```
>>> class Indexer:
...     data = [5, 6, 7, 8, 9]
...     def __getitem__(self, index):   # Wywoływany przy indeksowaniu i wycinaniu
...         print('getitem:', index)
...         return self.data[index]     # Realizuje dostęp po indeksie lub wycinanie
...
>>> X = Indexer()
>>> X[0]                                # Indeksowanie wysyła metodzie __getitem__ liczbę całkowitą
getitem: 0
5
>>> X[1]
getitem: 1
6
>>> X[-1]
getitem: -1
9
```

W przypadku operacji wycinania nasza metoda otrzymuje obiekt wycinka, który po prostu przekazujemy do osadzonego obiektu `data` w nowym wyrażeniu indeksowania:

```
>>> X[2:4]                              # Wycinanie wysyła metodzie __getitem__ obiekt wycinka
getitem: slice(2, 4, None)
[7, 8]
>>> X[1:]
getitem: slice(1, None, None)
[6, 7, 8, 9]
>>> X[:-1]
getitem: slice(None, -1, None)
[5, 6, 7, 8]
>>> X[::2]
getitem: slice(None, None, 2)
[5, 7, 9]
```

Metoda `__setitem__` implementuje analogiczny mechanizm przypisywania wartości zarówno dla indeksów, jak i dla wycinków — w tym ostatnim przypadku również otrzymuje obiekt wycinka, który można przekazać dalej tak samo jak w przypadku metody `__getitem__`.

```
def __setitem__(self, index, value):    # Przechwytuje przypisania do indeksu lub wycinka
    ...
    self.data[index] = value            # Przypisanie do indeksu lub wycinka
```

W rzeczywistości metoda `__getitem__` jest wywoływana również w innych kontekstach, co omówimy w następnym punkcie.

Iteracja po indeksie — __getitem__

Istnieje sztuczka, która może być trudna do odgadnięcia dla początkujących, ale okazuje się niezwykle użyteczna. Instrukcja `for` przy każdej iteracji odczytuje jeden element, wykorzystując kolejny indeks, licząc od zera, aż zostanie wywołany wyjątek końca zakresu. Z tego powodu metoda `__getitem__` może być jednym ze sposobów implementacji iteratora w Pythonie: jeśli pętla `for` wykryje tę metodę, będzie ją wywoływać z kolejnymi indeksami. Metoda `__getitem__` jest więc przykładem transakcji: „Kup jeden, drugi dostaniesz gratis” — każdy obiekt obsługujący indeksowanie potrafi również obsłużyć iterację:

```
>>> class stepper:
...     def __getitem__(self, i):
...         return self.data[i]
...
>>> X = stepper()                       # X jest instancją klasy stepper
```