Python Template for Deterministic Dynamic Programming

This template assumes that we are minimizing the objective and states are nonnegative whole numbers, and stages are numbered starting at 1.

```
hugeNumber = float("inf")
Initialize all needed parameters and data
stages = number of stages

f = [[0]*((highest-numbered state)+1) for t in range(stages + 2)]
x = [["uninitialized"]*((highest-numbered state)+1) for t in range(stages + 1)]

If not zero, set each f[stages+1][s] to the terminal value of being in state s at the end
For states that are not allowed, use hugenumber for minimization, -hugenumber for maximization

for t in range(stages,0,-1) :
    # End of t loop
    If necessary, determine which states are possible at stage t
    for i in (states that are possible at stage t) :
        # End of i loop
        Determine set of decisions d which are possible from this state and stage
        value = hugeNumber  # use -hugenumber instead hugenumber if maximizing

        for d in (set of allowed decisions d) :
            j = (resulting next state)
            Compute immediate costs and/or rewards from decision d
            moveValue = (immediate costs and/or rewards) + f[t+1][j]
            if moveValue < value :  # use > instead of < if maximizing
                value = moveValue
                bestMove = d

        # End of d loop
        f[t][i] = value
        x[t][i] = bestMove

# End of t loop

print("Optimal cost is " + str(f[1][initial state]))

solutionString = "(something explanatory) :

solutionString += " " + str(x[t][i]) (more explanation could be inserted in output here)

print(solutionString)
```

See overleaf for more notes:
If your states are not nonnegative integers, you can set up \( f \) and \( x \) with

\[
\begin{align*}
  f &= [\text{dict()} \text{ for } t \text{ in range(stages + 2)]} \\
  x &= [\text{dict()} \text{ for } t \text{ in range(stages + 1)]}
\end{align*}
\]

This will make \( f \) and \( x \) into Python dictionaries, which are array-like objects which will accept any kind of data as an index, at the cost of running more slowly. In this case, immediately after creating these objects, you must explicitly initialize \( f[\text{stages+1]} \) as follows:

\[
\text{for } s \text{ in (states that can be reached after the last stage) :} \\
  f[\text{stages+1}][s] = \text{(terminal value of being in state } s \text{ at the end)}
\]

For states that are not allowed, use \text{hugenumbe}r for minimization, and \(-\text{hugenumbe}r\) for maximization.

If you use the \text{dict} approach, unlike the case on the first page in which we use a two-level list/array initialized to zero, you will need to do this end-stage initialization for all states that are reachable at the end, even if their terminal value is zero. After this, the rest of the code on the first page (starting with the loop over the stages \( t \)) should be usable without alteration.