이터레이터와 제너레이터

이터레이터와 제너레이터는 파이썬에서 반복을 지원하는 중요한 개념입니다. 이 둘은 유사한 점이 있지만, 몇 가지 중요한 차이점이 있습니다. 이제 이 차이점들을 자세히 살펴보겠습니다.

이터레이터

1. **정의**: 이터레이터는 값을 순차적으로 반환하는 객체입니다. 이터레이터는 `__iter__()`와 `__next__()` 메소드를 구현해야 합니다.

2. **구현 방법**:
- 이터레이터는 클래스를 사용하여 구현할 수 있습니다.
- `__iter__()` 메소드는 이터레이터 객체 자체를 반환합니다.
- `__next__()` 메소드는 다음 값을 반환하며, 더 이상 반환할 값이 없을 때 `StopIteration` 예외를 발생시킵니다.

3. **예제**:
```python
class MyIterator:
    def __init__(self, data):
        self.data = data
        self.index = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.index < len(self.data):
            result = self.data[self.index]
            self.index += 1
            return result
        else:
            raise StopIteration

my_iter = MyIterator([1, 2, 3])
for item in my_iter:
    print(item)
```

제너레이터

1. **정의**: 제너레이터는 `yield` 키워드를 사용하여 값을 반환하는 특별한 종류의 이터레이터입니다. 제너레이터 함수는 호출될 때 제너레이터 객체를 반환합니다.

2. **구현 방법**:
- 제너레이터는 함수나 표현식을 사용하여 구현됩니다.
- `yield` 키워드는 함수의 실행을 일시 중단하고 값을 반환하며, 이후에 함수의 상태를 유지하면서 다시 실행을 재개할 수 있습니다.
- 제너레이터는 자동으로 `__iter__()`와 `__next__()` 메소드를 구현합니다.

3. **예제**:
```python
def my_generator():
    yield 1
    yield 2
    yield 3

gen = my_generator()
for item in gen:
    print(item)
```

주요 차이점

1. 구현의 용이성:
- 이터레이터: `__iter__()`와 `__next__()` 메소드를 명시적으로 구현해야 합니다.
- 제너레이터: 함수와 `yield` 키워드를 사용하여 쉽게 구현할 수 있습니다.

2. 상태 유지:
- 이터레이터: 상태를 명시적으로 관리해야 합니다(예: 클래스 속성으로 인덱스를 관리).
- 제너레이터: 함수의 지역 변수가 상태를 자동으로 유지합니다.

3. 메모리 사용:
- 이터레이터: 모든 항목을 메모리에 저장할 필요는 없지만, 제너레이터만큼 효율적이지는 않을 수 있습니다.
- 제너레이터: 필요할 때마다 값을 생성하므로 메모리 사용이 매우 효율적입니다.

4. 코드 가독성:
- 이터레이터: 복잡한 이터레이터를 구현할 때 코드가 길어질 수 있습니다.
- 제너레이터: 코드가 간결하고 이해하기 쉽습니다.

요약

- 이터레이터: `__iter__()`와 `__next__()`를 구현하여 반복을 지원하는 객체입니다.
- 제너레이터: `yield` 키워드를 사용하여 쉽게 구현할 수 있으며, 필요할 때 값을 생성하고 함수의 상태를 자동으로 유지하는 특별한 이터레이터입니다.

제너레이터는 특히 큰 데이터셋을 처리하거나 복잡한 반복 로직을 간단하게 구현할 때 매우 유용합니다. 이터레이터는 보다 세부적인 제어가 필요할 때 사용될 수 있습니다.