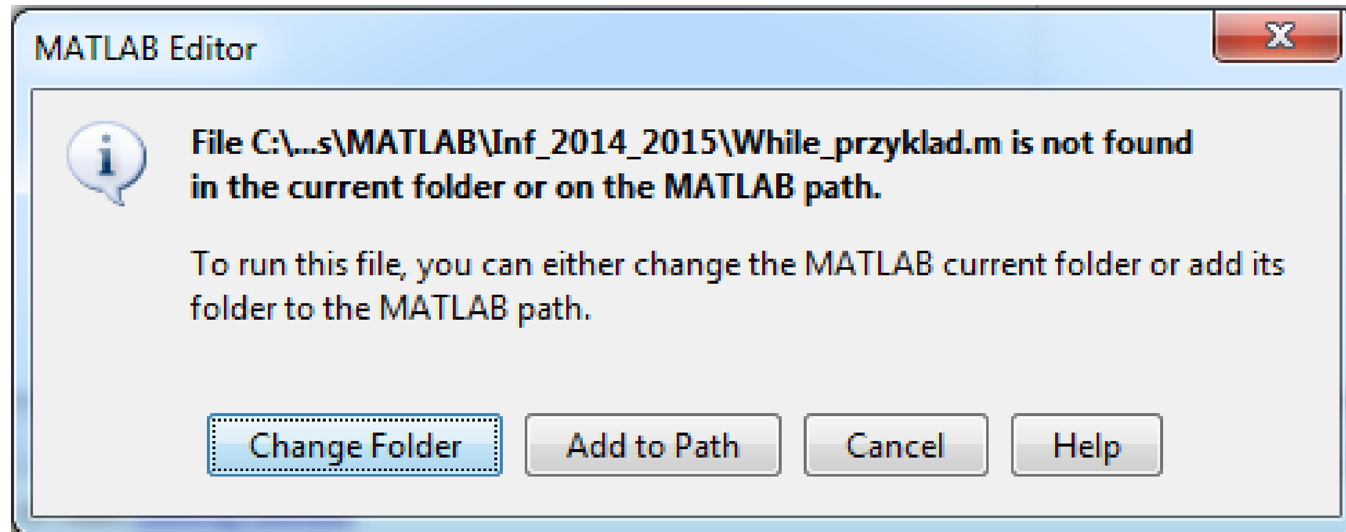
1. Czy nazwa jest aktualnie widzianą zmienną?
2. Czy nazwa jest subfunkcją?
3. Czy nazwa jest funkcją prywatną?
4. Czy nazwa jest funkcją w folderze roboczym?
5. Czy nazwa jest funkcją w folderze na ścieżce przeszukiwań?
6. Jeżeli w folderze jest kilka funkcji, to kolejność jest następująca:

   (1) Funkcja wbudowana, (2) MEX-plik, (3) MDL-plik, (4) P-plik, (5) M-plik.

Rozszerzona lista priorytetów Mrozek & Mrozek, 2010, s. 82

Okno zmiany folderu roboczego

- Pojawia się, gdy uruchamiamy skrypt z poza folderu roboczego lub z poza ścieżki przeszukiwań



Skrypty vs Funkcje

- Skrypt
  - Pracuje na ogólnej przestrzeni roboczej
  - Nie przyjmuje argumentów wejściowych
  - Nie zwraca argumentów wyjściowych
  - Wygodny przy analizie danych wykonywanej „ad hoc”
  - Wygodne przy wstępnym przygotowywaniu aplikacji

- Funkcja
  - Pracuje na lokalnej przestrzeni roboczej
  - Przyjmuje argumenty wejściowe
  - Zwraca argumenty wyjściowe
  - Wykorzystywane przy budowie większych aplikacji i rozbudowie MATLABA o dodatkowe funkcjonalności

