Dynamic Pricing Competition
Scenario descriptions

Scenario 1: Duopoly with inventory constraints

In this scenario, each pair of submitted algorithms (called 'players') will compete against each other in an iterated duopoly with finite inventories. For each pair of players (henceforth called player 1 and player 2), we will run a large number of S simulations. Each simulation consists of $T = 10000$ discrete time periods, indexed by $t = 1, \ldots, T$. The time horizon is divided into 100 consecutive non-overlapping selling seasons, each consisting of 100 time periods. Thus, the j-th selling seasons consists of time periods $100(j-1)+1, \ldots, 100j$, for $j = 1, \ldots, 100$. The inventory of player $i \in \{1, 2\}$ at the beginning of period $t \in \{1, \ldots, T\}$ is denoted by $c_{i,t}$. At the beginning of a selling season, the inventory of both players is set to $C = 80$; we thus have $c_{i,100(j-1)+1} = C$ for all $i = 1, 2$ and $j = 1, \ldots, 100$. In each time period $t \in \{1, \ldots, T\}$, both players $i = 1, 2$ set a price $p_{i,t} \in \{0.00, 0.01, \ldots\}$, after which sales $s_{1,t} \in \{0, 1, \ldots, c_{1,t}\}$ for player 1 and sales $s_{2,t} \in \{0, 1, \ldots, c_{2,t}\}$ is realized according to an undisclosed sales-generating mechanism that may be different in each simulation (but, conditionally on selling prices and the number of sellers having inventory available, statistically identical in different selling seasons within the same simulation). Each player can observe its own prices, its own sales, the competitor's prices, and whether or not the competitor is out-of-stock (but not the precise sales data of the competitor). After sales has realized, player 1 earns revenue $p_{1,t}s_{1,t}$ and player 2 earns revenue $p_{2,t}s_{2,t}$, and one proceeds to the next time period. Unsold inventory at the end of a selling season perishes and has no salvage value. The objective is to maximize expected revenue earned over the whole time horizon $t = 1, \ldots, T$, averaged over all S simulations, and over all plays against different competitors. The following domain knowledge is available: (i) within a selling season, demand is weakly increasing over time, and (ii) prices larger than 100 will generate a negligible amount of sales, regardless the competitor's price.

Sample code Scenario 1:

```
import numpy as np

def p(current_selling_season, selling_period_in_current_season,
        prices_historical_in_current_season=None,
        demand_historical_in_current_season=None,
        competitor_has_capacity_current_period_in_current_season=True,
     information_dump=None):
    """
    this pricing algorithm would return the minimum price used
    by the potentially multiple competitors in the last iteratiom
    (in our duopoly case, it is the previous competitor price),
    it returns a random price if it is the first iteration and
    1000 if the compatitor is out of stock

    input:
        current_selling_season:
                        int of current selling season 1..100
        selling_period_in_current_season:
                        int of current period in current selling season 1..100
        prices_historical_in_current_season:
                        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_in_current_season:
                        numpy 1-dim array: (past iterations)
                        it contains the history of your own past observed demand
                        over the last iterations
        competitor_has_capacity_current_period_in_current_season:
                        boolean indicator if the competitor has some free capacity
                        at the beginning of the current period/ selling interval
        information_dump: some information object you like to pass to yourself
                        at the next iteration
    """

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

    if not competitor_has_capacity_current_period_in_current_season:
        return ( 1000.0 , "Competitor sold out, let's go to 1000")

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