Zmienne:
W programie operuje się 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.

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

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

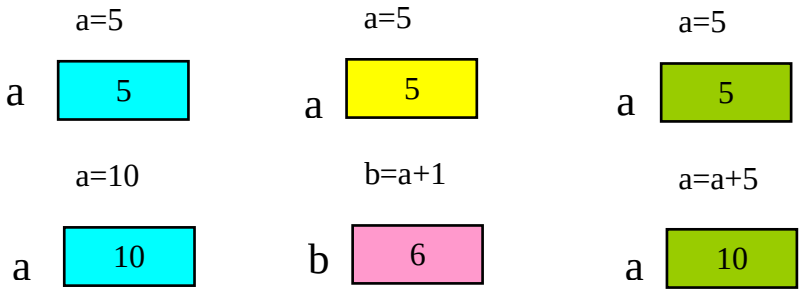
Nadanie zmiennej wartości powoduje zniszczenie jej poprzedniej wartości:

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

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.



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?)

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, 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`)

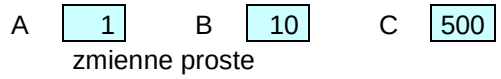
Argumenty funkcji w Scilabie umieszcza się w nawiasach ( ).

Scilab nie wymaga *deklarowania* zmiennych.

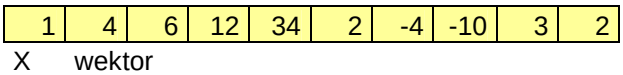
Ogólne operacje na zmiennych
`who` – wyświetlenie wszystkich używanych zmiennych
`who_user` – wyświetlenie zmiennych użytkownika
`clear` – usunięcie (wyczyszczenie) wszystkich zdefiniowanych zmiennych
`clear nazwa` – usunięcie zmiennej o nazwie `nazwa`.

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).

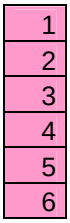
Wektory
Program Scilab jest programem wektorowym, większość operacji jest wykonywanych w odniesieniu do wektorów. Wektor to ciąg wartości. O ile zmiennej prostej odpowiadało pudełko z jedną wartością, to wektorem jest ciąg pudełek. Ten ciąg pudełek, z których każde może przechowywać jakąś wartość, jest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



zmienne proste



X wektor



Y- też wektor

Istotne jest rozróżnienie wektora wierszowego i kolumnowego, ale o tym za chwilę.

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

Wektor wierszowy – kolejne elementy oddzielone spacją lub przecinkiem:

```
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

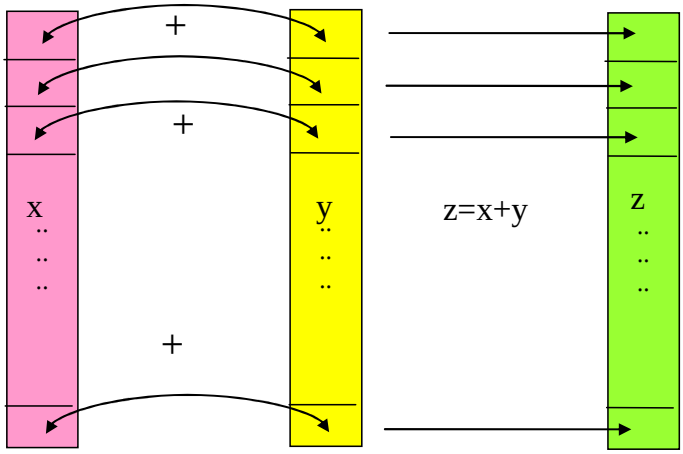
```
v=linspace (-10, 10, 101)’;
```

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(10)
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);
```

Szybkie tworzenie specjalnych wektorów

`c=ones(5)` – wektor pięcioelementowy, wartości wszystkich elementów są równe 1

`d=zeros(10)` - wektor 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 (wierszowy) o 10 elementach pseudolosowych o rozkładzie jednostajnym <0, 1).

