Source Code 1. Ein Iterator als Klasse, am Beispiel der k–elementigen Teilmengen.

```
# Eine selbst definierte Klasse (englisch: class) in Python
class subsets_colex_iterator:
    """Iterator: k--elem. Teilmengen von [n] in co-lexikographischer
    Reihenfolge"""

    # Eine Klasse "braucht" (in der Regel) eine "Initialisierung":
    def __init__(self, n, k):
        if n < 0 or k > n:
            print('Sehr lustig:-(')
            n = k = 0
        self.n = n
        self.k = k

    # Die Funktionen __iter__ und __next__ implementieren die
    # "Iterator-Eigenschaft" der Klasse:
    def __iter__(self):
        # Nicht vergessen: Listen-Indizierung beginnt bei Null, range(k)
        # liefert {0,1,...,k-1}; die erste Teilmenge in der
        # co-lexikographischen Ordnung ist {1,...,k} - wir speichern
        # aber _ein Element mehr ab_, aus Gründen, die weiter unten klar
        # werden ...
        self.current_subset = [i+1 for i in range(self.k+1)]
        # ... genauer gesagt: Wir schreiben ans Ende n+1 als
        # "Ende-Markierung" ...
        self.current_subset[self.k] = self.n+1
        # ... und erstes Element 0 als Flag, daß wir grade erst begonnen
        # haben:
        self.current_subset[0]=0

        # Plan: self_current_subset ist IMMER eine Liste von aufsteigend
        # geordneten k+1 positiven ganzen Zahlen, deren letztes Element
        # IMMER gleich n+1 ist.
        return self

    def __next__(self):
        # Wenn die Teilmenge self.current_subset gleich {n-k+1,...,n} ist,
        # dann geht's nicht mehr weiter:
        if self.k == 0 or self.current_subset[0] >= self.n-self.k + 1:
            raise StopIteration
        else:
            # Suche das erste Element, das um mindestens 2 kleiner ist als
            # sein Nachfolger (wenn wir bis hierher gekommen sind, _muß_ es
            # so ein Element geben - spätestens bei j = n, dann gilt die
            # Abbruchbedingung für die "Endmarkierung".)
            j = 0
            while self.current_subset[j]+1 >= self.current_subset[j+1]:
                j+= 1
```

```
            # Dieses Element erhöhen wir um 1 ...
            self.current_subset[j]+= 1
            # ... und setzen alle vorhergehenden Elemente auf den
            # "co-lex-Minimalwert":
            self.current_subset[:j] =  [i+1 for i in range(j)]

            return self.current_subset[:-1]
```