Pętle – wielokrotne wykonywanie ciągu instrukcji.

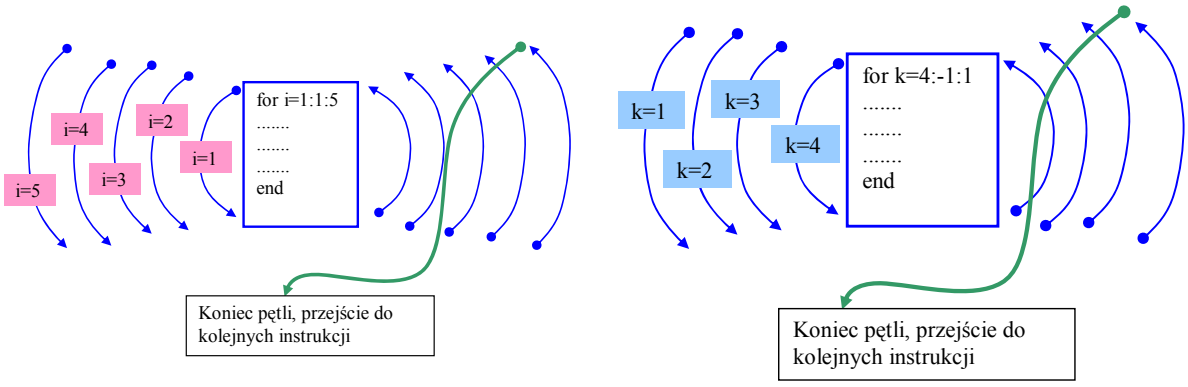
Bardzo często w programowaniu wykorzystuje się wielokrotne powtarzanie określonego ciągu czynności (instrukcji).
Rozróżniamy sytuacje, gdy liczba powtórzeń jest znana – oraz gdy liczba powtórzeń zależy od konkretnej sytuacji napotkanej w toku obliczeń.

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*'). Warto popatrzeć też 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:

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

```
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;
end
```
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, ...)

Zadanie: Wyznaczyć 10 pierwszych elementów ciągu Fibonacciego.

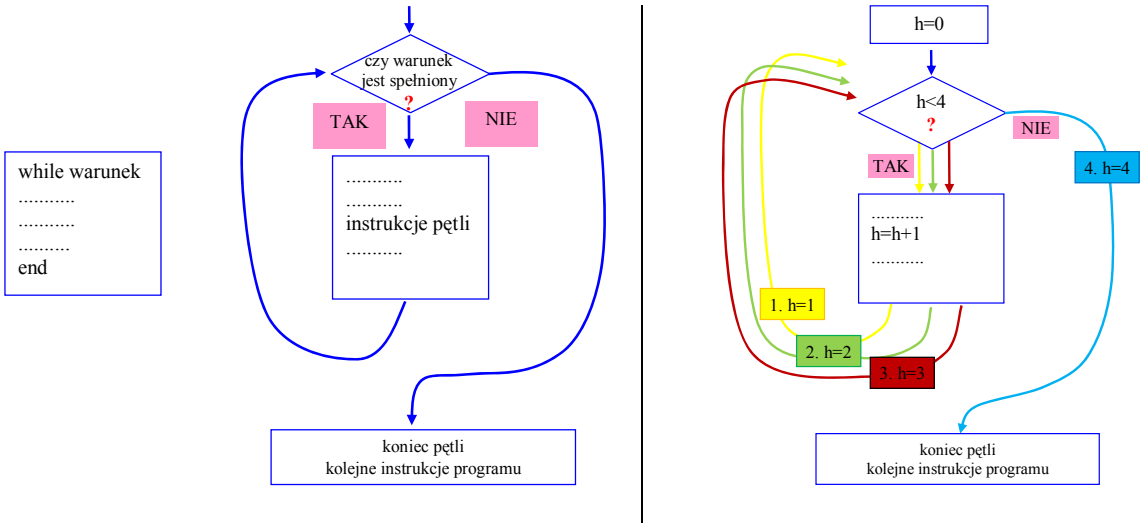
Zadanie: Obliczyć wartość 6!

Pętla while

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

```
h=0;
while h<4
  h=h+1;
  disp (h)
end;
```

Frazę `while h<4` czytamy: '*dopóki spełniony jest warunek h<4 wykonuj...*'. Zauważmy, że gdyby wartość zmiennej `h` nie była modyfikowana wewnątrz pętli, to pętla ta nie miałaby szansy się zakończyć.



Uwaga na marginesie:
Jak już było wspomniane, instrukcja `disp(h)` powoduje wypisanie wartości zmiennej `h` na konsoli. Tu wykorzystujemy tę instrukcję jako przykładową 'zawartość' pętli.

Warunki, wyrażenia logiczne i instrukcja warunkowa

Rozważmy warunek: `h<7`. W zależności od wartości zmiennej `h` warunek ten ma wartość *prawda* (ang. *true*) albo *fałsz* (ang. *false*). Inne możliwości nie ma. Warunek: `h<7` jest przykładem *wyrażenia logicznego*.

