Zmienne

Po nieco intuicyjnych początkach, zajmiemy się obiektami, na których opiera się programowanie – są to *zmienne*.

Zmienne

Programy operują na *zmiennych*. Nadawanie im wartości odbywa się poprzez instrukcję *podstawienia*. Interpretacja tej instrukcji jest następująca: zmiennej znajdującej się z lewej strony instrukcji podstawienia nadana jest wartość wyrażenia znajdującego się po prawej stronie instrukcji. Zmienne są identyfikowane poprzez *nazwę*.

```
a=2.5
b=6.3
c=a+b
```

Uwagi:

Umieszczenie średnika na końcu polecenia sprawia, że wyliczona wartość (lub wartości) **nie jest wyświetlana** na konsoli.

W powyższych przykładach zmienne `a, b, c` przechowują po jednej wartości.

Odwołanie do wartości zmiennej (prawa strona instrukcji podstawienia) nie wpływa na jej wartość:

```
a=5;
c=a+5;
```

Wartością zmiennej `a` jest 5, a zmiennej `c` – 10.

Jest błędem odwołanie się do zmiennej, której nie przypisano wcześniej żadnej wartości.

Nadanie zmiennej wartości powoduje zniszczenie jej poprzedniej wartości:

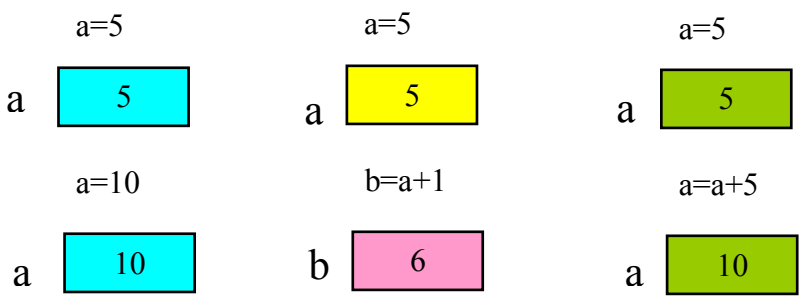
```
a=5;
a=10;
```

Ważne! Popatrzmy na sekwencję instrukcji:

```
n=1;
n=n+1;
```

Przypomnijmy interpretację instrukcji podstawienia: zmiennej z lewej strony nadawana jest wartość wyrażenia z lewej strony instrukcji. A zatem w tym przykładzie zmiennej `n` (lewa strona instrukcji) nadana jest wartość wyrażenia z prawej strony instrukcji (wynosząca `n+1`, czyli 2, bowiem `n=1`). A zatem nową wartością `n` staje się 2. Poprzednia wartość jest niszczona.

Instrukcje podstawienia tego typu są bardzo często wykorzystywane w programach.



Oprócz zmiennych, w programach wykorzystuje się stałe, np. 5, 2, 3.456.

Scilab wyróżnia kilka stałych specjalnych:

`%i` - wartość urojona równa $\sqrt{-1}$

`%pi` - π

`%e` - podstawa logarytmu naturalnego

`%eps` - największa wartość, dla której $1 + \%eps = 1$

`%inf` - nieskończoność (komputerowa)

`%t, %f` - zmienne logiczne o wartościach prawda (T - true) i fałsz (F - false) (Co wyświetli się po wypisaniu *wyrażenia logicznego* 2>5?)

%nan – Not a Number

Na zmiennych i stałych wykonuje się operacje. Na zmiennych i stałych o typie liczbowym wykonuje się operacje arytmetyczne.

Operatory arytmetyczne:

w wyrażeniach wykorzystuje się następujące operatory:

+ – dodawanie

- – odejmowanie

* – mnożenie

/ – dzielenie

^ – potęgowanie

Priorytety wykonywania operacji arytmetycznych:

Wartość wyrażenia jest wyznaczana od lewej strony do prawej, w kolejności zgodnej z priorytetami operacji (potęgowanie, mnożenie/dzielenie, dodawanie/odejmowanie). Kolejność można zmieniać poprzez użycie nawiasów ().

W wyrażeniach mogą pojawiać się funkcje Scilaba, na przykład:

`sin(x), cos(x), tan(x), cotg(x)`

`abs(x)` – wartość bezwzględna

`sqrt(x)` – pierwiastek kwadratowy

`exp(x)` – e ^x (e- stała Eulera, można się do niej odwoływać poprzez `%e`)

Można również wykorzystywać własne funkcje, ale tym zajmiemy się poźniej.

Argumenty funkcji w Scilabie umieszcza się w nawiasach ( ).

Scilab nie wymaga *deklarowania* zmiennych.

