Source Code 1. product-with-itertools

```
def product_with_itertools(the_list, the_func):
    """Verwende die Python-Bibliotheksfunktion itertools.product()
    zur Erzeugung des cartesischen Produkts: """
    # Vorbereiten: itertools.product übernimmt _beliebig viele Listen_ als
    # Argumente; wir erzeugen also zunächst eine Liste von Listen (der
    # Befehl range(1,i+1) entspricht der Liste [1,2,...,i]) ...
    arguments = [list(range(1,i+1)) for i in the_list]
    # ... und "entfernen die äußersten Listenklammern [ ... ]" durch
    # "*" bei der Übergabe von arguments: Das muß man sich so vorstellen
    # product(*[l_1, l_2, ..., l_m]) -> product(l_1, l_2, ..., l_m)
    # (denn itertools.product "erwartet" die Argumente in dieser Form):
    for pi in itertools.product(*arguments):
        # Wende die Funktion the_func auf das aktuelle Tupel an:
        # Ein weiteres, sehr nützliches Merkmal von Python ist die
        # problemlose Übergabe einer Funktion als Argument an eine andere
        # Funktion!
        the_func(pi)
```

Wie so viele andere nützliche Algorithmen und Routinen ist auch die Erzeugung des kartesischen Produkts in Pythn bereits in einem Modul (itertools) ausprogrammiert, das man nur importieren muß.

`itertools.product(the_list,len(the_list))` liefert einen Iterator: Das heißt, daß *nicht* die gesamte Liste aller Tupel auf einmal erzeugt wird (was ja einen großen Speicherplatzverbrauch bedeuten würde), sondern eine *Funktion*, die (z.B. in einer for–Schleife) die Tupel *eines nach dem anderen* liefert. Wir sehen hier auch ein weiteres nützliches Merkmal von Python: Man kann einer Funktion ohne weiteres eine *Funktion* (hier: `the_func`) als Argument übergeben!