Wyrażenia logiczne
Wyrażenie logiczne przyjmuje wartość *prawda* lub *fałsz*.
W wyrażeniu logicznym występują *operatory relacji*:

==	równy
<>	różny
> >=	większy, większy równy
< <=	mniejszy, mniejszy równy

np: `h<7, 5<>2, a==b, c^2==a^2+b^2, y<2*x+5`

W wyrażeniach logicznych występują *operatory logiczne*:

&	„i" (*and*) koniunkcja
|	„lub" (*or*) alternatywa
~	„nie" (*not*) negacja

np: `a>b & c>=10` wyrażenie ma wartość prawda jeśli spełnione jest `a>b` oraz `c>=10`
`x<5 | x>10` wyrażenie ma wartość prawda jeśli `x<5` lub `x>10` (pytanie: a jak inaczej można to zapisać?)

W szczególności wyrażeniami logicznymi są `%t` i `%f` – specjalne zmienne Scilaba, o wartościach równych odpowiednio prawda i fałsz.

Definiowanie warunków jest niezbędne do wykorzystania chyba najważniejszej instrukcji, jaką jest instrukcja warunkowa *if*.

Instrukcja warunkowa

Instrukcja warunkowa pozwala realizować określone fragmenty programu (ciąg instrukcji) w zależności od spełnienia (lub nie) warunku i ma postać (zaczynamy od najprostszej):

```
if warunek then
...
...
end
```

```
if a>5 then
   z=z*2
end
```

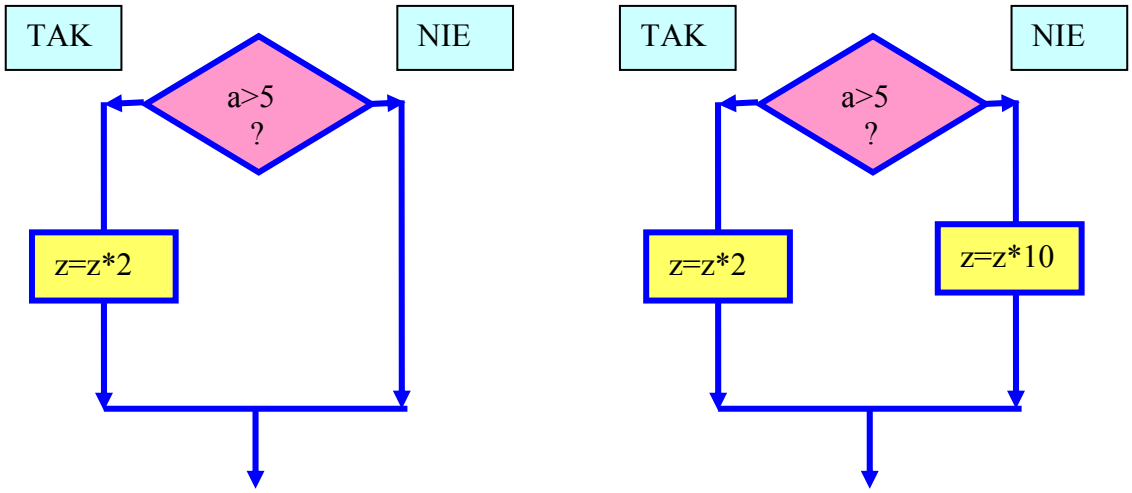
Jeśli spełniony jest warunek `a>5` wykonana zostanie instrukcja `z=z*2`; jeśli warunek nie jest spełniony, instrukcja nie zostanie wykonana.

Kolejna postać instrukcji warunkowej:

```
if warunek then
...
else
...
end
```

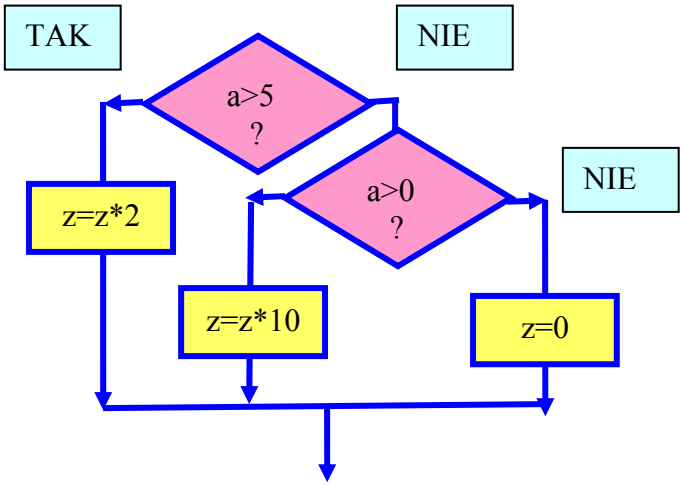
```
if a>5 then
   z=z*2
else
   z=z*10
end
```

