Real Python Cheat Sheet (realpython.com)

Python Itertools Examples

Learn more about using Python's `itertools` module in our in-depth tutorial at realpython.com/python-itertools/.

`itertools.combinations(iterable, n)`

Return successive r-length combinations of elements in the iterable.

```
>>> combinations([1, 2, 3], 2)
(1, 2), (1, 3), (2, 3)
```

`itertools.combinations_with_replacement(iterable, n)`

Return successive r-length combinations of elements in the iterable allowing individual elements to have successive repeats.

```
>>> combinations_with_replacement([1, 2], 2)
(1, 1), (1, 2), (2, 2)
```

`itertools.product(*iterables, repeat=1)`

Cartesian product of input iterables. Equivalent to nested for-loops.

```
>>> product([1, 2], ['a', 'b'])
(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')
```

`itertools.permutations(iterable)`

Return successive r-length permutations of elements in the iterable.

```
>>> permutations('abc')
('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'),
('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')
```

`itertools.count(start=0, step=1)`

Return a count object whose `.__next__()` method returns consecutive values.

```
>>> count()
0, 1, 2, 3, 4, ...

>>> count(start=1, step=2)
1, 3, 5, 7, 9, ...
```

itertools.repeat(object[, times])

Create an iterator which returns the object for the specified number of times. If not specified, returns the object endlessly.

```
>>> repeat(2)
2, 2, 2, 2, 2 ...

>>> repeat(2, 5)  # Stops after 5 repititions.
2, 2, 2, 2, 2
```

itertools.cycle(iterable)

Return elements from the iterable until it is exhausted. Then repeat the sequence indefinitely.

```
>>> cycle(['a', 'b', 'c'])
a, b, c, a, b, c, a, ...
```

itertools.accumulate(iterable[, func])

Return series of accumulated sums (or other binary function results).

```
>>> accumulate([1, 2, 3])
1, 3, 6
```

itertools.tee(iterable, n=2)

Create any number of independent iterators from a single input iterable.

```
>>> iter1, iter2 = tee(['a', 'b', 'c'], 2)
>>> list(iter1)
['a', 'b', 'c']
>>> list(iter2)
['a', 'b', 'c']
```

itertools.islice(iterable, stop)

itertools.islice(iterable, start, stop, step=1)

Return an iterator whose `__next__()` method returns selected values from an iterable. Works like a `slice()` on a list but returns an iterator.

```
>>> islice([1, 2, 3, 4], 3)
1, 2, 3

>>> islice([1, 2, 3, 4], 1, 2)
2, 3
```

itertools.chain(*iterables)

Return a chain object whose `__next__()` method returns elements from the first iterable until it is exhausted, then elements from the next iterable, until all of the iterables are exhausted.

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

itertools.chain.from_iterable(iterable)

Alternate `chain()` constructor taking a single iterable argument that evaluates lazily.

```
>>> chain.from_iterable(['abc', [1, 2, 3]])
'a', 'b', 'c', 1, 2, 3
```

itertools.filterfalse(pred, iterable)

Return those items of sequence for which `pred(item)` is false. If `pred` is `None`, return the items that are false.

```
>>> filterfalse(bool, [1, 0, 1, 0, 0])
0, 0, 0
```

itertools.takewhile(pred, iterable)

Return successive entries from an iterable as long as `pred` evaluates to true for each entry.

```
>>> takewhile(bool, [1, 1, 1, 0, 0])
1, 1, 1
```

itertools.dropwhile(pred, iterable)

Drop items from the iterable while `pred(item)` is true. Afterwards, return every element until the iterable is exhausted.

```
>>> dropwhile(bool, [1, 1, 1, 0, 0])
0, 0
```

itertools.zip_longest(*iterables, fillvalue=None)

Return a zip_longest object whose `.__next__()` method returns a tuple where the i-th element comes from the i-th iterable argument. The `.__next__()` method continues until the longest iterable in the argument sequence is exhausted and then it raises `StopIteration`. When the shorter iterables are exhausted, the fillvalue is substituted in their place. The `fillvalue` defaults to `None` or can be specified by a keyword argument.

```
>>> zip_longest([1, 2, 3], ['a', 'b'])
(1, 'a'), (2, 'b'), (3, None)

>>> zip_longest([1, 2], ['a', 'b', 'c'], fillvalue=0)
(1, 'a'), (2, 'b'), (0, 'c')
```

itertools.groupby(iterable, key=None)

Make an iterator that returns consecutive keys and groups from the iterable. If the key function is not specified or is `None`, the element itself is used for grouping.

```
>>> groupby([1, 1, 1, 2, 2, 3, 3, 3, 3])
(1, [1, 1, 1]), (2, [2, 2]), (3, [3, 3, 3, 3])

>>> groupby([('a', 1), ('a', 2), ('b', 0)], key=lambda x: x[0])
('a', [1, 2]), ('b', [0])
```

itertools.starmap(func, iterable)

Return an iterator whose values are returned from the function evaluated with an argument tuple taken from the given sequence.

```
>>> starmap(math.pow, [(2, 0), (2, 1), (2, 2), (2, 3)])
1, 2, 4, 8
```

itertools.compress(data, selectors)

Return data elements corresponding to true selector elements. Forms a shorter iterator from selected data elements using the selectors to choose the data elements.

```
>>> compress('abcdef', [1, 0, 1, 0, 1, 1])
'a', 'c', 'e', 'f'
```