

SOURCE CODE 1. Unranking: Zahl-Partitionen von n in umgedrehter lexikographischer Ordnung.

```
def unrank_integer_partitions(rank, n):
    """Bestimme die Zahlpartition von n mit Rang rank
    in umgedrehter lexikographischer Ordnung"""
    # Berechne rekursiv die Funktionswerte  $p(n,k)$  in einer Tabelle
    pnk = p_nk_by_recursion(n)
    p_n = pnk[n][n]
    if rank < 0 or rank >= p_n:
        print(f'Rank {rank} not in [0,{p_n-1}]!')
        return None
    # Finde die eindeutige Zahlendarstellung von  $p(n)-1-\text{rank}$  als Summe
    #  $p(n_j, pi_j-1)$ , wobei  $n_j = \text{rank} - pi_1 - p_2 - \dots - p_{j-1}$ .
    pi = []
    num = p_n - rank - 1
    n_j = n
    while n_j:
        # Finde das größte  $pi_j$  sodaß  $p(n_j, pi_j - 1) \leq \text{num}$ :
        # "Technisch" machen wir das durch die numpy-Funktion
        # "count_nonzero", die wir auf Zeile "pnk[n_j] <= num"
        # anwenden; das ist eine Zeile, die 1 enthält, wo die
        # Ungleichung erfüllt ist, und 0 sonst.
        pi_j = np.count_nonzero(pnk[n_j] <= num)
        # Wenn  $pi_j = 1$ , dann ist die restliche Partition schon
        # eindeutig bestimmt (als  $n_j$  Einser):
        if pi_j == 1:
            return np.array(pi + [1]*n_j)
        # Ansonsten: Update num, pi und n_j ...
        num -= pnk[n_j][pi_j - 1]
        pi += [pi_j]
        n_j -= pi_j
        # ... und weiter in der Schleife!

    return np.array(pi, dtype = int)
```

Unter Verwendung der Funktion $p(n,k)$ müssen wir die (eindeutige) Darstellung von rank als $\sum p(n_j, pi_j - 1)$ finden: (pi_1, pi_2, \dots) ist dann die gesuchte Zahlpartition.