Procedury użytkownika

W napisanych do tej pory programach korzystaliśmy z funkcji wchodzących w skład Scilaba, takich jak np. `sin, cos, sum, floor, rand`... (tu możecie spróbować wymienić więcej poznanych do tej pory funkcji). Scilab oferuje do szeroki wybór funkcji, wystarczy zajrzeć do Pomocy, aby zobaczyć ich listę oraz zapoznać się z opisem działania i przykładami wykorzystania.
Możliwe jest również opracowywanie (a następnie wykorzystywanie) własnych funkcji. Takie funkcje, dla odróżnienia, będziemy nazywać **procedurami**. Podobnie jak programy, procedury mogą być bardziej lub mniej skomplikowane, mogą składać się jednej bądź bardzo wielu instrukcji. Zaczniemy od bardzo prostego przykładu.

Uwaga: w tym opracowaniu nie będziemy się zagłębiać w formalne rozróżnienia pojęcia funkcji i procedury.

Przypomnijmy sposób, w jaki korzystaliśmy z, na przykład, funkcji sinus. Takie **wywołanie** mogło mieć postać:
```
y=sin(x);
```

W wyniku wykonania tej instrukcji obliczona jest wartość sinusa wartości zmiennej `x` (`x` musi mieć określoną wartość) i podstawiona do zmiennej `y`. Jeśli `x` byłby na przykład wektorem 100—elementowym, to również wektor `y` miałby 100 elementów, o wartościach równych odpowiednio `y(1)=sin(x(1)), y(2)=sin(x(2)),..., y($)=sin(($)).`
Inne wywołanie funkcji sinus mogłoby być takie:
```
velo=sin(alfa);
```

W wywołaniu pojawia się nazwa funkcji (`sin`) oraz argument (parametr); w powyższych przykładach parametrem były kolejno zmienne `x` oraz `alfa`. W ogólnym przypadku, w zależności od funkcji, parametrów może być więcej. Wynik działania funkcji podstawiany jest pod zmienną (tu: `y` i `velo`). Jednak może być i tak, że wynik działania funkcji stanowi część bardziej złożonego wyrażenia, np:
```
z=2*%pi*sin(2*beta);
```
W tym przykładzie wartość funkcji `sin` jest wyznaczona dla argumentu równego `2*beta`, a wyliczona wartość sinusa jest wykorzystana do wyznaczenia wartości zmiennej `z`. A zatem argumentem może być nie tylko zmienna, ale również wyrażenie.

Projektujemy procedurę użytkownika
Projektując procedurą oczywiście przede wszystkim trzeba wiedzieć, co dana procedura ma robić (dosyć oczywiste stwierdzenie). To, co funkcja ma robić, należy zapisać za pomocą zmiennych i instrukcji, podobnie jak pisze się fragmenty kodu.
Załóżmy, że funkcja ma wyznaczać średnią dwóch wartości.
Instrukcja to realizująca może wyglądać tak:
```
z=(x+y) * 0.5;
```

O ile znane są wartości `x` i `y`, wartością zmiennej `z` będzie średnia `x` i `y`. Czyli wysyłając naszą procedurę (której jeszcze nie ma) do realizacji, musimy przekazać jej informację o wartościach `x` i `y`. Oznacza to, że zmienne `x` i `y` powinny być jej parametrami.

Budujemy procedurę użytkownika
Aby z ciągu instrukcji 'zrobić' procedurę należy obudować ją instrukcją 'tytułową', wykorzystującą słowo kluczowe Scilaba *function* oraz instrukcją kończącą *endfunction*. Można uważać procedurę za 'paczkę' zawierającą ciąg instrukcji, opakowaną w instrukcje `function` i `endfunction`.

Nasza przykładowa procedura może wyglądać tak:
```
function z=srednia(x, y)
z=(x+y)*0.5;
endfunction
```

Jest to funkcja o dwóch parametrach wejściowych (`x,y`) i jednym parametrze wyjściowym (wyniku) `z`.

Uwaga:
Nazwy parametrów (tu: `x` i `y`) pełnią rolę symboliczną – w wywołaniu w ich miejsce zostaną podstawione parametry, które pojawią się w wywołaniu. Procedura jest 'wzorcem' czynności do opisania.
Zmienna wynikowa (tu: `z`) koniecznie musi mieć nadaną wartość w toku realizacji instrukcji wchodzących w skład funkcji.

Parametry procedury
Tak jak w przykładzie wykorzystania funkcji `sin` widzieliśmy, że nazwy parametrów w wywołaniu funkcji mogą być różne, podobnie, w wywołaniu naszej funkcji `srednia` można w różny sposób podawać wartości, dla których faktycznie zostaną wykonane obliczenia.

Przykładowe wywołania:
Parametrem w wywołaniu funkcji może być stała:
```
y = srednia (1, 10);
```
Realizacja: wartość pierwszego argumentu (`1`) jest 'podstawiana' pod wartość pierwszego argumentu w definicji procedury (`x`), wartość drugiego argumentu (`10`) jest podstawiana pod wartość drugiego argumentu w definicji procedury (`y`), następuje realizacja instrukcji z procedury dla 'konkretnych' wartości, wyniki wyprowadzany jest na zewnątrz. Czyli kolejność parametrów ma podstawowe znaczenie, natomiast ich nazwy nie mają znaczenia.