`los2=rand(1,10)` – wektor (kolumnowy) o 10 elementach pseudolosowych o rozkładzie jednostajnym <0, 1).

Uwaga: Czy pamiętacie zadania o Pimpku rzucającym kostką do gry? Napiszmy instrukcję ‘symulującą’ 10 kolejnych rzutów kostką.
Co się może przydać oprócz funkcji `rand`? Zapewne funkcje zaokrąglające wartości rzeczywiste do wartości całkowitych.

`round` – zaokrąglenie do najbliższej wartości całkowitej

`floor` – zaokrąglenie do dołu

`ceil` – zaokrąglenie do góry

Może tak?

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

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.

Zadanie

Narysować spiralę Archimedesa dla danej wartości parametru a.
Narysować różę – porównać wykresy dla k parzystego i nieparzystego.

Spirala Archimedesa

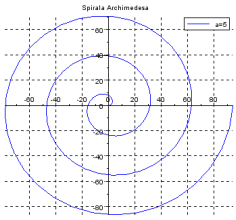
$r = a \cdot \varphi, \quad \varphi \geq 0$

a – parametr

```
-->a=5;
-->fi = linspace(0, 6*%pi, 100);
-->r=a*fi;

Przypomnienie: przechodzimy z układu
współrzędnych biegunowych do układu
współrzędnych kartezjańskich:

-->x=r.*cos(fi);
-->y=r.*sin(fi);
-->plot(x,y)
```



Róża

$r = a\cos(k\varphi), \quad \varphi \geq 0$

a, *k* - parametry

Skrypty

Polecenia programu Scilab można umieścić w pliku (skrypcie) i wielokrotnie wykonywać. W plikach oprócz poleceń można umieszczać komentarze; są to linie rozpoczynające się //. Najwygodniej skorzystać z okna edytora.

Zwyczajowe rozszerzenie plików skryptowych to `.sce`. Uruchomienie poleceń z pliku następuje po wprowadzeniu polecenia:

```
exec plik.sce
```

Na przykład:

```
a=5;
fi = linspace(0, 6*%pi, 100);
r=a*fi;
x=r.*cos(fi);
y=r.*sin(fi);
clf();
a1=10;
r1=a1*fi;
x1=r1.*cos(fi);
y1=r1.*sin(fi);
//Uwaga: .* oznacza mnożenie 'element po elemencie'
plot(x,y,x1,y1);
title('Spirala Archimedesa');
legend ('a=5','a=10');
xgrid();
```

Innym sposobem wykonania ciągu instrukcji zapisanych w oknie edytora jest ich skopiowanie do okna konsoli (Copy – Paste).

Pętla – wielokrotne wykonywanie ciągu instrukcji.

Bardzo często w programowaniu wykorzystuje się wielokrotne powtarzanie określonego ciągu czynności (instrukcji).

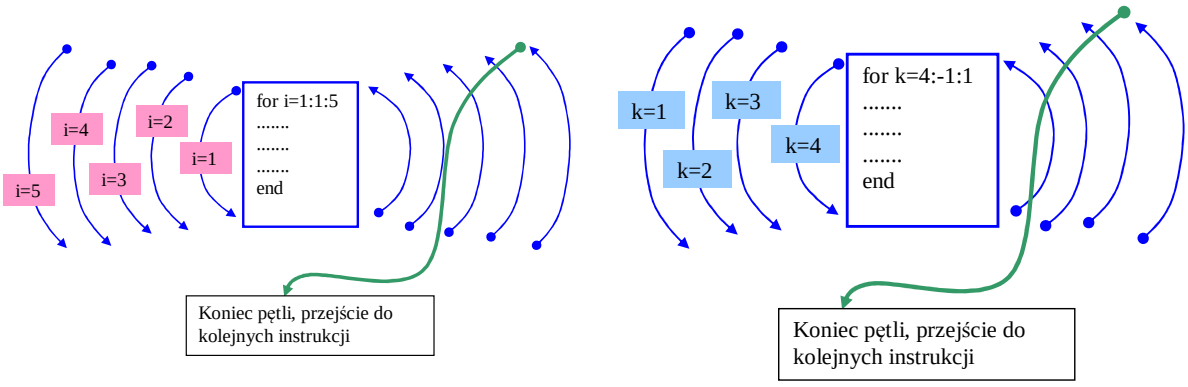
Rozróżniamy sytuacje, gdy liczba powtórzeń jest znana – oraz gdy liczba powtórzeń zależy od konkretnej sytuacji napotkanej w toku obliczeń.

