Lecture 15_New_iterables_in_python

10 September 2024 13:13

Lecture_15_New_iterables_in_python.ipynb - Colab

iter with Range


```
R = range(10) # range returns an iterator, not a list
```

```
R
```

```
range(0, 10)
```

for i in range( )
iterator

```
range(0, 10)
```

```
range(0, 10)
```

```
I = iter(R) # Make an iterator from the range
```

```
next(I) # Advance to next result
```

```
0
```

```
next(I)
```

```
1
```

+ Code + Text

```
next(I)
```

```
2
```

forcefully R = range(10) list(R)

```
list(range(10)) # To force a list if required
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
len(R)
```

length

```
10
```

```
R[0]
```

indices

```
0
```

```
R[-1]
```

```
9
```

```
next(I)
```

```
3
```

```
I.__next__() #
```

```
4
```

Map.

Unlike range, though, they are their own iterators—after you step through their results once, they are exhausted. In other words, you can't have multiple iterators on their results that maintain different positions in those results.

iter

```
M = map(abs, (-1, 0, 1)) # map returns an iterator, not a list
```

M = map (abs, (-1, 0, 1))

```
M
```

```
<map at 0x7f8c831cf490>
```

```
next(M)
```

```
1
```

```
next(M)
```

exhausted

```
1
```

exhausted

```
next(M)
```

```
0
```

```
next(M) # Use iterator manually: exhausts results
        # These do not support len() or indexing
```

```
1
```


M = (_ _ _) exhausted


```
for x in M: print(x) # map iterator is now empty: one pass only
```

```
M = map(abs, (-1, 0, 1)) # Make a new iterator to scan again
```

```
for x in M: print(x) # Iteration contexts auto call next()
```

```
1
0
1
```

len(M)

```
list(map(abs, (-1, 0, 1))) # Can force a real list if needed
```

```
[1, 0, 1]
```

Multiple Versus Single Iterators

```
R = range(3) # range allows multiple iterators
```

I1 = iter(R)

I2 = iter(R)

```
next(R)
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-28-1ef46f494a83> in <cell line: 1>()
----> 1 next(R)

TypeError: 'range' object is not an iterator
```

Next steps: Explain error

```
I1 = iter(R)
```

R → (0, 3)

```
next(I1)
```

```
0
```

```
next(I1)
```

```
1
```

```
I2 = iter(R) # Two iterators on one range
```

```
next(I2)
```

```
0
```

```
next(I1) # I1 is at a different spot than I2
```

```
2
```

Dictionary View Iterators

```
D = dict(a=1, b=2, c=3)
```

```
D
```

```
{'a': 1, 'b': 2, 'c': 3}
```

```
K = D.keys() # A view object in 3.0, not a list
```

```
{'a': 1, 'b': 2, 'c': 3}
```

```
K = D.keys() # A view object in 3.0, not a list
```

```
K
```

```
dict_keys(['a', 'b', 'c'])
```

```
next(K) # Views are not iterators themselves
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-0766142846ec> in <cell line: 1>()
----> 1 next(K) # Views are not iterators themselves

TypeError: 'dict_keys' object is not an iterator
```

Next steps: Explain error

```
I = iter(K) # Views have an iterator,
```

```
next(I) # which can be used manually
```

```
5
```

```
next(I)
```

```
6
```

```
for k in D.keys(): print(k, end=' ') # All iteration contexts use auto
```

```
a b c
```

```
K = D.keys()
```

```
list(K) # Can still force a real list if needed
```

```
['a', 'b', 'c']
```

```
V = D.values() # Ditto for values() and items() views
```

```
V
```

```
dict_values([1, 2, 3])
```

```
list(V)
```

```
[1, 2, 3]
```

```
list(D.items())
```

```
[('a', 1), ('b', 2), ('c', 3)]
```

```
for (k, v) in D.items(): print(k, v, end=' ')
```

```
a 1 b 2 c 3
```

Start coding or generate with AI.