Korzyści zamiany skryptu na funkcję

```
%Operacje zwiazane z wczytywaniem pliku
file_name1='data.txt'
data=dlmread(file_name1);
data=data(data>0);
%Obliczenia
corr=corrcoef(data);
av=mean(data);
s=std(data);
%Operacje zwiazane z wizualizacja
hist(data)
title('Histogram danych')
figure
plot(data)
title('Tytul wykresu')
xlabel('Opis osi X')
ylabel('Opis osi Y')
```

```
function [corr, av, s] = Statystyki (file_name)
data=PrzygotujDane(file_name) %Operacje zwiazane z wczytywaniem pliku
[corr, av, s] = ObliczParametry(data) % Obliczenia
WizualizujDane(data) %Operacje zwiazane z wizualizacja
end

>> [corr1, av1, s1]=Statystyki(`data.txt`)
```

1. **Lepsza i łatwiejsza kontrola kodu: nazw i wartości zmiennych**
2. **Modułowość - możliwość korzystania z funkcji w wielu różnych programach**
3. **Możliwość łatwej modyfikacji parametrów wejściowych**
4. **Poprawa przejrzystości**

Instrukcje sterujące przebiegiem programu

- Instrukcje warunkowe: ***if-elseif-else-end***, ***try-catch-end***
- Instrukcja wyboru: ***switch-case-otherwise-end***,
- Instrukcje pętli: ***while-end***, ***for-end***,

Instrukcje warunkowe if

```
if wyrazenie1                 %pierwszy sprawdzany warunek
                              %polecenia wykonywane, jezeli wyrazenie1 jest prawda
        polecenia             %czyli wyrazenie1 == TRUE

elseif wyrazenie2             %wyrazenie2 jest sprawdzane, gdy wyrazenie1 == FALSE

        polecenia             %wykonuja sie jezeli wyrazenie2 jest prawda

else

        polecenia             %wykonuja sie, gdy zadno z wczesniejszych wyrazen nie jest prawda

end
```

Instrukcje warunkowe if (przykład 1)

```
dawka_leku = 150  %domyslna dawka leku
wiek=0.5;         %zadeklarowanie wieku pacjenta
if (wiek<1)       %sprawdzenie warunku
   dawka_leku=0;  %jeżeli pacjent ma mniej niż 1 rok
end               %nie można podac leku
dawka_leku        %wyswietlenie koncowej dawki leku
```

Instrukcje warunkowe if (przykład 2)

```
wiek=0.5;            %ustalenie wieku pacjenta
if (wiek<1)          %sprawdzenie warunku
    dawka_leku=0;    %jeżeli pacjent ma mniej niż 1 rok
else                 % w przeciwnym wypadku
    dawka_leku=150 %domyslna dawka leku
end
dawka_leku           %wyswietlenie koncowej dawki leku
```

Instrukcje warunkowe (przykład 3)

```
waga=20;                        %ustawienie wagi pacjenta
wiek=5;                         %ustawienie wieku pacjenta
if (wiek<1)                     %sprawdzenie warunku 1
    dawka_leku=0;               %jeżeli pacjent ma mniej niż 1 rok
elseif (wiek<6 & waga<25)       %jeżeli „nie” warunek 1 to sprawdzenie warunku 2
    dawka_leku=100              %jeżeli 1<wiek<6  i jednocześnie waga < 25
else                            % jeżeli żaden z warunków nie jest spełniony to:
    dawka_leku=150              %domyslna dawka leku
end
dawka_leku                      %wyswietlenie koncowej dawki leku
```

Instrukcje warunkowe if (przykład 4)

```
waga=7              %ustawienie wagi pacjenta
wiek=0.5;                    %ustalenie wieku pacjenta
if (wiek<1)                  %sprawdzenie warunku
    dawka_leku=0;            %jeżeli pacjent ma mniej niż 1 rok
     if (waga>5)    %jeżeli waga jest większa niż 5 to:
        dawka_leku=50
    end
else                         % w przeciwnym wypadku
     dawka_leku=150 %domyslna dawka leku
end
dawka_leku                   %wyswietlenie koncowej dawki leku
```

Zagnieżdżony „if”

Instrukcje warunkowe „if” - podsumowanie

- Zastosowanie:
  - Wywołanie fragmentu kodu, jeżeli spełniony jest określony warunek
  - Wywołanie alternatywnych fragmentów kodu w zależności od spełnionych warunków

