# Overview of itertools

Print to PDF

Let us go through one of the important library to manipulate collections called as `itertools`.

- Functions such as `filter` and `map` are part of core Python libraries.
- `reduce` is part of `functools`.
- It is not possible to use these functions to perform advanced operations such as grouped aggregations, joins etc.
- Python have several higher level libraries which provide required functionality to perform advanced aggregations such as grouped aggregations, joins etc. `itertools` is one of the popular library to manipulate collections.

```
import itertools
```

```
itertools?
```

```
Type:        module
String form: <module 'itertools' (built-in)>
Docstring:
Functional tools for creating and using iterators.

Infinite iterators:
count(start=0, step=1) --> start, start+step, start+2*step, ...
cycle(p) --> p0, p1, ... plast, p0, p1, ...
repeat(elem [,n]) --> elem, elem, elem, ... endlessly or up to n times

Iterators terminating on the shortest input sequence:
accumulate(p[, func]) --> p0, p0+p1, p0+p1+p2
chain(p, q, ...) --> p0, p1, ... plast, q0, q1, ...
chain.from_iterable([p, q, ...]) --> p0, p1, ... plast, q0, q1, ...
compress(data, selectors) --> (d[0] if s[0]), (d[1] if s[1]), ...
dropwhile(pred, seq) --> seq[n], seq[n+1], starting when pred fails
groupby(iterable[, keyfunc]) --> sub-iterators grouped by value of keyfunc(v)
filterfalse(pred, seq) --> elements of seq where pred(elem) is False
islice(seq, [start,] stop [, step]) --> elements from
       seq[start:stop:step]
starmap(fun, seq) --> fun(*seq[0]), fun(*seq[1]), ...
tee(it, n=2) --> (it1, it2 , ... itn) splits one iterator into n
takewhile(pred, seq) --> seq[0], seq[1], until pred fails
zip_longest(p, q, ...) --> (p[0], q[0]), (p[1], q[1]), ...

Combinatoric generators:
product(p, q, ... [repeat=1]) --> cartesian product
permutations(p[, r])
combinations(p, r)
combinations_with_replacement(p, r)
```

## Task 1

Get cumulative sales from list of transactions.

```
ns = [1, 2, 3, 4]
```

```
import itertools as iter
```

```
iter.accumulate?
```

```
Init signature: iter.accumulate(self, /, *args, **kwargs)
Docstring:
accumulate(iterable[, func]) --> accumulate object

Return series of accumulated sums (or other binary function results).
Type:           type
Subclasses:
```

```
list(iter.accumulate(ns))[:10]
```

```
[1, 3, 6, 10]
```

```
import operator as o
list(iter.accumulate(ns, o.mul))[:10]
```

```
[1, 2, 6, 24]
```

```
iter.chain?
```

```
Init signature: iter.chain(self, /, *args, **kwargs)
Docstring:
chain(*iterables) --> chain object

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.
Type:           type
Subclasses:
```

```
l1 = [1, 2, 3, 4]
l2 = [4, 5, 6]

list(iter.chain(l1, l2))
```

```
[1, 2, 3, 4, 4, 5, 6]
```

```
help(iter.starmap)
```

```
Help on class starmap in module itertools:

class starmap(builtins.object)
 |  starmap(function, sequence) --> starmap object
 |
 |  Return an iterator whose values are returned from the function evaluated
 |  with an argument tuple taken from the given sequence.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
```

## Task - Compute Commission Amount

Create a collection with sales and commission percentage. Using that collection compute total commission amount. If the commission percent is None or not present, treat it as 0.

- Each element in the collection should be a tuple.
- First element is the sales amount and second element is commission percentage.
- Commission for each sale can be computed by multiplying commission percentage with sales (make sure to divide commission percentage by 100).
- Some of the records does not have commission percentage, in that case commission amount for that sale shall be 0

```
transactions = [(376.0, 8),
(548.23, 14),
(107.93, 8),
(838.22, 14),
(846.85, 21),
(234.84, None),
(850.2, 21),
(992.2, 21),
(267.01, None),
(958.91, 21),
(412.59, None),
(283.14, None),
(350.01, 14),
(226.95, None),
(132.7, 14)]
```

```
iter.starmap(lambda rec: rec[0] * rec[1] if rec[1] else 0, transactions)
```

```
<itertools.starmap at 0x7f29b8a12f28>
```

```
a = iter.starmap(lambda sale_amount, comm_pct: round(sale_amount * comm_pct, 2) if comm_pct
else 0, transactions)
```

```
for i in a: print(i)
```

```
3008.0
7675.22
863.44
11735.08
17783.85
0
17854.2
20836.2
0
20137.11
0
0
4900.14
0
1857.8
```