Jeśli spełniony jest warunek `a>5` wykonana zostanie instrukcja `z=z*2`; jeśli warunek nie jest spełniony („w przeciwnym przypadku"), wykonana zostanie instrukcja `z=z*10`



Instrukcję **`if`** można jeszcze bardziej rozbudować, sprawdzając sekwencję warunków:

```
if warunek then
...instrukcje
elseif warunek1 then
...instrukcje1
elseif warunek2 then
...instrukcje2
else
...
end
```




```
if a>5 then
     z=z*2
elseif a>0 then
     z=z*10
else
     z=0
end
```

Jeśli spełniony jest warunek „warunek” wykonany jest ciąg instrukcji „instrukcje”. W przeciwnym przypadku i o ile spełniony jest warunek „warunek1” wykonany jest ciąg instrukcji „instrukcje1”. W przeciwnym przypadku (czyli jeśli nie jest spełniony warunek „warunek” i nie jest spełniony warunek „warunek1”) i jeśli spełniony jest warunek „warunek2” realizowany jest ciąg instrukcji „instrukcje2”, itd. Instrukcje znajdujące się po frazie „else” wykonywane są tylko wtedy, jeśli nie został spełniony żaden z występujących wcześniej warunków.

Powrót do pętli while – i nie tylko

Zakończenie pętli `while` bardzo często jest sterowane za pomocą instrukcji `if`.
Na przykład:

```
i=0;
while %t                  taka pętla, o ile jej jakoś nie przerwiemy, wykonywałaby się bez
                          końca – bowiem wartością wyrażenia logicznego %t zawsze jest
                          prawda
   i = i + 1;             W każdym obiegu pętli zwiększa się wartość zmiennej i
  x(i) = 2*i;             Wyznacza się wartość kolejnego (i-tego) elementu wektora x.
     if  i > 10 then      Jeśli i > 10 to:
     break                następuje zakończenie (przerwanie) wykonywania pętli
   end
end
```

Uwagi:

- Instrukcja warunkowa `if` jest zagnieżdżona wewnątrz instrukcji pętli.
- Można zagnieżdżać w sobie kolejne instrukcje warunkowe.
- Można też zagnieżdżać pętle – czyli umieszczać wewnątrz jednej pętli kolejną pętlę, i tak dalej. Koniecznie trzeba używać w takim przypadku innych zmiennych sterujących.
- „Wcinanie’ zagnieżdżonych instrukcji zwiększa czytelność kodu.

Uwagi inne:

Trzeba uważać na porównania, czyli operator ==! W przypadku porównywania wartości całkowitych nie powinno być niespodzianek, natomiast w przypadku wartości rzeczywistych należy uwzględnić wpływ arytmetyki komputerowej, czyli zaokrąglenia.

Przykład: Twierdzenie Pitagorasa

Dany jest trójkąt prostokątny, powstały z przecięcia przekątną kwadratu o boku równym 2. Jest to trójkąt prostokątny, o bokach równych odpowiednio 2, 2 (przyprostokątne) i $\sqrt{2}\cdot 2$ (przeciwprostokątna). A zatem powinno zachodzić twierdzenie Pitagorasa. Sprawdzamy to w Scilabie:

```
a=2;
b=2;
c=sqrt(2)*2;
pitagoras=a^2+b^2==c^2
```

Ostatnia instrukcja podstawienia jest pozornie dosyć ‘dziwaczna”. Jej prawa strona to *wyrażenie logiczne* – porównujemy sumę kwadratów przyprostokątnych i kwadrat przeciwprostokątnej. Wynikiem jest fałsz! Czy prawo Pitagorasa nie jest spełnione?
Sprawdźmy to inaczej obliczając:

```
error=c^2-a^2-b^2
```

Otrzymuje się:

```
error  =
     0.00000000000000178
```

Albo jeszcze inaczej:

```
-->a^2+b^2
 ans  =
    8.
-->c^2
 ans  =
    8.00000000000000178
```

Przyczyną zamieszania są błędy zaokrąglenia związane z wyznaczeniem $\sqrt{2}\cdot 2$.

Zadanie:

Utworzyć wektor x złożony ze 100 elementów o wartościach losowych z przedziału [0, 1).

```
x=rand(100,1);
```

1) Narysować wykres.
2) Obliczyć ile elementów wektora x ma wartości większe od 0.7

```
ile = 0
for i=1:100
      if x(i) > 0.7 then
            ile = ile + 1;
      end
end
disp (‘Liczba elementow wiekszych od 0.7=’+string(ile))
```

3) Obliczyć ile elementów wektora x ma wartości mniejsze od 0.5.
4) Utworzyć wektor y poprzez zastąpienie elementów wektora x większych od 0.6 wartością 0.6. Narysować wykres.

```
y=x;
for i=1:100
      if x(i)>0.6 then
            y(i)=0.6;
      end
end
```