Instrukcja wyboru „*switch*”

```
switch wyrazenie lub zmienna wyboru %to co jest sprawdzane
    case lista stalych wyboru        %wartości zmiennej wyboru dla
        polecenia                    %pierwszego przypadku
    case lista stalych wyboru        %wartości zmiennej wyboru dla
        polecenia                    %drugiego przypadku
    otherwise                        %gdy zaden z powyzszych
        polecenia
end
```

Instrukcja wyboru „*switch*” - przykład

```
a=2
b=3
operator=`+`
switch operator             %to co jest sprawdzane
   case `+`                 %wartości zmiennej wyboru dla pierwszego przypadku
      c=a+b
   case {`*`,`.*`}          %wartości zmiennej wyboru dla drugiego przypadku
      c=a*b
   otherwise                %gdy zaden z powyzszych
      c=a-b
end
```

Co się stanie gdy:
operator = `/`

Instrukcja wyboru „*switch*” – przykład 2

```
plec=`k`
switch plec     %to co jest sprawdzane
    case `k`                %wartości zmiennej wyboru dla pierwszego przypadku
        [BMI, komunikat] = ObliczBMI_K(wzrost,waga)
    case `m`                %wartości zmiennej wyboru dla drugiego przypadku
        [BMI, komunikat] = ObliczBMI_M(wzrost,waga)
    otherwise               %gdy zaden z powyzszych
        [BMI, komunikat]=ObliczBMI(wzrost, waga)
end
```

Instrukcja wyboru „*switch*”

- Zastosowanie:
  - Realizacja alternatywnych fragmentów kodu w zależności od warunku sterującego

Instrukcje iteracyjne – pętla „**for**”

```
for zmienna_sterujaca=wektor_wartosci % ile razy petla ma sie wykonac
    polecenia                         % polecenia ktore beda iterowane
end
```

Ilość powtórzeń jest określona przez rozmiar wektora generowanego przez linię „for zmienna_sterująca=wyrażenie”

Pętla „**for**” - przykład

```
suma=0;              %zadeklarowanie zmiennej „suma”
a=[2 -1 0 3];
for i=1:4
   suma=suma+a(i)   %dodawanie kolejnych elementów wektora a
end
sum(a)               % sumowanie można wykonać dużo szybciej i łatwiej
```

Pętla „**for**”

- Zastosowanie:
  - Powtórzenie określonego fragmentu programu N razy. **Liczba N jest znana.**
- UWAGA!
  - Pętle „for” bardzo często lepiej jest zastąpić działaniami macierzowymi bądź tablicowymi

Instrukcje iteracyjne – pętla „**while**”

```
while wyrazenie %petla wykonuje się tak długo jak wyrażenie==TRUE
    polecenia
end
```

- Ilość powtórzeń **NIE JEST** określona z góry.
- Pętlę można przerwać komendą **break**

