Multiplikationstabelle einer Permutationsgruppe.

```
# itertools.permutations() liefert einen Iterator, der seinerseits
# Python-Tupel (keine Python-Listen!) erzeugt: Eine Liste aller dieser
# Tupel könnten wir so erzeugen:
# alle_permutationen = []
# for pi in itertools.permutations(range(1,5)):
#     alle_permutationen+= [pi]
# denn der Operator "+" wirkt auf Listen wie das "Aneinanderhängen",
# z.B:
print([1,2,3,4]+[-4,-3,-2,-1])
# Eleganter ist aber "list comprehension", eine syntaktische
# Besonderheit von Python:
grundmenge = list(range(1,4))
alle_permutationen = [pi for pi in itertools.permutations(grundmenge)]
print(f'Die Permutationen von {grundmenge} sind: {alle_permutationen}.')
# Die Permutationen können wir (bei Null (!) beginnend) durchnumerieren:
nummern = list(range(len(alle_permutationen)))
# Eine "Übersetzung" von n in die entsprechende Permutation mit Nummer n
# ist leicht:  alle_permutationen[n] liefert das sofort. Umgekehrt können
# wir ein weiteres überaus nützliches Sprachelement von Python nutzen,
# das Dictionary.
# Eine Übersetzungstabelle, die jeder Permutation ihre Nummer zuordnet,
# können wir mit dem Python-Sprachelement "zip" ganz leicht erzeugen,
# das die Elemente von zwei Listen oder Tupeln "reißverschlußartig"
# zusammenfaßt. Die gewünschte Übersetzungstabelle erhalten wir so:
pi_num = dict(zip(alle_permutationen, nummern))
print(f'Permutation (3,1,2) hat Nummer {pi_num[(3,1,2)]}.')
# Wenn wir nun den Permutationen ihre Nummern zuordnen, können wir
# die Multiplikationstabelle automatisch erzeugen: Als erstes
# basteln wir eine Zeile der gesuchten Tabelle (die anfangs
# Werte enthält, die wir später überschreiben):
mult_tab_zeile = list(range(len(alle_permutationen)))
# Jetzt befüllen wir diese Zeile mit den Nummern der Permutationen,
# die den Produkten entsprechen, in einer doppelten for-Schleife.
# Wir verwenden dabei ein nützliches Sprachelement in Python:
#      enumerate(liste)
# ergibt einem Iterator, der der Reihe nach die Paare (Tupel)
#      0,liste[0]
#      1,liste[1]
#      etc.
# liefert. In unserem Beispiel entsprechen die ersten Elemente dieser
# Paare den Zeilen/Spalten der Tabelle in den Schleifen entsprechen:
print('Multiplikationstabelle:')
for sigma in alle_permutationen:
    for spalten_index,tau in enumerate(alle_permutationen):
        # Rückgabewert der Permutationsmultiplikation ist eine
        # Liste: Umwandeln in ein Tupel ...
        rho = tuple(perm_mul(sigma, tau))
        # ... Übersetzen in die entsprechende Nummer ...
        num = pi_num[rho]
        # ... und Eintragen in die Tabellenzeile:
        mult_tab_zeile[spalten_index] = num
    # Ausgabe der Zeile - ergibt insgesamt einen Ausdruck der Tabelle:
    print(mult_tab_zeile)
```