8 puzzle in python

- Look at a simple implementation of eight puzzle in python
- `p8.py`
- Solve using A* with three different heuristics
  - NIL: $h = 0$
  - OOP: $h = \# \text{ of tiles out of place}$
  - MHD: $h = \text{sum of manhattan distance between each tile’s current \& goal positions}$
- All three are admissible
What must we model?

• A state
• Goal test
• Actions
• Result of doing action in state
• Heuristic function
A State

• Represent state as string of nine characters with blank as *
  
  E.g.: “1234*5678”

• Position of blank in state S is just S.index(‘*’)
def actions8(S):  # returns list of legal actions in state S
    action_table = {
        0: ['down', 'right'],
        1: ['down', 'left', 'right'],
        2: ['down', 'left'],
        3: ['up', 'down', 'right'],
        4: ['up', 'down', 'left', 'right'],
        5: ['up', 'down', 'left'],
        6: ['up', 'right'],
        7: ['up', 'left', 'right'],
        8: ['up', 'left']
    }
    return action_table[S.index('*)')]
def result8(S, A):
    blank = S.index('*')  # blank position
    if A == 'up':
        swap = blank - 3
        return S[0:swap] + '*' + S[swap+1:blank] + S[swap] + S[blank+1:]
    elif A == 'down':
        swap = blank + 3
        return S[0:blank] + S[swap] + S[blank+1:swap] + '*' + S[swap+1:]
    elif A == 'left':
        swap = blank - 1
        return S[0:swap] + '*' + S[swap] + S[blank+1:]
    elif A == 'right':
        swap = blank + 1
        return S[0:blank] + S[swap] + '*' + S[swap+1:]
    raise ValueError('Unrecognized action: ' + A)
Heuristic function

class P8_h1(P8):
    """ Eight puzzle using a heuristic function that counts number of tiles out of place"""
    name = 'Out of Place Heuristic (OOP)'

def h(self, node):
    """8 puzzle heuristic: number of tiles 'out of place' between a node's state and the goal"""
    mismatches = 0
    for (t1,t2) in zip(node.state, self.goal):
        if t1 != t2: mismatches += 1
    return mismatches
Path_cost method

Since path cost is just the number of steps, we can use the default version define in Problem

```python
def path_cost(self, c, state1, action, state2):
    """Return cost of a solution path that arrives at state2 from state1 via action, assuming cost c to get up to state1. If problem is such that the path doesn't matter, this function will only look at state2. If the path does matter, it will consider c and maybe state1 and action. The default method costs 1 for every step in the path."""
    return c + 1
```
Example

python> python p8.py 10

Problems using 10 random steps from goal

Using No Heuristic (NIL) from *32415678 to *12345678
  72 states, 27 successors, 40 goal tests, 0.002507 sec
  Solution of length 5

Using Out of Place Heuristic (OOP) from *32415678 to *12345678
  32 states, 11 successors, 17 goal tests, 0.001228 sec
  Solution of length 5

Using Manhattan Distance Heuristic (MHD) from *32415678 to *12345678
  48 states, 16 successors, 24 goal tests, 0.002736 sec
  Solution of length 5
Example

>> Python p8.py 50
Problems using 50 random steps from goal

*61724358 => *12345678 using No Heuristic
    Solution length 19
    52656 states, 19120 successors, 19122 goal tests (262.9092 sec)

*61724358 => *12345678 using Out of Place Heuristic
    Solution length 19
    32942 states, 12306 successors, 12308 goal tests (96.4233 sec)

*61724358 => *12345678 using Manhattan Distance Heuristic
    Solution length 19
    34412 states, 12633 successors, 12635 goal tests (100.9926 sec)