Pętla **„while”** - przykład

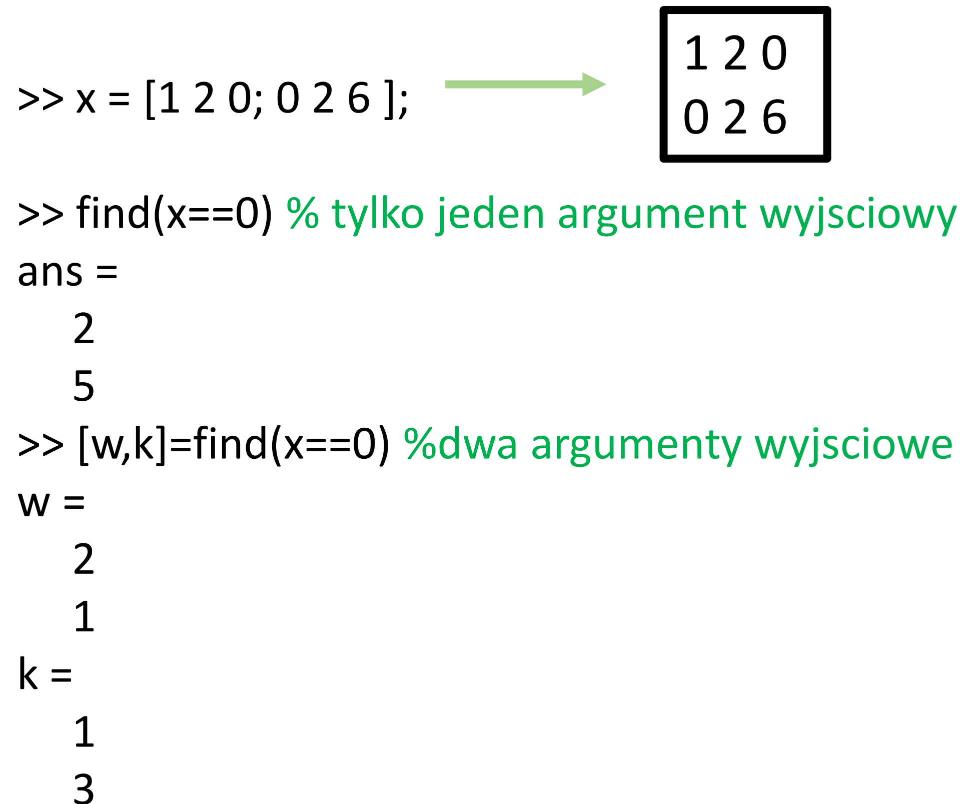
```
a=0
licznik=0;
while a<0.5                    %pętla wykonuje się do póki „a” jest < 0.5
    a=rand;                    %losowanie liczby z rozkładu U(0,1)
    licznik=licznik+1;         %licznik iteracji
end
licznik
a
```

Pętla **„while”**

- Zastosownie:
  - Powtarzanie fragmentu kodu, aż do osiągnięcia określonego warunku

Zmienna liczba argumentów wejściowych i wyjściowych funkcji

- Zachowanie funkcji może być różne w zależności od liczb argumentów podanych i oczekiwanych przy wywołaniu



```
>> x = [1 2 0; 0 2 6 ];   →   1 2 0
                              0 2 6

>> find(x==0) % tylko jeden argument wyjsciowy
ans =
    2
    5
>> [w,k]=find(x==0) %dwa argumenty wyjsciowe
w =
    2
    1
k =
    1
    3
```

```
>> sum(x) % tylko jeden argument wejściowy
ans =

    1     4     6

>> sum(x,2) %dwa argumenty wejściowe

ans =

    3
    8
```

Kontrola liczby argumentów: nargin / nargout

- *nargin* (number of arguments in) => zwraca liczbę argumentów wejściowych, użytych przy wywoływaniu funkcji
- *nargout* (number of arguments out) => zwraca liczbę argumentów oczekiwanych przy wywołaniu
- Np.:

```
>>[w, k] = find (x==0)
```

nargout zwróciłby „2” **nargin zwróciłby 1**

- Nargin/nargout mogą służyć do modyfikowania zachowania funkcji

Nargin/nargout - przykład

```
function [corr, av, s] = Statystyki (file_name1) % funkcja przyjmuje maksymalnie jeden arugment wejsciowy

if nargin == 0
   data=rand(100,2);                 %wygeneruj macierz z dwiema kolumnami losowych liczb
else
   data=dlmread(file_name1);         %wczytaj dane z pliku tekstowego
 end
corr=corrcoef(data);                 %wykonaj obliczenia
av=mean(data);
s=std(data);
end
```

Wywołanie:

```
>> [corr, av, s] = Statystyki ()             % wyliczone zostaną parametry dla losowych liczb
>> [corr, av] = Statystyki ('data.txt') %zwrócone zostaną macierz korelacji i wartości średnie dla danych z pliku „data.txt”
>> [corr, av] = Statystyki ('data.txt' , 'data2.txt') % wystąpi błąd – za dużo argumentów wejściowych
```

