WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

Sprawozdanie Laboratoryjne

Języki i paradygmaty programowania

Prowadzący:

mgr inż. Ewa Żesławska

Wykonał: Wojciech Nycz 69823

Rzeszów 2026r

1. LAB 1 **2**
2. LAB 2 **11**
3. LAB 3 **19**
4. LAB 4, 5, 6 **25**

1. LAB 1

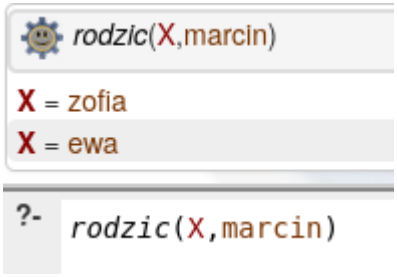
1. Zaproponuj bazę wiedzy dotycząca rodziny w postaci faktów: rodzic/2, kobieta/1, mężczyzna/1.

```
rodzic(marcin, ania).
rodzic(zofia, marcin).

rodzic(andrzej, jan).
rodzic(jan, mikołaj).

rodzic(ewa, kasia).
rodzic(ewa, marcin).

kobieta(ania).
kobieta(zofia).
kobieta(ewa).
kobieta(kasia).

mężczyzna(marcin).
mężczyzna(jan).
mężczyzna(andrzej).
mężczyzna(mikołaj).
```

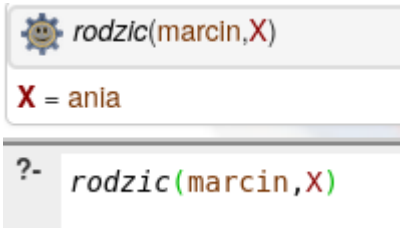
2. Następnie zaproponuj reguły (jedno- i dwuetapowe) potomek(Y,X) (Y – potomek X), matka(X,Y),dziadkowie(X,Z), siostra(X,Y) (z warunkiem X \= Y)oraz sprawdź wyniki dla zapytań:
    - ?- potomek(ania, marcin).
    - ?- matka(zofia, marcin).
    - ?- dziadkowie(andrzej, mikołaj).
    - ?- siostra(kasia, marcin).

```
potomek(Y,X) :-
    rodzic(X, Y).

matka(X,Y) :-
    rodzic(X, Y),kobieta(X).

dziadkowie(X,Z):-
    rodzic(X,Y),rodzic(Y,Z).

siostra(X,Y):-
    kobieta(X),
    rodzic(R,X),
    rodzic(R,Y),
    X \= Y.
```

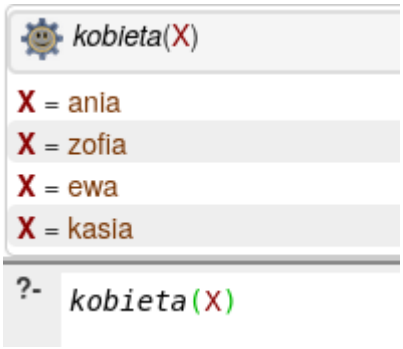
3. Zadaj pytania z wykorzystaniem zmiennych, aby uzyskać listę wszystkich:
    - Rodziców Marcina



