Iterators

A container can provide an iterator that provides access to its elements in order

```
iter(iterable): Return an iterator over the elements of an iterable value
definite
next(iterator): Return the next element in an iterator
```

```python
>>> x = [3, 4, 5]
>>> y = iter(x)
>>> next(y)
3
>>> next(y)
4
>>> u = iter(x)
>>> next(u)
5
>>> next(u)
6
```

Demonstration: A container can provide an iterator that provides access to its elements in order

Views of a Dictionary

An iterable value is any value that can be passed to `iter` to produce an iterator

An iterator is returned from `iter` and can be passed to `next`; all iterators are mutable

A dictionary, its keys, its values, and its items are all iterable values

• The order of items in a dictionary is the order in which they were added (Python 3.6+)

• Historically, items appeared in an arbitrary order (Python 3.5 and earlier)

```python
>>> # = {'one': 1, 'two': 2, 'three': 3}
>>> # = iter(d.keys()) # or iter(d)    # or iter(d.items())
>>> next(k)            # or next(x)   # or next(l)
>>> next(k)            # or next(x)   # or next(l)
>>> 'one'              # or 'two'     # or 'three'
>>> next(k)            # or next(x)   # or next(l)
>>> 'two'              # or 'two'     # or 'three'
```

Demonstration: A dictionary, its keys, its values, and its items are all iterable values

For Statements

Built-In Iterator Functions
**Built-in Functions for Iteration**

Many built-in Python sequence operations return iterators that compute results lazily:

- `map(func, iterable)`: Iterate over `func(x)` for `x` in `iterable`
- `filter(func, iterable)`: Iterate over `x` in `iterable` if `func(x)`
- `zip(first_iter, second_iter)`: Iterate over co-indexed `(x, y)` pairs
- `reversed(sequence)`: Iterate over `x` in a sequence in reverse order

To view the contents of an iterator, place the resulting elements into a container:

- `list(iterable)`: Create a list containing all `x` in `iterable`
- `tuple(iterable)`: Create a tuple containing all `x` in `iterable`
- `sorted(iterable)`: Create a sorted list containing `x` in `iterable`

(Demo)

**Generators and Generator Functions**

A generator function is a function that yields values instead of returning them. A normal function returns once; a generator function can yield multiple times. A generator is an iterator created automatically by calling a generator function. When a generator function is called, it returns a generator that iterates over its yields.

(Demo)

**Generators can Yield from Iterators**

A `yield` statement yields all values from an iterator or iterable (Python 3.3):

```python
>>> list(a_then_b([3, 4], [5, 6]))
[3, 4, 5, 6]
```

```python
def countdown(k):
    if k > 0:
        yield k
        yield from countdown(k-1)
```

(Demo)