Dynamic Pricing Competition
Scenario descriptions

Scenario 3: Large market with horizontal product differentiation

In this scenario, all players compete against each other in a single market, where each player sells a single product with unbounded inventory. We run a large number of S simulations. In each simulation all players compete against each other during $T = 1000$ discrete time periods indexed by $t = 1, \ldots, T$. Let n denote the total number of players. In each time period $t \in \{1, \ldots, T\}$ all players $i = 1, \ldots, n$ determine a price $p_{i,t} \in \{0.00, 0.01, \ldots\}$, after which sales $s_{i,t}$ is realized, for each player $i = 1, \ldots, n$. Sales is realized according to an undisclosed sales-generating mechanism that may be different in each simulation (but, conditionally on selling prices, statistically identical in different time periods within the same simulation). The prices are observable for all players, but each player can *only observe his/her own sales*. After sales has realized, each player $i = 1, \ldots, n$ earns revenue $p_{i,t}s_{i,t}$ and one proceeds to the next time period. The objective is to maximize expected revenue earned over the whole time horizon $t = 1, \ldots, T$, averaged over all simulations. The following domain knowledge is available: (i) sales is stationary (ii) prices larger than 100 will generate a negligible amount of sales (regardless the other player's prices), and (iii) demand is influenced by some form of horizontal product differentiation.

Sample code Scenario 3:

```
import numpy as np

def p(prices_historical=None, demand_historical=None, information_dump=None):
    """
    this pricing algorithm would return the minimum price used
    by any competitor in the last iteration, it returns a random
    price if it is the first iteration

    input:
        prices_historical: numpy 2-dim array: (number competitors) x (past iterations)
                           it contains the past prices of each competitor
                           (you are at index 0) over the past iterations
        demand_historical: numpy 1-dim array: (past iterations)
                           it contains the history of your own past observed demand
                           over the last iterations
        information_dump: some information object you like to pass to yourself
                          at the next iteration
    """

    if demand_historical is None :
        return ( round(np.random.uniform(30,80),1) , None)

    next_price = np.min(prices_historical[1:, -1])
    return ( round(next_price,1) , information_dump)
```