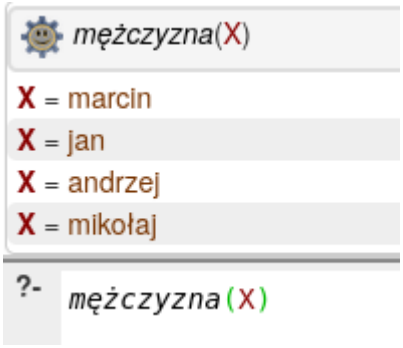
    - Dzieci Marcina



    - Kobiet w bazie

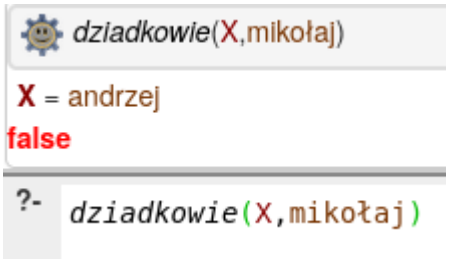


    - Mężczyzn w bazie

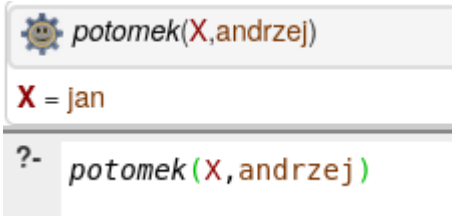


4. Zapisz zapytanie, które pozwoli uzyskać wszystkie pary (rodzic, dziecko) w bazie. Podpowiedź: rodzic(X, Y).

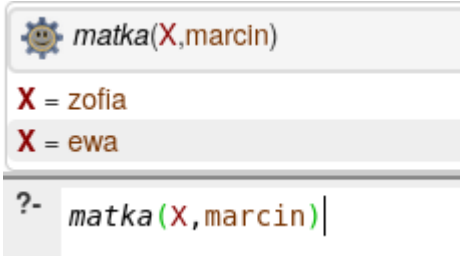
5. Ustal, kto jest dziadkiem lub babcią Mikołaja, używając dziadkowie/2



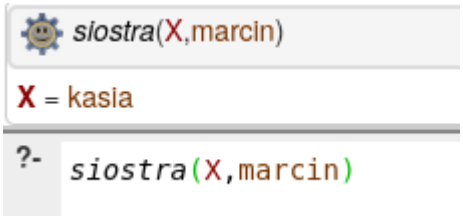
6. Ustal, kto jest potomkiem Andrzeja (czyli wszystkie osoby, które pochodzą od Andrzeja).



7. Kto jest matką Marcina?



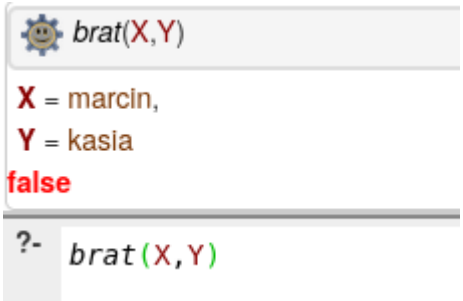
8. Kto jest siostrą Marcina?



9. Czy Andrzej ma jakiekolwiek dzieci.

10. Dodaj do bazy własną regułę brat(X, Y), która będzie prawdziwa, jeśli:
    - X i Y mają wspólnego rodzica,
    - X jest mężczyzną,
    - X i Y to różne osoby.

```
brat(X,Y):-
    rodzic(R,X),
    rodzic(R,Y),
    mężczyzna(X),
    X \= Y.
```



Zadanie 2
Zaprojektuj i przetestuj bazę wiedzy opisującą sposoby spędzania wolnego czasu, a następnie zdefiniuj reguły rekomendacyjne i odpowiedz na pytania w Prologu.

1. Utwórz min. 10–15 faktów, korzystając z co najmniej 4 różnych predykatów:
    - lubi(Osoba, Aktywność).
    - intensywność(Aktywność, Poziom). % np. niski/średni/wysoki
    - miejsce(Aktywność, Lokalizacja). % np. indoor/outdoor/miasto/natura
    - koszt(Aktywność, ZlNaGodz). % liczba, np. 0, 15, 30
    - pora(Aktywność, PoraDnia). % rano/popołudnie/wieczór/weekend
    - wiek(Osoba, Lata). % przyda się do reguł
    - inne …. Np. np. towarzyska/1, samotna/1, sprzęt(Aktywność, Wymaga)

```
lubi(ania, bieganie).
lubi(ania, czytanie).
lubi(bartek, bieganie).
lubi(magda, joga).
lubi(jan, chodzenie).

intensywność(bieganie, wysoki).
intensywność(joga, niski).
intensywność(czytanie, niski).
intensywność(chodzenie, średni).

miejsce(bieganie, na_zewnątrz).
miejsce(joga, wewnątrz).
miejsce(czytanie, wewnątrz).

koszt(bieganie, 5).
koszt(joga, 10).
koszt(czytanie, 0).

pora(bieganie, rano).
pora(joga, rano).
pora(czytanie, wieczór).

wiek(ania, 25).
wiek(jan, 40).
```