Maksymalna liczba argumentów przy wywołaniu

```
>> [corr, av] = Statystyki ('data.txt' , 'data2.txt') % wystąpi błąd – za dużo argumentów wejściowych
Error using Statystyki
Too many input arguments
```

- Nagłówek funkcji określa maksymalną liczbę argumentów WEJ i WYJ, możliwą przy wywołaniu funkcji

```
function [corr, av, s] = Statystyki(file_name1)
```

Tutaj maksymalna liczba argumentów WYJ to „3”

Maksymalna liczna argumentów WEJ to „1”

Zmienna liczba i typ argumentów wejściowych/wyjściowych

- varargin (variable arguments in)=> zmienna typu „cell”, w której przechowywane są argumenty wejściowe

- varargout (variable arguments out)=> zmienna typu „cell” w której umieszczane są argumenty wyjściowe

„Cell” – tablica, w której każdy element może być zmienną innego typu.

```
>> zmienna_cell = { [1 2 3], [0 1 ; 3 -6], 'test'};        %inicjalizacja zmiennej „cel”
>> zmienna_cell{2}                                         % odwołanie się do elementu 2 w zmiennej typu „cell”
ans =
    0    1
    3   -6
```

Varargin/varargout - przykład

```
function [corr, av, s] = Statystyki_var(varargin) % funkcja może przyjąć dowolny ciąg argumentów
if nargin == 0
    data=rand(100,2);
else
    data=[];
    for i=1:nargin                          %dodawanie do macierzy „data” danych z kolejnych plików
        data=[data dlmread(varargin{i})];
    end
end
corr=corrcoef(data);                        %obliczenia
av=mean(data);
s=std(data);
end
```

Wywołanie:

```
>>[corr,av]=Statystyki_var('data1.txt','data2.txt') %argumenty wejściowe sa automatycznie pakowane do zmiennej
                                                    %komórkowej
```

Zmienna liczba i typ argumentów WEJ/WYJ - podsumowanie

nargin/nargout

- Zmiana zachowana funkcji w zależności od liczby argumentów WEJ/WYJ użytych przy wywołaniu funkcji
- Liczby i typy argumentów są ograniczone przez definicję funkcji!

nargin/nargout + varargin/varargout

- Zmiana zachowania funkcji w zależności od:
  - liczby argumentów WEJ/WYJ
  - TYPU argumentów WEJ
- Można zwracać zmienne dowolnego typu
- Liczba argumentów WEJ i WYJ nie jest ograniczona z góry.

Zwracanie błędów – *error* i *assert*

- Przerywanie działania funkcji z powodu błędu

  Error('msgString') – następuje przerwanie funkcji i wyświetlenie komunikatu msgString na czerwono

- Sprawdzanie warunków początkowych:
  - Instrukcja „if" + *error*

```
if (~warunek)          %jeżeli warunek jest nie jest spełniony to zglos blad
    error ('msgString')
end
```

  - Komenda *assert*

```
assert (warunek, 'msgString')
```

Zwracanie błędów – przykład (error)

- Warunek istnienia trójkąta:

„Najdłuższy bok musi być krótszy niż suma pozostałych boków”

```
function [ pole, obwod ]= trojkatParametry_err( a,b,c, ha)

boki = [a b c];
warunek = max(boki)<sum( boki(boki~=max(boki))) % sprawdzenie warunku, wynik to zmienna logiczna
if ~warunek                                      %jeżeli warunek nie jest spełniony to zglos blad
    error('Taki trojkat nie istnieje')
end

pole=PoleTrojkata(a,ha)
obwod=ObwodTrojkata(a,b,c);
end
```

Zwracanie błędów – przykład (*assert*)

- Warunek istnienia trójkąta:

„Najdłuższy bok musi być krótszy niż suma pozostałych boków”

```
function [ pole, obwod ]= trojkatParametry_err( a,b,c, ha)

boki = [a b c];
warunek = max(boki)<sum( boki(boki~=max(boki))) % sprawdzenie warunku, wynik to zmienna logiczna

assert(warunek,'Taki trojkat nie istnieje')            %jeżeli warunek nie jest spełniony to zglos blad


pole=PoleTrojkata(a,ha)
obwod=ObwodTrojkata(a,b,c);
end
```

```
>> [pole,obwod]=trojkatParametry_ass(1,2,5,2)
warunek =
    0
Error using trojkatParametry_ass (line 6)
Taki trojkat nie istnieje
```

