Python Template for Stochastic Dynamic Programming

Assumptions: the states are nonnegative whole numbers, and stages are numbered starting at 1.

```
import numpy
hugeNumber = float("inf")

Initialize all needed parameters and data
stages = number of stages

f = numpy.zeros([stages + 2, (highest-numbered state) + 1])
x = numpy.zeros([stages + 1, (highest-numbered state) + 1])

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

for t in range(stages,0,-1) :

    for i in (possible states) :

        Determine set of decisions d which are possible in this stage/state combination
        value = -hugeNumber if maximizing or hugenumber if minimizing

        for d in (set of allowed decisions d) :

            Compute rewards/costs that are not random
            moveValue = (net rewards/costs that are not random)

            for r in (set of random outcomes r) :
                  j = (resulting next state)
                  Compute rewards/costs that depend on r
                  moveValue += (probability of r)*((rewards/costs depending on r) + f[t+1,j])
                  #  If net present value is involved, beta*f[t+1,j]) instead, where
                  # beta = 1/(1 + r) is the discount factor

            if moveValue > value :    (use < instead of > if minimizing)
                value = moveValue
                bestMove = d

        # End of d loop

        f[t,i] = value
        x[t,i] = bestMove

    # End of i loop

# End of t loop

print("Optimal solution value is " + str(f[1,(initial state)]))
print("In stage 1, (describe decision) " + str(x[1, (initial state)])
for t in range(2,stages+1) :
    print("In stage " + str(t) + ":")
    for i in (possible states) :
      print("  If (describe state) " + str(i) + ", (describe decision) " + str(x[t,i])
```