2. Zaproponuj 5 reguł dla zdefiniowanych faktów np.: klasyfikacja aktywności, dopasowanie osobowe i kontekstowe, wspólne zainteresowania, itp.

```
%zainteresowania tańsze niz K
tanie(A):-
    koszt(A,K),
    K =< 5.

%wspólne zainteresowania
wspólne(O1,O2,A):-
    lubi(O1,A),
    lubi(O2,A).

%aktywności o niskim poziomie
relaks(A):-
    intensywność(A,niski).

%aktywności na zewnątrz
na_zewn(A):-
    miejsce(A, na_zewnątrz).

%pasuuje dla osoby o budżecie =<15
pasuje_dla(Os,A):-
    lubi(Os, A),
    koszt(A,K),
    K=<15.
```

3. Wypisz informacje o:
    - Lista aktywności na zewnątrz: ?- na_zewnatrz(A).
    - Tanie aktywności (≤10 zł/h): ?- tania(A).
    - Co pasuje danej osobie wieczorem w budżecie 15 zł/h? (załóż fakty lubi/2, pora/2) ?-pasuje_dla(Osoba, A), pora(A, wieczor), koszt(A, K), K =< 15.
    - Wspólne zainteresowania dwóch osób: ?- wspólne(ania, bartek, A).
    - Aktywności relaksujące w naturze: ?- relaks(A), miejsce(A, natura).
      Propozycje dla osoby dorosłej na weekend

2. LAB 2

Zadanie 1

Sprawdź działanie oraz wysuń wnioski.

1. Sprawdzić działanie:
   - ?- X is 2 + 2.
   - ?- Y is 2.5 + (4 / 2).
   - ?- Z is 2 + 0.001.

```
X is 2+2
X = 4
Y is 2.5 + ( 4 / 2)
Y = 4.5
Z is 2 + 0.001
Z = 2.001
```

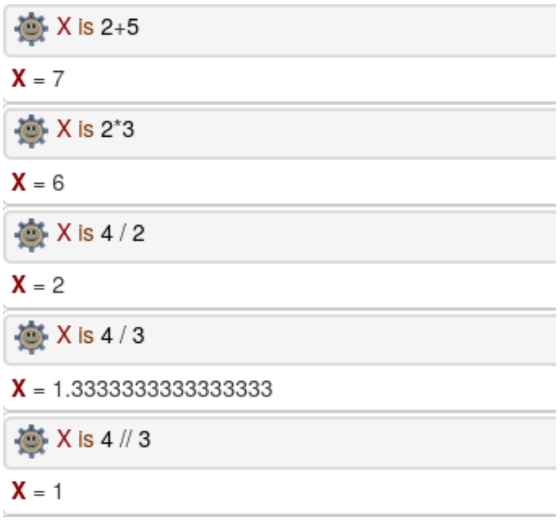
   Is działa z liczbami całkowitymi i z rzeczywistymi, oraz zwraca uwagę na kolejność wykonywania obliczeń