Ogólne operacje na zmiennych

`who` – wyświetlenie wszystkich używanych zmiennych

`who_user` – wyświetlenie zmiennych użytkownika

`clear()` – usunięcie (wyczyszczenie) wszystkich zdefiniowanych zmiennych

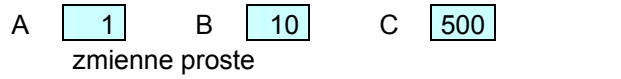
`clear nazwa` – usunięcie zmiennej o nazwie `nazwa`.

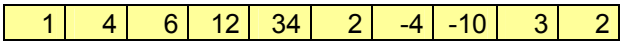
Wektory i macierze

Na razie w przykładach występowały *zmienne proste* – jednej nazwie odpowiadała jedna wartość (można powiedzieć, jedno pudełko służące do przechowywania wartości). Ale przypomnijmy, że do rysowania wykresów wykorzystywane były ‘większe’ obiekty, w

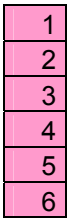
których pod jedną nazwą znajdowała się większa liczba wartości (na przykład wartości ‘x-ów’ lub ‘y-ków’). Takie większe obiekty to *wektory* (inaczej: tablice jednowymiarowe).

Wektory
W programie Scilab większość operacji jest wykonywanych w odniesieniu do wektorów. Wektor to ciąg wartości. O ile zmiennej prostej odpowiadało pudełko z jedną wartością, to wektorem jest ciąg pudełek. Ten ciąg pudełek, z których każde może przechowywać jakąś wartość, jest identyfikowany za pomocą jednej nazwy.





X wektor



Y- też wektor

Istotne jest rozróżnienie wektora wierszowego i kolumnowego, ale o tym za chwilę. Jeśli znacie programy typu arkusz kalkulacyjny, to możecie wyobrażać sobie wektory jako fragmenty wiersza lub kolumny w arkuszu. Zauważmy też, że **zmienna prosta jest szczególnym przypadkiem wektora** – złożonego z jednego elementu.

Definiowanie wartości elementów wektorów – pierwsze podejście (chociaż już to robiliśmy podczas rysowania wykresów)

Wektor wierszowy – kolejne elementy oddzielone spacją lub przecinkiem, całość umieszczona w nawiasach kwadratowych:

```
x=[1 4 6 12 34 2 -4 -10 3 2]
```

albo

```
x=[1, 4, 6, 12, 34, 2, -4, -10, 3, 2]
```

Wektor kolumnowy - elementy oddzielone średnikiem lub pisane od nowego wiersza:

```
y=[1; 2; 3; 4; 5; 6]
r=[1
2
3
4
5
6]
```

Uwaga: W przypadku dużych zestawów wartości lepiej jest umieszczać średnik na końcu polecenia.

Wektor kolumnowy można zamienić na wierszowy – i odwrotnie – korzystając z symbolu ‘ (apostrof). Taka operacja nazywa się **transpozycją**.

```
xt = x';
```

```
yt = y';
```

Wektory można konstruować w oparciu o wcześniej zdefiniowane zmienne:
```
a=1
b=2
c=3
x=[a, b, c]
```

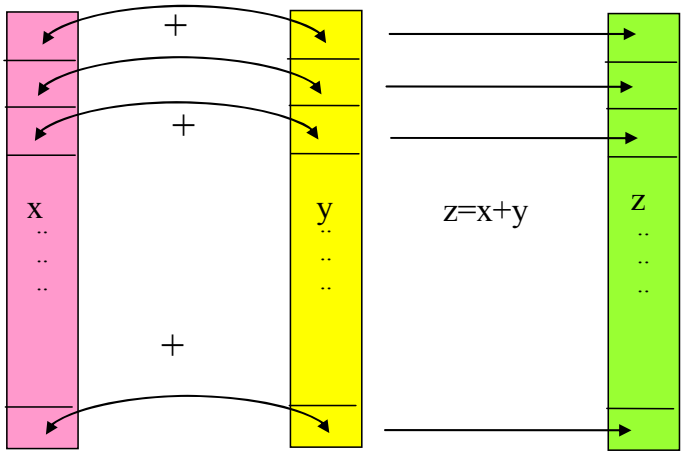
Wektor można zdefiniować zadając wartość pierwszego elementu, krok oraz wartość ostatniego elementu, oddzielając je dwukropkiem:
```
x=[-10:0.5:10]
```
Zwróćmy uwagę na różnicę po umieszczeniu znaku ‘:
```
x=[-10:0.5:10]’
```

Z kolei polecenie:
```
z=linspace (-10, 10, 101);
```
wygeneruje wektor (wierszowy) o 101 elementach, przyjmujących wartości z przedziału [-10, 10].
Polecenie
```
v=linspace (-10, 10, 101)’;
```
wygeneruje wektor kolumnowy – wartości kolejnych elementów będą takie same jak poprzednio.

