Generator für Funktionen von beschränktem Wachstum (in Bijektion mit Mengen-Partitionen).

```
def functions_of_restricted_growth(n):
    """Generator, der einen Iterator liefert, der alle Funktionen
    von beschränktem Wachstum auf [n] in lexikographischer Ordnung
    liefert."""
    the_func = np.ones(n, dtype=int)
    the_max  = np.ones(n, dtype=int)
    if n>1:
        the_max[1:]+= 1
    # the_max is a non-decreasing pointwise upper bound for the_func;
    # and the sequence of the_max-es is "pointwise non-decreasing":
    while True:
        yield the_func
        # For which indices is the_max > the_func? The numpy-function
        # flatnonzero(a) returns the indices that are non-zero in the
        # flattened version of a. (Flattened heißt hier: das array a
        # wird "als Vektor interpretiert"; eine Matrix [[1,2],[3,4]]
        # also z.B. als [1,2,3,4] - das ist genau das, was wir brauchen.)
        indices_of_possible_increase = np.flatnonzero(the_max-the_func)
        if len(indices_of_possible_increase):
            # Choose the last of these possible indices ...
            index_to_increase = indices_of_possible_increase[-1]
            # ... and increase the function at this index ...
            the_func[index_to_increase]+=1
            # ... and set the function at all later indices to 1:
            the_func[index_to_increase+1:] = 1
            # Update the_max, if necessary:
            if the_max[index_to_increase] == the_func[index_to_increase]:
                the_max[index_to_increase+1:] = the_max[index_to_increase]+1
        else:
            break
```

Die Funktion f wird als numpy-Vektor (f(1), f(2), ... f(n)) codiert, ebenso die "Wachstumsbeschränkung". Durch Verwendung von numpy-Funktionalität wird der Code wieder recht übersichtlich.