2. Uwaga:
   - ?- A is 3.
   - ?- B is A + 4.
   - ?- A is 3, B is A + 4.

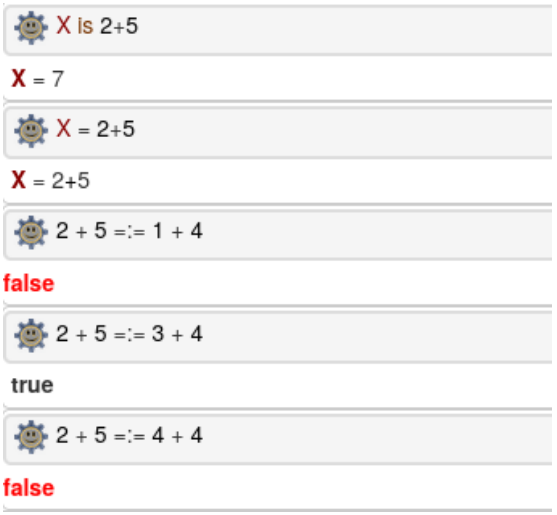
```
A is 3
A = 3
B is A + 4
Arguments are not sufficiently instantiated
In:
   [1] _2294 is _2300+4
A is 3, B is A+4
A = 3,
B = 7
```

   is wymaga aby wszystkie zmienne po prawej były już obliczone(znane)

3. Operacje arytmetyczne:
    - ?- X is 2 + 2.
    - ?- X is 2 * 3.
    - ?- X is 4 / 2.
    - ?- X is 4 / 3.
    - ?- X is 4 // 3.

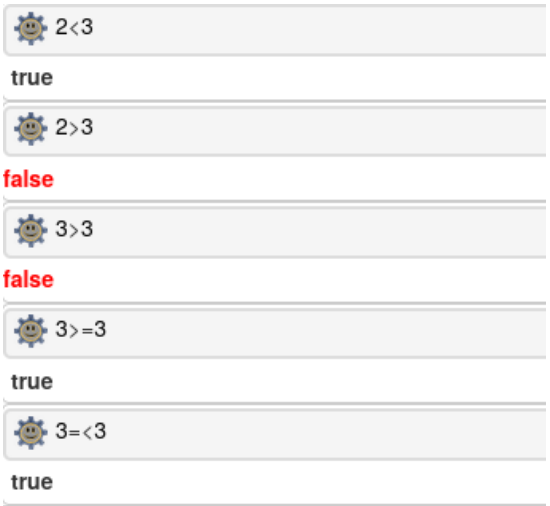


4. Uwaga na „podstawianie":
    - ?- X is 2 + 5.
    - ?- X = 2 + 5.
    - ?- 2 + 5 =:= 1 + 4.
    - ?- 2 + 5 =:= 3 + 4.
    - ?- 2 + 5 =:= 4 + 4.



is służy do operacji matematycznych gdzie = służy do przypisania, =:= służy do porównywania obliczonych wyrażeń

5. Przećwiczyć użycie operatorów:
    - ?- 2 < 3.
    - ?- 2 > 3.
    - ?- 3 > 3.
    - ?- 3 >= 3.
    - ?- 3 =< 3.



Zadanie 2

Zdefiniuj predykaty:

- wieksza(X, Y) – wypisz, która liczba jest większa.
- maksimum(X, Y, M) – zwraca większą liczbę.
- czy_parzysta(X) – sprawdza, czy liczba jest parzysta

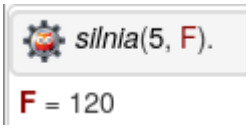
```
wieksza(X,Y):-
    X > Y,
    write(X), write(' wieksze niż '),write(Y).

wieksza(X,Y):-
    Y > X,
    write(Y), write(' wieksze niż '),write(X).

wieksza(X,Y):-
    X =:= Y,
    write('liczby są równe').

maks(X,Y,X):- X >= Y.
maks(X,Y,Y):- Y > X.

czyParz(X):-
    0 is X mod 2,
    write('Liczba którą podałeś jest parzysta').

czyParz(X):-
    1 is X mod 2,
    write('Liczba którą podałeś nie jest parzysta').
```

Zadanie 3

Zaproponuj implementację w prologu dla poniższych podstawowych algorytmów:

- Silnia (rekurencyjnie)

```
silnia(0, 1).
silnia(N, F) :-
    N > 0,
    N1 is N - 1,
    silnia(N1, F1),
    F is N * F1.
```