Jeśli mamy wektor x, o elementach przykładowo równych:
```
x=linspace(0, %pi, 20);
```
to instrukcja
```
y=sin(x);
```
utworzy wektor `y` o takiej samej liczbie elementów jak wektor `x`, i też będzie to wektor wierszowy.

Na wektorach o **zgodnych wymiarach** (co to oznacza?) **można wykonywać różne operacje**:
```
z=x+y;
w=x-y;
v=x+2*y;
u=sqrt(abs(x));
```

W powyższych przykładach działania są wykonywane ‘element po elemencie’.



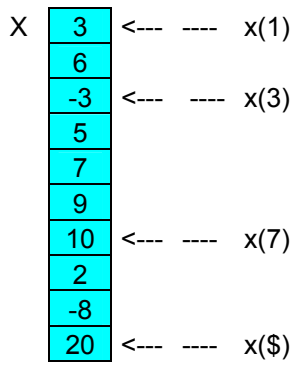
Uważajmy na mnożenie, w przypadku wektorów operacja `x*x` nie jest określona. Pomijamy tę sprawę, a na razie na pociechę mamy działanie:

```
x.*x
```
oznaczające mnożenie element wektora po elemencie.
Podobnie jest dzieleniem, na razie ograniczmy się do działania 'element po elemencie', czyli:
```
x./x
```

Pytanie: Jakie wartości będzie miał wektor w wyniku wykonania ostatniej instrukcji?
Uwaga: w przypadku próby wykonania dzielenia przez 0, wynikiem będzie 'Nan' (*Not a number*), co oznacza, że działanie nie było możliwe do wykonania. Ogólnie, poprawne programy powinny być tak skonstruowane, alby nie dopuścić do takiej sytuacji. Wrócimy do tego później

Odwołanie do elementu wektora:
Do *elementów* wektorów można odwoływać się poprzez podanie indeksu, czyli numeru elementu, na przykład:
```
a=x(1)
b=2*x(7)
c=y(2)+x(3)
```
Element wektora to już tylko jedna wartość, czyli obiekt 'taki sam' jak zmienna prosta.
Indeksem może być zmienna:
```
i=5;
y=x(i);
```

Jeśli jako indeks poda się $, to następuje odwołanie do *ostatniego* elementu wektora.
```
last=x($)
```



Szybkie tworzenie specjalnych wektorów
`c=ones(5,1)` – wektor kolumnowy pięcioelementowy, wartości wszystkich elementów są równe 1
`c=ones(1,5)` – wektor wierszowy pięcioelementowy, wartości wszystkich elementów są równe 1
`d=zeros(10,1)` - wektor kolumnowy dziesięcioelementowy, wartości wszystkich elementów są równe 0
`d=zeros(1,10)` - wektor kolumnowy dziesięcioelementowy, wartości wszystkich elementów są równe 0

`liczby=linspace(1,100,100)` – wektor 100 elementowy, którego kolejne elementy są równe 1, 2, ..., 100.
`los1=rand(10,1)` – wektor (kolumnowy) o 10 elementach pseudolosowych o rozkładzie jednostajnym [0, 1).
`los2=rand(1,10)` – wektor (wierszowy) o 10 elementach pseudolosowych o rozkładzie jednostajnym [0, 1).

Zadanie: Napisać instrukcję 'symulującą' 10 kolejnych rzutów kostką do gry.
Co się może przydać oprócz funkcji `rand`? Zapewne funkcje zaokrąglające wartości rzeczywiste do wartości całkowitych. Oraz trzeba przeliczyć zakres wartości [0,1) na zakres od 1 do 6.
`round` – zaokrąglenie do najbliższej wartości całkowitej
`floor` – zaokrąglenie do dołu
`ceil` – zaokrąglenie do góry
Może tak?

```
kostka=floor(rand(10,1)*6+1)
```

Jeszcze o wektorach...
Wektory można ze sobą 'zlepiać':

```
x=ones(1,5);
z=2*ones(1,5);
xiz=[x,z]
-->x=ones(1,5);
-->z=2*ones(1,5);
-->xiz=[x,z]
 xiz  =

    1.    1.    1.    1.    1.    2.    2.    2.    2.    2.
```

Można wyciąć fragment wektora:

```
-->srodek=xiz(3:6)
 srodek  =

    1.    1.    1.    2.
```

