SICP

God’s Programming Book

Lecture-14 Iterators

Iterators

Slides Adapted from cs61a of UC Berkeley

Iterators

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
- **next**(iterator): Return the next element in an iterator

(Demo)

Dictionary Iteration

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)

Views of a Dictionary

```
>>> d = {'one': 1, 'two': 2, 'three': 3}
>>> d['zero'] = 0
>>> k = iter(d.keys())  # or iter(d)
>>> next(k)
'one'
>>> next(k)
'two'
>>> next(k)
'three'
>>> next(k)
'zero'
```

```
>>> v = iter(d.values())
>>> next(v)
1
>>> next(v)
2
>>> next(v)
3
>>> next(v)
0
```

```
>>> i = iter(d.items())
>>> next(i)
('one', 1)
>>> next(i)
('two', 2)
>>> next(i)
('three', 3)
>>> next(i)
('zero', 0)
```

(Demo)

For Statements

(Demo)

Built-In Iterator Functions

(Demo)

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
```

Built-in Functions for Iteration

- 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
```

Generators

Generators and Generator Functions

```
>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>
```

- A *generator function* is a function that **yield**s values instead of **return**ing them
- A normal function **return**s 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

Generators & Iterators

Generators can Yield from Iterators

- A **yield from** statement yields all values from an iterator or iterable (Python 3.3)

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

```
def a_then_b(a, b):
    for x in a:
        yield x
    for x in b:
        yield x
```

```
def a_then_b(a, b):
    yield from a
    yield from b
```

```
>>> list(countdown(5))
[5, 4, 3, 2, 1]
```

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

Thanks for Listening