- Suma liczb od 1 do N

```
suma_do_n(0, 0).
suma_do_n(N, S) :-
    N > 0,
    N1 is N - 1,
    suma_do_n(N1, S1),
    S is N + S1.
```

```
suma_do_n(5, S).
```

S = 15

- N-ty wyraz ciągu Fibonacciego

```
fib(0, 0).
fib(1, 1).
fib(N, F) :-
    N > 1,
    N1 is N - 1,
    N2 is N - 2,
    fib(N1, F1),
    fib(N2, F2),
    F is F1 + F2.
```

```
fib(7, F).
```

F = 13

- Największy wspólny dzielnik (NWD, algorytm Euklidesa)

```
nwd(A, 0, A) :- A > 0.
nwd(A, B, D) :-
    B > 0,
    R is A mod B,
    nwd(B, R, D).
```

```
nwd(24, 36, D).
```

D = 12

Zadanie 4

Utworzy bazę wiedzy o produktach w sklepie: produkt(jablko, 2.50, 10). % nazwa, cena za szt., ilość na stanieipd…

Następnie zdefiniuj predykaty

- wartosc_produktu(Nazwa, W) — wylicz W = cena * ilosc dla pojedynczego produktu
- wartosc_magazynu(W) - zsumuj konkretne, znane pozycje
- drogi_produkt(N) — wypisz produkty o cenie > 4

```
produkt(jabłko, 2.50, 10).
produkt(banan, 3.00, 5).
produkt(pomarańcza, 4.50, 8).
produkt(marchew, 1.20, 15).
produkt(ziemniak, 2.00, 20).
produkt(truskawka, 5.00, 7).
produkt(mleko, 3.50, 12).
produkt(chleb, 4.20, 6).

wartość_produktu(Nazwa, W) :-
    produkt(Nazwa, Cena, Ilosc),
    W is Cena * Ilosc.

wartość_magazynu(W) :-
    findall(V, (produkt(_, C, I), V is C*I), Lista),
    sum_list(Lista, W).


drogi_produkt(Nazwa) :-
    produkt(Nazwa, Cena, _),
    Cena > 4.
```

Zadanie 5

Zaproponuj bazę o wynagrodzeniach w firmie w postaci: pracownik(jan, 4800, 500). % imie, podstawa, premia ipd…

Następnie zdefiniuj predykaty:

- brutto(X, B) — suma podstawy i premii
- podatek17(X, T) — 17% od brutto
- netto(X, N) — brutto minus podatek
- czy_zarabia_wiecej(X, Y, Kto) — porównanie dwóch osób

```
%pracownik(imie, podstawa, premia)
pracownik(jan, 4800, 500).
pracownik(ania, 5200, 600).
pracownik(bartek, 4500, 700).
pracownik(magda, 6000, 800).
pracownik(kasia, 4000, 300).

brutto(Imie, B):-
    pracownik(Imie, Podstawa, Premia)
    B is Podstawa + Premia.

podatek17(Imie, T) :-
    brutto(Imie, B),
    T is B * 0.17.

netto(Imie, N) :-
    brutto(Imie, B),
    podatek17(Imie, T),
    N is B - T.

czy_zarabia_wiecej(X, Y, X) :-
    netto(X, NX),
    netto(Y, NY),
    NX > NY.

czy_zarabia_wiecej(X, Y, Y):-
    netto(X, NX),
    netto(Y, NY),
    NX < NY.

czy_zarabia_wiecej(X, Y, rowne) :-
    netto(X, NX),
    netto(Y, NY),
    NX =:= NY.
```

brutto(jan,B)

B = 5300

podatek17(ania, T)

T = 986.0000000000001

netto(bartek, N)

N = 4316.0

czy_zarabia_wiecej(ania, jan, W).