Parameterem może być zmienna:
```
a=5;
b=15;
cel = srednia(a,b);
```
Parametrem mogą być wyrażenia:
```
zet = srednia(2*a, 4*a);
```
Realizacja: wartością pierwszego argumentu (`x`) będzie wartość `2*a`, czyli `10`, zaś wartością drugiego argumentu (`y`) będzie `4*a=20`.

Jeśli argumentem będą wektory (o zgodnych wymiarach), to wynik również będzie wektorem:
```
x=(1:1:10)
d=(2:2:20)
wek=srednia(x,d)

-->x=(1:1:10)
 x  =
```

```
    1.    2.    3.    4.    5.    6.    7.    8.    9.    10.

-->d=(2:2:20)
 d  =
     2.    4.    6.    8.    10.    12.    14.    16.    18.    20.

-->wek=srednia(x,d)
 wek  =
    1.5    3.    4.5    6.    7.5    9.    10.5    12.    13.5    15.
```

Próba wykonania obliczeń dla wektorów o niezgodnych długościach, powoduje pojawienie się komunikatu o błędzie:

```
-->d_pol=(2:2:10);
-->wek=srednia(x,d_pol)
 !--error 8
Niezgodne dodawanie.
at line       2 of function srednia called by :
wek=srednia(x,d_pol)
```

Nasza procedura zadziała również dla macierzy; poniżej pokazujemy przykład wyznaczenia średnich elementów macierzy o wymiarze `2x2`.

```
-->a=[1, 1, ; 2, 2]
 a  =
     1.    1.
    2.    2.
 -->m=[3, 3 ; 8, 10];
 m  =
     3.    3.
    8.    10.
 -->x=srednia(a, m)
 x  =
     2.    2.
    5.    6.
```

Jak pisać procedury i gdzie je przechowywać?

Instrukcje wchodzące w skład procedury można po prostu wpisać w konsoli Scilaba, a następnie wykorzystywać w dowolnym momencie sesji. Jednak to rozwiązanie nie jest praktyczne, lepiej jest wpisać je w oknie edytora Scilaba i zapisac w pliku.

Aby taką funkcję zapisaną w pliku ‘zaktywować’ (tak aby Scilab ‘przypomniał’ sobie jej treść) należy napisać:

```
exec (‘nazwa.pliku’);
```

a następnie wywołać funkcję, która znajduje się w tym pliku.

Przykłady:

```
function y = rzut()
y = floor(rand(1,1)*6+1)
endfunction
```

Funkcja generująca wartość całkowitą z przedziału [1, 6], którą można wykorzystać do symulowania rzutów kostką do gry. Jest to przykład funkcji *bezparametrowej*.

- Wykorzystać procedurę `srednia` do obliczenia średnich wartości sąsiadujących ze sobą elementów dowolnego wektora.

Załóżmy, że dany jest wektor `x` o pewnej liczbie elementów.

Chcemy obliczyć kolejno wartości: `(x(1)+x(2))/2, (x(2)+x(3))/2`, itd.

Zróbmy wektor pomocniczy:
`y=[0,x];`
Jeśli teraz wykonamy instrukcję:
`z=srednia (x(2:$), y(2:$-1));`
to w wektorze z znajdą się kolejno wartości `(x(1)+x(2))/2, (x(2)+x(3))/2, ...`

Uwaga:
`$` oznacza numer ostatniego elementu wektora. Czyli `x(2:$)` oznacza wycinek wektora `x` od drugiego do ostatniego elementu.
Dzięki temu, że kolejne elementy wektora pomocniczego `y` są przesunięte o `1` względem elementów wektora `x` – uzyskujemy pożądany wyniki.

Zadania

Napisać procedury realizujące następujące czynności:

- dla danej wartości promienia `r` wyznaczyć pole powierzchni trójkąta równobocznego wpisanego w okrąg o takim promieniu.

```
function p=pole(r)
// trojkat rownoboczny jest dzielony na trzy trojkaty rownoramienne
// o ramionach o dlugosci r, wysokosci h (do obliczenia)
// i podstawie o dlugosci 2*a (do obliczenia)
// rysunek pomoze zrozumiec ponizsze instrukcje
kat=360 / 3;
kat_rad = kat * %pi /180;
h = r * cos(kat_rad/2);
a= r * sin (kat_rad/2);
p= 3 * h * a;
endfunction;
```

Wywołanie:
```
trojkat = pole(1);
```

- dla danej wartości promienia `r` wyznaczyć pole powierzchni kwadratu wpisanego w okrąg o takim promieniu.
- dla danej wartości promienia `r` wyznaczyć pole powierzchni pięciokąta foremnego wpisanego w okrąg o takim promieniu.
- Uogólnić program na przypadek dowolnego wielokąta foremnego (potraktować liczbę boków jako dodatkową daną, a zatem trzeba wprowadzić dodatkowy parametr do funkcji).

Wskazówka: W powyższych zadaniach można wykorzystać i zmodyfikować procedurę wyznaczającą pole trójkąta wpisanego w okrąg – wystarczą nieznaczne modyfikacje. Pomocny będzie rysunek.