Zbiory pozwalają modelować wiele interesujących zagadnień (nie tylko matematycznych), ale ich implementacja programowa może napotkać szereg ograniczeń związanych z używanym językiem programowania.

W popularnym niegdyś języku Pascal można było napotkać intuicyjne definicje zbliżone do:

```
type Litery = 'A' .. 'Z';
ZbiorLiter = set of Litery;
var Alfabet:ZbiorLiter;
c: char;
begin
Alfabet:=['A'..'Z'];
read(c);
if c in Alfabet then {itd.}
end.
```

Oczywiście to, co dla komputera jest zbiorem, wcale nim nie jest dla matematyka z uwagi na wymóg jednakowego typu zapamiętywanych elementów.

Jednak do podstawowych zastosowań informatyczna realizacja zbiorów podobna do tych znanych z Pascala nadaje się znakomicie, gdyż możliwe jest np. wykonywanie operacji typu: dodawanie elementu do zbioru, mnożenie (iloczyn) zbiorów, odejmowanie zbiorów, sprawdzanie, czy obiekt należy do zbioru.

Spróbujmy zrealizować przykładową klasę, która będzie zachowywała się jak zbiór danych. Załóżmy, że w komputerze występuje tylko prosty alfabet znakowy obejmujący 26 znaków (A... Z). Do zasymulowania zbioru takich znaków wystarczy wówczas najzwyklejsza tablica rejestrująca dodanie do zbioru określonego znaku identyfikowanego przez indeksy od 0 (dla A) do 25 (dla Z), tak jak w przykładzie poniżej:

Popatrzmy, jak może wyglądać przykładowa realizacja listy dwukierunkowej (*ZbiorLitery.py* w katalogu *MojeTypy*).

```
class ZbiorLitery: # Przykładowa implementacja klasy 'ZbiorLitery' (zbiór liter z zakresu A...Z)
def __init__(self):
self.zbior=[False for x in range (26)]
```

Niech wpis *True* w tablicy na określonej pozycji oznacza obecność w tym zbiorze znaku indeksowanego przez dany indeks (0 to *A*, 1 to *B* itd. aż do 25 oznaczającego *Z*).

Pomimo dużej prostoty powyższa implementacja umożliwia już manipulacje typowe dla zbiorów, łatwo jest np. zrealizować dodawanie elementu do zbioru lub utworzenie zbioru jako sumy dwóch innych. Do dalszych operacji potrzebna nam będzie prosta funkcja konwertująca znak na indeks tablicy:

```
# Funkcja pomocnicza
def do_indeksu(c):                    # Konwersja znaku 'c' na wartość 0...25 dla liter z zakresu A...Z
    if (c <= 'Z') or (c >= 'a') and (c <= 'z')):
        return ord(c.upper())-65
    else:
        return -1                     # Błąd zakresu (znak spoza alfabetu)
```


Popatrzmy teraz na realizacje kilku przykładowych metod klasy `ZbiorLitery`:

```
def dodaj(self, c):                   # Dodaj znak 'c' do zbioru
    i = do_indeksu(c)
    if i in range(26):
        self.zbior[i]=True
    else:
        print("[Błąd] Znak spoza dozwolonego alfabetu:", c, i)

def __add__(self, x2):                # Ta metoda zwraca obiekt będący sumą dwóch zbiorów
    suma = ZbiorLitery()              # w operacji x1+x2, gdzie x1 jest bieżącym obiektem (self)
    for i in range (26):
        suma.zbior[i]= self.zbior[i] or x2.zbior[i]
    return suma
```

Usuwanie elementu jest równie proste jak wstawianie:

```
def usun(self, c):                    # Usuń znak 'c' ze zbioru
    i= do_indeksu(c)
    print("Usuwam", c.upper())
    if  i != -1 and self.zbior[i]==True:
        self.zbior[i]=False
    else:
        print(f"\n[Błąd] Znak nie należy do zbioru:", c)
```

W celu prezentacji zawartości proponuję następującą metodę:

```
def wypisz(self,s):                   # Wypisuje zawartość zbioru
    print(s,"= {", end=" ")
    for i in range(26):
    if self.zbior[i] == True:         # Wypisuje obecne elementy
        print( chr(i+65) + " ", end= " ") # Konwersja indeksu 0...25 na znaki A...Z
    print("}\n")
```

Poniżej znajdziesz przykładowy program testujący nasz nowy typ danych.

```
from MojeTypy import ZbiorLitery as z
z1=z.ZbiorLitery()
z2=z.ZbiorLitery()
z1.dodaj('A'), z1.dodaj('K'), z1.dodaj('K'), z1.dodaj('M')
z2.dodaj('B'), z2.dodaj('K'), z2.dodaj('R')
z1.wypisz("z1")
z2.wypisz("z2")
z1.usun('a')
z1.usun('X')
z1.wypisz("z1")
(z1+z2).wypisz("z1+z2")
```

Uruchomienie programu powinno spowodować wyświetlenie na ekranie następujących komunikatów:

```
z1 = { A  K  M  }
z2 = { B  K  R  }
```