W = ania

Next | 10 | 100 | 1,000 | Stop

czy_zarabia_wiecej(jan, kasia, Kto)

Kto = jan

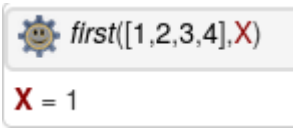
Next | 10 | 100 | 1,000 | Stop

3. LAB 3

Ćwiczenie 1
Napisz predykaty:

- first(List, X). – zwraca pierwszy element.

```
first([X|_], X)
```



- last(List, X). – znajduje ostatni element.

```
last([X],X).
last([_|T], X):-
    last(T,X).
```

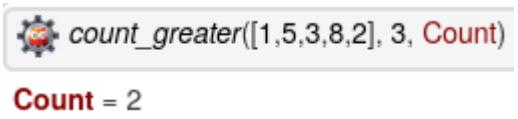
- sum_list(List, Sum). – suma elementów listy.

```
sum_list([], 0).
sum_list([H|T], Sum):-
    sum_list(T, Sum1),
    Sum is H + Sum1.
```

a następnie zaproponuj przykładowe zapytania

Ćwiczenie 2
Napisz predykat count_greater(List, N, Count) – policz elementy > N.

```
count_greater([], _, 0).
count_greater([H|T], N, Count):-
    H > N,
    count_greater(T, N, Count1),
    Count is Count1 + 1.

count_greater([H|T], N, Count):-
    H =< N,
    count_greater(T, N, Count).
```



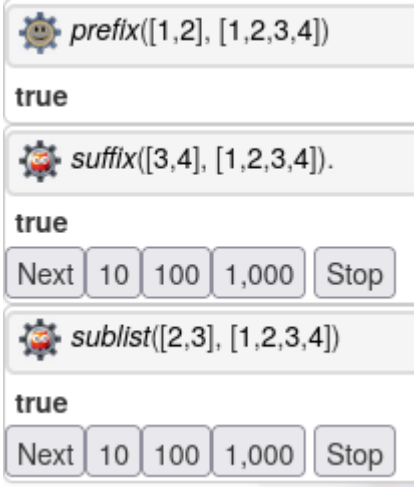
Ćwiczenie 3

Zaimplementuj predykaty związanie z podlistami.

- prefix/2
- suffix/2
- sublist/2

Następnie przetestuj powyżej.

```
prefix(P, L):-
    append(P, _, L).

suffix(S, L):-
    append(_, S, L).

sublist(Sub, L):-
    append(_, L2, L),
    append(Sub, _, L2).
```



Ćwiczenie 4

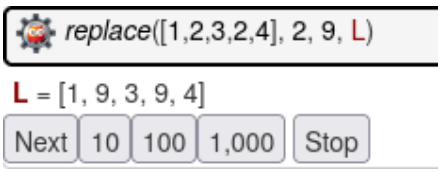
Zaproponuj implementacje dla:

- Odbuduj własną funkcję maximum, ale z akumulatorem.
- Napisz wariant sum_list_acc.
- Napisz wariant product_list_acc

Ćwiczenie 5

Napisz predykat: replace(List, X, Y, NewList) – zamień wszystkie wystąpienia X na Y.

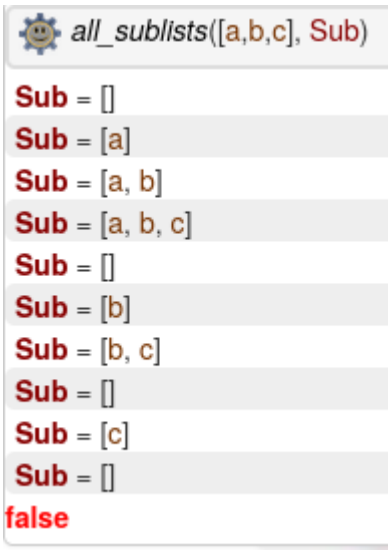
```
replace([], _, _, []).
replace([X|T], X, Y, [Y|T2]) :-
    replace(T, X, Y, T2).
replace([H|T], X, Y, [H|T2]) :-
    H \= X,
    replace(T, X, Y, T2).
```



Ćwiczenie 6

Napisz predykat generujący wszystkie podlisty: all_sublists(List, Sub).

