Exercises

1) Create an iterator that returns the cumulative sum of a given list.
2) Create an iterator that returns only the unique elements of a given list.
3) Create an iterator that returns the running maximum of a given list.
4) Create an iterator that returns the running minimum of a given list.
5) Create an iterator that returns the running average of a given list.
6) Create an iterator that returns the longest continuous run of a given element in a list.
7) Create an iterator that returns the first n permutations of a given list.
8) Create an iterator that returns the first n combinations of a given list.
9) Create an iterator that returns the first n combinations with replacement of a given list.
10) Create an iterator that returns all the subsets of a given set.

Exercises and solution

1) Create an iterator that returns the cumulative sum of a given list.

```
class CumulativeSumIterator:
    def __init__(self, lst):
        self.cumulative_sum = 0
        self.iterator = iter(lst)

    def __iter__(self):
        return self

    def __next__(self):
        value = next(self.iterator)
        self.cumulative_sum += value
        return self.cumulative_sum
```

Example usage:

```
>>> csi = CumulativeSumIterator([1, 2, 3, 4])
>>> for value in csi:
...     print(value)
...
1
3
6
10
```

2) Create an iterator that returns only the unique elements of a given list.

```
class UniqueElementsIterator:
    def __init__(self, lst):
        self.iterator = iter(lst)
```

```
        self.unique_set = set()

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            value = next(self.iterator)
            if value not in self.unique_set:
                self.unique_set.add(value)
                return value
```

Example usage:

```
>>> uei = UniqueElementsIterator([1, 2, 2, 3, 3, 3, 4, 4, 4, 4])
>>> for value in uei:
...     print(value)
...
1
2
3
4
```

3) Create an iterator that returns the running maximum of a given list.

```
class RunningMaximumIterator:
    def __init__(self, lst):
        self.iterator = iter(lst)
        self.running_max = float('-inf')

    def __iter__(self):
        return self

    def __next__(self):
        value = next(self.iterator)
        self.running_max = max(value, self.running_max)
```

```
        return self.running_max
```

Example usage:

```
>>> rmi = RunningMaximumIterator([1, 2, 3, 2, 1, 4, 5, 3])
>>> for value in rmi:
...     print(value)
...
1
2
3
3
3
4
5
5
```

4) Create an iterator that returns the running minimum of a given list.

```
class RunningMinimumIterator:
    def __init__(self, lst):
        self.iterator = iter(lst)
        self.running_min = float('inf')

    def __iter__(self):
        return self

    def __next__(self):
        value = next(self.iterator)
        self.running_min = min(value, self.running_min)
        return self.running_min
```

Example usage:

```
>>> rmi = RunningMinimumIterator([1, 2, 3, 2, 1, 4, 5, 3])
>>> for value in rmi:
...     print(value)
```

```
...
1
1
1
1
1
1
1
1
```

5) Create an iterator that returns the running average of a given list.

```
class RunningAverageIterator:
    def __init__(self, lst):
        self.iterator = iter(lst)
        self.running_sum = 0
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        value = next(self.iterator)
        self.running_sum += value
        self.count += 1
        return self.running_sum / self.count
```

Example usage:

```
>>> rai = RunningAverageIterator([1, 2, 3, 4])
>>> for value in rai:
...     print(value)
...
1.0
1.5
```

```
2.0
2.5
```

6) Create an iterator that returns the longest continuous run of a given element in a list.

```
class LongestRunIterator:
    def __init__(self, lst, element):
        self.iterator = iter(lst)
        self.current_run = 0
        self.max_run = 0
        self.element = element

    def __iter__(self):
        return self

    def __next__(self):
        value = next(self.iterator)
        if value == self.element:
            self.current_run += 1
            self.max_run = max(self.max_run, self.current_run)
        else:
            self.current_run = 0
        return self.max_run
```

Example usage:

```
>>> lri = LongestRunIterator([1, 2, 3, 3, 3, 4, 5, 3], 3)
>>> for value in lri:
...     print(value)
...
1
1
3
3
3
3
```

```
3
3
```

7) Create an iterator that returns the first n permutations of a given list.

```
import itertools

class NPermutationsIterator:
    def __init__(self, lst, n):
        self.iterator = itertools.permutations(lst)
        self.n = n
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count == self.n:
            raise StopIteration
        self.count += 1
        return next(self.iterator)
```

Example usage:

```
>>> npi = NPermutationsIterator([1, 2, 3], 3)
>>> for value in npi:
...     print(value)
...
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
```

8) Create an iterator that returns the first n combinations of a given list.

```
import itertools

class NCombinationsIterator:
    def __init__(self, lst, n):
        self.iterator = itertools.combinations(lst, n)
        self.n = n
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count == self.n:
            raise StopIteration
        self.count += 1
        return next(self.iterator)
```

Example usage:

```
>>> nci = NCombinationsIterator([1, 2, 3], 2)
>>> for value in nci:
...     print(value)
...
(1, 2)
(1, 3)
(2, 3)
```

9) Create an iterator that returns the first n combinations with replacement of a given list.

```
import itertools

class NCombinationsWithReplacementIterator:
    def __init__(self, lst, n):
        self.iterator =
itertools.combinations_with_replacement(lst, n)
        self.n = n
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count == self.n:
            raise StopIteration
        self.count += 1
        return next(self.iterator)
```

Example usage:

```
>>> ncwri = NCombinationsWithReplacementIterator([1, 2,
3], 2)
>>> for value in ncwri:
...     print(value)
...
(1, 1)
(1, 2)
(1, 3)
(2, 2)
(2, 3)
(3, 3)
```

10) Create an iterator that returns all the subsets of a given set.

```
import itertools

class SubsetsIterator:
    def __init__(self, lst):
        self.subsets = []
        for i in range(len(lst) + 1):
            self.subsets += itertools.combinations(lst, i)
        self.iterator = iter(self.subsets)

    def __iter__(self):
        return self

    def __next__(self):
        return next(self.iterator)
```

Example usage:

```
>>> si = SubsetsIterator([1, 2, 3])
>>> for value in si:
...     print(value)
...
()
(1,)
(2,)
(3,)
(1, 2)
(1, 3)
(2, 3)
(1, 2, 3)
```