- iterable : 순서가 있는 데이터 집합
- iterator : next 함수를 실행시켜서 값을 생성하는 값 생성기
- generator : iterator를 간단하게 구현한 문법

In [1]:

```
# 이터레이터 만들기
```

In [2]:

```
iterator = iter([1, 2, 3])
```

In [3]:

```
type(iterator)
```

Out[3]:

```
list_iterator
```

In [7]:

```
next(iterator)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 next(iterator)

StopIteration:
```

In [8]:

```
# 클래스로 이터레이터 만들기 : 피보나치 수열
```

In [9]:

```
class Fib:

    def __init__(self):
        self.prev = 0
        self.curr = 1

    def __iter__(self):
        return self

    def __next__(self):
        value = self.curr
        self.curr += self.prev
        self.prev = value
        return value
```

In [10]:

```
# 0 1 1 2 3 5 8 13 ..
```

In [11]:

```
fib = Fib()
```

In [24]:

```
next(fib)
```

Out[24]:

```
233
```

In [25]:

```
# yield : 일반 함수를 제너레이터로 만들어주는 예약어
```

In [26]:

```
def fib():
    prev, curr = 0, 1
    while True:
        yield curr
        prev, curr = curr, prev + curr
```

In [27]:

```
f = fib()
```

In [42]:

```
next(f)
```

Out[42]:

```
89
```

In [43]:

```
def test():
    yield 1
    yield 2
    yield 3
```

In [44]:

```
t = test()
```

In [47]:

```
next(t)
```

Out[47]:

```
3
```