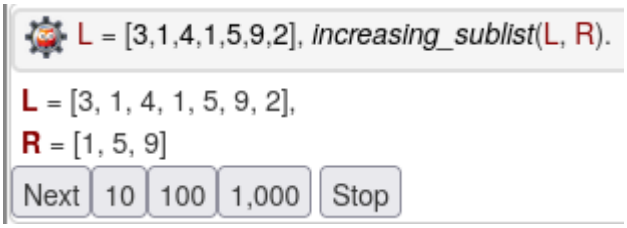
```
all_sublists(List, Sub) :-
    append(_, Rest, List),
    append(Sub, _, Rest).
```



Ćwiczenie 7

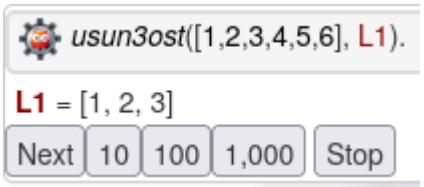
Dla listy: L = [3,1,4,1,5,9,2] napisz predykat: increasing_sublist(L, R) – znajdź najdłuższą rosnącą podlistę ciągłą.

```
is_increasing([]).
is_increasing([_]).
is_increasing([X,Y|T]) :-
    X < Y,
    is_increasing([Y|T]).

sublists(Sub, List) :-
    append(_, Rest, List),
    append(Sub, _, Rest).

longest([L], L).
longest([H1,H2|T], Max) :-
    length(H1, Len1),
    length(H2, Len2),
    (Len1 >= Len2 -> longest([H1|T], Max)
                   ; longest([H2|T], Max)).

increasing_sublist(List, MaxSub) :-
    findall(Sub, (sublists(Sub, List), is_increasing(Sub)), Subs),
    longest(Subs, MaxSub).
```



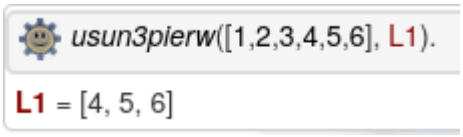
Zadanie 1
Zdefiniować predykat, powodujący usunięcie 3 ostatnich elementów listy L, w wyniku powstaje lista L1, użyć sklej.

```
usun3ost(L,L1):-
    append(L1, [_,_,_],L).
```



Zadanie 2
Zdefiniować predykat, powodujący usunięcie 3 pierwszych elementów listy L, w wyniku powstaje lista L1, użyć sklej.

```
usun3pierw(L, L1) :-
    append([_, _, _], L1, L).
```



Zadanie 3
Zdefiniować predykat, powodujący usunięcie 3 pierwszych i ostatnich elementów listy L, w wyniku powstaje lista L2, użyć sklej.

```
usun3pierw_i_ost(L, L2) :-
    append([_, _, _], Srodek, L),
    append(L2, [_, _, _], Srodek).

usun3pierw_i_ost([1,2,3,4,5,6,7], L1).
```

Zadanie 4
Zdefiniować parę komplementarnych predykatów nieparzysta(L) oraz parzysta(L) sprawdzających czy argument jest listą o odpowiednio nie/parzystej długości. Czy Twój predykat potrafi również utworzyć listę o zadanej parzystości? (jako argument podajemy niewiadomą, a nie stałą)

```
parzysta([]).
parzysta([_, _ | T]) :-
    parzysta(T).

nieparzysta([_]).
nieparzysta([_, _ | T]) :-
    nieparzysta(T).
```

parzysta([a,b,c,d])

true

parzysta([a,b,c])

false

nieparzysta([a,b,c])

true

Next | 10 | 100 | 1,000 | Stop

nieparzysta([a,b,c,d])

false

4. LAB 4, 5, 6

Rozwiązania z laboratoriów 4, 5 i 6 znajdują się na repozytorium github pod linkiem: https://github.com/Wojecki/JIPP_69823 wraz z plikami źródłowymi z poprzednich laboratoriów.