Pętla for

```
for i=1:10
....
....
end
```

Instrukcje zawarte pomiędzy instrukcją `for` a instrukcją `end` zostaną wykonane 10 razy, wynika to z zapisu `i=1:10`. Zmienna `i` (nazwa przykładowa) nazywana jest *zmienną sterującą*; jej wartość zmienia się przy każdym obiegu pętli, począwszy od *wartości początkowej* (tu: 1), aż do *wartości końcowej* (10). *Krok*, czyli skok wartości w tym przykładzie wynosi 1 i dlatego został pominięty, chociaż równie dobrze można napisać:

```
for i=1:1:10
....
end
```



Krok może mieć wartość dodatnią:

```
for k=1:2:10
   disp(k)
end
```

... i ujemną, a wtedy wartość zmiennej sterującej w kolejnych wykonaniach pętli się zmniejsza:

```
for licz=4:-1:1
   disp(licz)
end
```

Uwagi:

Instrukcja `disp(k)` powoduje wypisanie wartości zmiennej `k` na konsoli. Można równocześnie wypisać większą liczbę zmiennych, oddzielając je przecinkiem (ale będą wypisywane ‘od końca’), można też wypisywać teksty (mądrze mówi się ‘*łańcuchy znakowe*’).

Popatrzmy na efekt instrukcji:

```
disp('zmienna licz '+string(licz))
```

Bardzo często zmienna sterująca jest powiązana z indeksami elementów wektorów pozwalając na ich systematyczne przetwarzanie/przeglądanie. Zacznijmy od takich przykładów:

```
x=linspace(1,10,10);
for i=1:10
   disp (x(i))
end
for i=10:-1:1
   disp(x(i))
end
```
albo
```
for i=1:2:10
   disp(x(i))
end
```

Przykład: Wyznaczyć 20 początkowych wyrazów ciągu zdefiniowanego wzorem:

$$\begin{cases} u_1 = 1 \\ u_{i+1} = 2u_i - 5 \end{cases}$$

`u(1)=1;`	Inicjalizacja. Od czegoś trzeba zacząć. Koniecznie wykonywana PRZED rozpoczęciem pętli.
`for i=1:1:19`	
`  u(i+1)=2*u(i)-5;`	Przy każdym obiegu pętli tworzony jest kolejny element wektora u. Począwszy od u(2)=2*u(1)-5=-3, u(3)=2*u(2)-5=-11, ...)
`end`	

Zadanie: Wyznaczyć 10 pierwszych elementów ciągu Fibonacciego.

Pętla while

Innym rodzajem pętli jest *pętla while*, uzależniająca wykonanie ciągu instrukcji od spełnienia warunku.

```
h=0;
while h<7
  h=h+1;
  disp (h)
end;
```

Frazę `while h<7` czytamy: '*dopóki spełniony jest warunek h<7 wykonuj*...'. Zauważmy, że gdyby wartość zmiennej `h` nie byłamodyfikowana wewnątrz pętli, to pętla ta nie miałaby szansy się zakończyć.

Za pomocą pętli `while` bardzo łatwo można zaprogramować symulację gry w kości z Pimpkiem, która ma się toczyć do określonej liczby zwycięstw (a zatem w góry nie wiadomo, ile rzutów będzie wykonanych). Jednak... brakuje nam jeszcze jednej instrukcji: *instrukcji warunkowej*. O tym opowiemy w kolejnym rozdziale.