Permutation

1. Gegeben sei ein Tupel von n Zahlen $(a_0, a_2, \ldots, a_{n-1})$ mit der Eigenschaft, dass genau die Zahlen $0, 1, 2, \ldots, n-1$ in beliebiger Reihenfolge enthalten sind. Dieses Tupel könnte eine Permutation darstellen. Schreiben Sie eine Funktion, die das obige Tupel als Eingabe erhält und ein Tupel ausgibt, in der anstatt der Zahl a_i an der Stelle i die Zahl i an der Stelle a_i liegt (also die Permutation invertiert).

2. Überlegen Sie sich einen Test für Ihren Algorithmus.
   Hinweis: Tests können mitunter Beispiele sein, bei denen wir das korrekte Ergebnis kennen, oder logische Tests.

Solution:

```
#!/usr/bin/env python
# coding: utf-8
# <h1>Table of Contents<span class="tocSkip"></span></h1>
# <div class="toc"><ul class="toc-item"><li><span><a href="#Inverse-Permutation" data-toc-modified-id="Inverse-Permutation-1"><span class="toc-item-num">1  </span>Inverse Permutation</a></span></li></ul></div>
# ### Inverse Permutation
#
# **Pseudo Code**
Input of some permutation L = (a0, ..., an-1) of length n.
Create empty list M
for all k in 0,..,n-1
    a = L[k]
    M[a] = k

print("L", L)
print("M", M)
# **Python**
L = [3,1,0,2]
def invPerm(L):
    """docstring"""
    n = len(L)  # length of list L
    M = list(range(n))  # placeholder list for result
    for k in range(n):  # iterate through the list, index by index
        a = L[k]  # a is the k-th element of L
        M[a] = k  # M_a is set to the index of a in L, i.e., k
    return M
print("L", L)
M = invPerm(L)
print("M", M)
# **Test**
#
# Da es sich um die inverse Permutation handelt sollte die Anwendung der Inversen auf die urspr ngliche Permutation immer die Identit t ergeben.
# M is the "right inverse" of L
print([L[m] for m in M])
# M is the "left inverse" of L
print([M[l] for l in L])
```