W poprzednim przykładzie zlepialiśmy dwa wektory wierszowe. Podobnie, można połączyć wektory kolumnowe; znakiem oddzielającym jest średnik:

```
-->wek1=[1,2,3]'
 wek1  =

    1.
    2.
    3.

-->wek2=[10,20,30]'
 wek2  =

    10.
    20.
    30.

-->wek3=[wek1;wek2]
 wek3  =

    1.
    2.
    3.
```

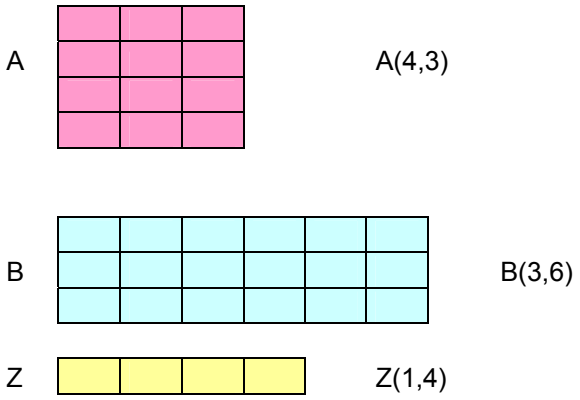
```
    10.
    20.
    30.
```

Jeśli jako znak oddzielający poda się przecinek... to uzyska się inny obiekt o bardziej złożonej strukturze:

```
-->obiekt=[wek1,wek2]
 obiekt  =

    1.    10.
    2.    20.
    3.    30.
```

Obiektem tym jest *macierz*, inaczej tablica *dwuwymiarowa*. Można powiedzieć, że wektor jest szczególnym przypadkiem macierzy (zbudowanej odpowiednio z jednej kolumny lub jednego wiersza), również zmienna prosta to szczególna macierz o jednym wierszu i jednej kolumnie. Macierz – to jakby wycinek arkusza kalkulacyjnego. I jak w przypadku arkusza, aby wskazać na konkretny element macierzy, trzeba wykorzystać dwa indeksy, pierwszy wskazuje na numer wiersza, a drugi na numer kolumny.



Zadanie
Utworzyć macierz o 5 wierszach i 5 kolumnach, przy czym wszystkie elementy pierwszego wiersza mają mieć wartość1, drugiego – 2, itd.

```
-->wiersz=ones(1,5)
 wiersz  =

    1.    1.    1.    1.    1.

-->macierz=[wiersz;wiersz*2;wiersz*3;wiersz*4;wiersz*5]
 macierz  =

    1.    1.    1.    1.    1.
    2.    2.    2.    2.    2.
    3.    3.    3.    3.    3.
    4.    4.    4.    4.    4.
    5.    5.    5.    5.    5.
```

Zadanie
Powtórzyć poprzednie zadanie, zastępując wiersze kolumnami.

Funkcje wektorowe
Kilka funkcji na wektorach:
`s=sum(x)` – suma elementów wektora x
`c=max(x)` – wartość maksymalnego elementu wektora x
`d=min(x)` – wartość minimalnego elementu wektora x
`z=gsort(x)` – uporządkowanie malejąco elementów wektora x – wynik umieszczony jest w wektorze z.

Zapisanie sesji do pliku:
```
diary ('sesja.txt')
.....
.....
diary (0)
```

W pliku `sesja.txt` znajduje się zapis wszystkich wykonanych poleceń oraz wyników wypisywanych na monitor pomiędzy poleceniem `diary ('sesja.txt')` a poleceniem `diary (0)`.

Uwaga: Jeśli w poleceniu wymagającym podania nazwy pliku poda się tylko nazwę, to domyślnie zakłada się, że plik ten znajduje się w *katalogu bieżącym*; nazwę katalogu bieżącego można wyświetlić za pomocą polecenia `pwd`. Katalog bieżący można zmienić poleceniem *Plik/ Zmiana bieżącego katalogu*. Innym sposobem uniknięcia nieporozumień jest podawanie nazwy pliku wraz z pełną ścieżką dostępu.

Skrypty
Polecenia programu Scilab można umieścić w pliku (skrypcie) i wielokrotnie wykonywać. W plikach oprócz poleceń można umieszczać komentarze; są to linie rozpoczynające się //. Do pisania skryptu najwygodniej jest skorzystać z okna edytora.
Zwyczajowe rozszerzenie plików skryptowych to `.sce`. Uruchomienie poleceń z pliku następuje po wprowadzeniu polecenia:
```
exec ('plik.sce')
```

Innym sposobem wykonania ciągu instrukcji zapisanych w oknie edytora jest ich skopiowanie do okna konsoli (`Copy – Paste`), albo wybór opcji `Wykonaj / plik (z echem`, albo `bez echa`).