Contents

Chapter 1

Functions

1.1 bigrange – range-like generator functions

1.1.1 count – count up

count(`n`: *integer*=**0**) → *iterator*

Count up infinitely from `n` (default to 0). See itertools.count.

`n` must be int, long or rational.Integer.

1.1.2 range – range-like iterator

range(`start`: *integer*, `stop`: *integer*=**None**, `step`: *integer*=**1**)
→ *iterator*

Return a range-like iterator which generates a finite integer sequence.

It can generate more than sys.maxint elements, which is the limitation of the range built-in function.

The argument names do not correspond to their roles, but users are familiar with the range built-in function of `Python` and understand the semantics. Note that the output is not a list.

Examples

```
>>> range(1, 100, 3) # built-in
[1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46,
 49, 52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91,
```

```
 94, 97]
>>> bigrange.range(1, 100, 3)
<generator object at 0x18f8c8>
```

1.1.3 arithmetic_progression – arithmetic progression iterator

arithmetic_progression(`init`: *integer*, `difference`: *integer*)
→ *iterator*

Return an iterator which generates an arithmetic progression starting form `init` and `difference` step.

1.1.4 geometric_progression – geometric progression iterator

geometric_progression(`init`: *integer*, `ratio`: *integer*)
→ *iterator*

Return an iterator which generates a geometric progression starting form `init` and multiplying `ratio`.

1.1.5 multirange – multiple range iterator

multirange(`triples`: *list of range triples*) → *iterator*

Return an iterator over Cartesian product of elements of ranges.

Be cautious that using multirange usually means you are trying to do brute force looping.

The range triples may be doubles `(start, stop)` or single `(stop,)`, but they have to be always tuples.

Examples

```
>>> bigrange.multirange([(1, 10, 3), (1, 10, 4)])
<generator object at 0x18f968>
>>> list(_)
[(1, 1), (1, 5), (1, 9), (4, 1), (4, 5), (4, 9), (7, 1),
 (7, 5), (7, 9)]
```

1.1.6 multirange_restrictions – multiple range iterator with restrictions

multirange_restrictions(`triples`: *list of range triples*, `**kwds`: *keyword arguments*)
→ *iterator*

`multirange_restrictions` is an iterator similar to the `multirange` but putting restrictions on each ranges.

Restrictions are specified by keyword arguments: `ascending`, `descending`, `strictly_ascending` and `strictly_descending`.

A restriction `ascending`, for example, is a sequence that specifies the indices where the number emitted by the range should be greater than or equal to the number at the previous index. Other restrictions `descending`, `strictly_ascending` and `strictly_descending` are similar. Compare the examples below and of **multirange**.

Examples

```
>>> bigrange.multirange_restrictions([(1, 10, 3), (1, 10, 4)], ascending=(1,))
<generator object at 0x18f978>
>>> list(_)
[(1, 1), (1, 5), (1, 9), (4, 5), (4, 9), (7, 9)]
```