Programowa obsługa błędów *try – catch - end*

- Cel:
  - Przechwycenie błędu i podanie szczegółowego komunikatu
  - Przechwycenie błędu, obsłużenie go i kontynuowanie działania programu
- *Try-catch-end* to instrukcja sterująca podobna do instrukcji *if-else-end*

```
try                 %spróbuj wykonac
      polecenia
catch               %jeżeli wystapi blad wykonaj polecenia2
      polecenia2
end
```

```
if (wyr)
      polecenia %wykonaj jeżeli wyr==TRUE
else
      polecenia2 %wykonaj gdy wyr==FALSE
end
```

Obsługa błędów – przykład (1)

```
function [ pole, obwod ]= trojkatParametry_try_catch( a,b,c, ha)

boki = [a b c];
warunek = max(boki)<sum( boki(boki~=max(boki)));
try                                         %spróbuj wykonac
   if ~warunek
      error('Taki trojkat nie istnieje')
   end
   pole=PoleTrojkata(a,ha) ;                %obliczanie pola i obwodu, gdy „warunek” jest spelniony
   obwod=ObwodTrojkata(a,b,c);
catch                                       %gdy w bloku „try” rzucony zostanie wyjątek
  warning('Warunek istnienia nie zostal spelniony')     %wyświetlenie ostrzezenia – zolty komunikat
  pole=NaN;                                 %przypisanie zmiennej „pole” wartosci NaN
  obwod=NaN;                                %przypisanie zmiennej „obwod” wartości Nan
end

end
```

Obsługa błędów – przykład (1 cd.)

```
>> [pole,obwod]=trojkatParametry_try_catch(1,3,5,2)

Warning: Warunek istnienia nie spelniony
> In trojkatParametry_try_catch at 12

pole =

   NaN


obwod =

   NaN
```

```
>> [pole,obwod]=trojkatParametry_try_catch(3,4,5,4)

pole =

     6


obwod =

    12
```

Obsługa błędów – przykład (2)

```
function wynik = MnozenieMacierzy(A,B)
try
   wynik=A*B
catch                    %jeżeli nastapi blad:
   sA=size(A);           %pobierz rozmiary A
   sB=size(B);           %pobierz rozmiary B
   error('Rozmiary A: %d x %d, Rozmiary B: %d x %d', sA, sB) %wyświetl wlasny komunikat bledu
end
```

```
>> MnozenieMacierzy([2 1], [2 5])
Error using MnozenieMacierzy (line 7)
Rozmiary A: 1 x 2, Rozmiary B: 1 x 2
```

Obsługa błędów – obiekt MException

```
function wynik = MnozenieMacierzy(A,B)
try
   wynik=A*B
catch err                %zlap „wyjątek”
   err                   %wyświetl obiekt err
   sA=size(A);           %pobierz rozmiary A
   sB=size(B);           %pobierz rozmiary B
   error('Rozmiary A: %d x %d, Rozmiary B: %d x %d', sA, sB)
end
```

```
>> MnozenieMacierzy([2 1], [2 5])

err =

  MException with properties:

    identifier: 'MATLAB:innerdim'
       message: 'Inner matrix dimensions must agree.'
         cause: {0x1 cell}
         stack: [1x1 struct]

Error using MnozenieMacierzy (line 8)
Rozmiary A: 1 x 2, Rozmiary B: 1 x 2
```

Obiekt MException zawiera informacje o błędzie: identyfikator, komunikat, przyczynę, miejsce w kodzie MATLABA, w którym wystąpił .

Co było najważniejsze?

- Ze skryptów korzystamy przy analizach „na szybko”.
- Większe programy należy budować przy użyciu funkcji.
- Działanie funkcji można uzależnić od jej argumentów WEJ/WYJ.