Iteratory

Iterator - klasa, która dzięki definicji odpowiednich metod zapewnia możliwość użycia jej obiektów w kontekście iteracyjnym.

- Instrukcja for wywołuje niejawnie funkcję iter (definiowaną wewnątrz klasy jako metoda __iter__) na obiekcie klasy.
- Funkcja zwraca obiekt iteratora, czyli klasy definiującej metodę __next__, która zwraca określoną wartość (np. pojedynczy element kontenera). Metoda ta jest wywoływana - także niejawnie - w kolejnych iteracjach.
- Gdy zwrócone zostały wszystkie przewidziane wartości metoda __next__ zgłasza wyjątek StopIteration, który informuje pętlę for o zakończeniu (automatyczne obsłużenie wyjątku i zakończenie iteracji).
- Metoda __next__ może zostać wywołana jawnie za pomocą wbudowanej funkcji next.
- Zarówno metoda __iter__, jak i __next__ przyjmują tylko jeden parametr - self. Metody te mogą być umieszczone w tej samej, bądź w osobnych klasach, zapewniając nieco inną funkcjonalność (patrz poniższe przykłady).

Przykład I

```
class Test:
    def __init__(self, mi, ma):
        self.mn = mi
        self.mx = ma

    def __iter__(self):
        return self

    def __next__(self):
        self.mn += 1
        if self.mn < self.mx:
            return self.mn
        raise StopIteration


test=Test(0,5)
for elz in test:
    for elw in test:
        print(f'({elz},{elw})', end=' ')          #(1,2) (1,3) (1,4)
    print()

for elz in Test(0,5):
    for elw in Test(0,5):
        print(f'({elz},{elw})', end=' ')    #(1,1) (1,2) (1,3) (1,4)
    print()                                 #(2,1) (2,2) (2,3) (2,4)
                                            #(3,1) (3,2) (3,3) (3,4)
                                            #(4,1) (4,2) (4,3) (4,4)
```

Przykład II

```
class Test:
    def __init__(self, mi, ma):
        self.tmp=mi
        self.max=ma

    def __iter__(self):
        self.min=self.tmp
        return self

    def __next__(self):
        self.min+=1
        if self.min<self.max:
            return self.min
        raise StopIteration

test=Test(0,5)
for elz in test:
    for elw in test:
        print(f'({elz},{elw})', end=' ')  #(1,1) (1,2) (1,3) (1,4)
    print()

for elz in Test(0,5):
    for elw in Test(0,5):
        print(f'({elz},{elw})', end=' ')  #(1,1) (1,2) (1,3) (1,4)
     print()                               #(2,1) (2,2) (2,3) (2,4)
                                           #(3,1) (3,2) (3,3) (3,4)
                                           #(4,1) (4,2) (4,3) (4,4)
```

Przykład III

```
class Test:
    def __init__(self, mi, ma):
        self.min=mi
        self.max=ma

    def __iter__(self):
        return TestNext(self.min, self.max)

class TestNext:
    def __init__(self, mi, ma):
        self.min=mi
        self.max=ma

    def __next__(self):
        self.min+=1
        if self.min<self.max:
            return self.min
        raise StopIteration


test=Test(0,5)
for elz in test:
    for elw in test:
        print(f'({elz},{elw})', end=' ')
     print()

for elz in Test(0,5):
    for elw in Test(0,5):
        print(f'({elz},{elw})', end=' ')
     print()
```

```
#(1,1) (1,2) (1,3) (1,4)
#(2,1) (2,2) (2,3) (2,4)
#(3,1) (3,2) (3,3) (3,4)

#(1,1) (1,2) (1,3) (1,4)
#(2,1) (2,2) (2,3) (2,4)
#(3,1) (3,2) (3,3) (3,4)
#(4,1) (4,2) (4,3) (4,4)
```