Source Code 1. Generator für Zahl–Partitionen.

```
def generate_integer_partitions(n):
    """Generator, der einen Iterator liefert, der alle Partitionen
    von n erzeugt"""
    # Eine Partition von n kann höchstens n Teile haben: Wir bereiten ein
    # numpy-array dieser Länge vor, mit Datentyp int
    mu = np.zeros(n, dtype=int)
    # Die "kleinste" Partition (in umgekehrt lexikographischer Ordnung)
    # ist einfach gleich [n] (bzw. [n,0,0, ...,0]).
    mu[0] = n
    nof_parts = 1
    while True:
        # Der Befehl "yield" (statt "return") macht aus der Funktion einen
        # Generator:  Rückgabe ohne "trailing zeros".
        yield mu[:nof_parts]
        # Wenn eine Partition mit 1 beginnt, gibt es keine "größere" (in
        # umgekehrt lexikographischer Ordnung) mehr:
        if mu[0] == 1:
            break
        # Suche letzten Teil der Partition, der größer als 1 ist:
        # Dazu verwenden wir die "Vektor-Funktionen" von numpy;
        # der Ausdruck "mu>0" liefert ein boolean numpy-array
        # in dem wir die "nicht False"-Einträge abzählen, die
        # sind aber intern "nicht Null"-Einträge:
        nof_greater_one = np.count_nonzero(mu>1)
        # Dieser letzte Teil, der größer ist als 1, wird um 1 vermindert ...
        new_part = mu[nof_greater_one-1]-1
        # ... und der ursprüngliche Teil wird mit den restlichen Einsern
        # (sofern vorhanden) zusammengezählt ...
        sum_rest = np.sum(mu[nof_greater_one-1:])
        # ... aus dieser Summe machen wir "soviel wie möglich" Teile
        # new_part (Achtung: ganzzahlige Division "//" verwenden, sonst
        # ist das Ergebnis vom Typ float!) ...
        nof_new_parts = sum_rest//new_part
        # ... der Rest wird hinten als ein neuer Teil angestückelt ...
        additional_part = sum_rest - new_part*nof_new_parts
        # ... und zwar durch "Slicing" (Herausschneiden) eines "Stückes"
        # aus mu, dem dann ein konstanter Wert zugewiesen wird; zuerst
        # mal die Teile der Größe new_part:
        alpha = nof_greater_one-1
        omega = alpha + nof_new_parts
        mu[alpha:omega] = new_part
        nof_parts = omega
        # Dann der zusätzliche Teil:
        if omega < n:
            mu[omega] = additional_part
            # Wenn der zusätzliche Teil nicht Null ist, erhöht er die
            # Anzahl der Teile:
            if additional_part:
                nof_parts += 1
```

```
            # Zuguterletzt: Alles nach dem zusätzlichen Teil wird auf
            # Null gesetzt.
            mu[omega+1:] = 0
```

Zahl–Partitionen von $n$ werden als *absteigend geordnete* Folge der Teile (Summanden) angeschrieben (hier intern mit angehängten Nullen, die wir bei den Rückgabewerten weglassen). Die Funktion liefert einen *Iterator*, der diese Folgen in umgekehrter lexikographischer Ordnung erzeugt.