CS1010E: Programming Methodology

PE2 (2019/2020 Sem 2)

Files

- `PE2.pdf`
- `Question1.py`
- `Question2.py`
- `bid_info.csv`
- `small_bid_info.csv`

Coursemology

- `Past PE2 > CS1010E 2019/20 Sem 2`

Questions

Question 1: Auction

You are given the following csv file: `bid_info.csv` with the following columns arranged in the given order (*i.e.*, the order is fixed).

TITLE	ITEM_NUM	BIDDER	PRICE

The details of the data is as follows:

- Each row represents a single bid
- Every bidder (*i.e.*, column **BIDDER**) may bid to supply a specific item (*i.e.*, column **ITEM_NUM**) in the project (*i.e.*, column **TITLE**) for a specific price (*i.e.*, column **PRICE**).
- Every bid for a specific project (*i.e.*, column **TITLE**) has a *unique* price (*i.e.*, column **PRICE**).

The *intended* data type for each column is as follows:

- **TITLE** : `str`
- **ITEM_NUM**: `int`
- **BIDDER** : `str`
- **PRICE** : `float`

Note, however, that the data is always read as string (`str`) in a `.csv` file. You have to perform your own conversion.

Note

- You are guaranteed that the order of the columns is as stated above.
- You are guaranteed that there are no blank data in any row and in any column.
- The rows may be randomly ordered.
- Your code should work and will be tested with other data files with the same format.
- You are NOT given an implementation for `map`/`filter`/*etc.*. You may copy the implementation given in notes or you may use the built-in implementation.

- You are given the function `read_csv` to read the `.csv` file and returns a list of list (`list` of `list`).

continue on the next page...

1.1 Vickrey Auction [20 marks]

> *A Vickrey auction is a type of sealed-bid auction. Bidders submit written bids without knowing the bid of the other people in the auction. The highest bidder wins but the price paid is the second-highest bid. This type of auction is strategically similar to an English auction and gives bidders an incentive to bid their true value. The auction was first described academically by Columbia University professor William Vickrey in 1961 though it had been used by stamp collectors since 1893. In 1797 Johann Wolfgang von Goethe sold a manuscript using a sealed-bid, second-price auction.*

In short, the winner is selected based on the highest price but the price to be paid is the second highest price. However, when there is only one bidder, the price is the highest bid price instead. Since the bid price is *guaranteed* to be *unique*, the second highest bidder (*if any*) is clear.

Question

Write the function `vickrey(bid_file, title, item_num)` that takes as inputs the file to bidding data (`str`), the title of the project (`str`) and the item number (`int`). The function returns a pair (`tuple`) of winning bid name (`str`) and the price (`float`) to be paid based on the Vickrey auction method. If there is no bid, return an empty tuple (`tuple`).

Assumptions

- `bid_file` is a path to a valid file name (*in Coursemology*)

Sample Run #1

```
>>> vickrey('bid_info.csv', 'AGGREGATES, HOT MIX ASPHALT(HWYS)', 9)
('TILCON NEW YORK INC', 38.2)
```

Sample Run #2

```
>>> vickrey('bid_info.csv', 'AGGREGATES, HOT MIX ASPHALT(HWYS)', 10)
('TILCON NEW YORK INC', 36.97)
```

Sample Run #3

```
>>> vickrey('bid_info.csv', 'AGGREGATES, HOT MIX ASPHALT(HWYS)', 13)
('TILCON NEW YORK INC', 36.76)
```

1.2 Successful Companies [30 marks]

Given the Vickrey auction above, we know the *winner* for a particular title and item. We can then define a successful company as follows:

> For each project, the successful company is the company with the ***most*** number of win on items in the given project.

Since there can be two companies with the same number of wins, we group the successful companies for a given project as a set (`set`). Your task is to return all the successful companies for all project in the `.csv` file as a dictionary (`dict`). The key of the dictionary are the project titles and the value of the dictionary are the successful companies for the given project title. Thus, your return value should be dictionary of set.

Question

Write the function `success(bid_file)` to return all the successful companies for all project in the given file (`bid_file`) as a dictionary of set (`dict` of `set`).

Assumptions

- `bid_file` is a path to a valid file name (*in Coursemology*)

Sample Run #1

```
>>> success('small_bid_info.csv')
{'AWD POLICE SEDAN - NYPD': {'BEYER OF MORRISTOWN LLC'}, 'AGGREGATES, HOT MIX
     ASPHALT(HWYS)': {'TILCON NEW YORK INC'}}
```

Question 2: Catan

For this question, we are going to design and model a simplified version of the popular boardgame Catan. The game is played by up to 4 players. The simplified rule of the game is as follows:

- Each player build a house on a tile.
  - A player may have multiple houses on a tile.
  - Each tile may only have a maximum of three houses on it.
- Each tile may produce a resource corresponding to the tile when the number is rolled.
  - If a player has a house on the tile, the player receives the resource produced by the tile.
  - If a number 7 is rolled, all players will be robbed. The rule for being robbed is described in Question 2.1.
- Players may trade resources with another player.
- There are 2 kinds of resources: wood and brick.

Your task is to model two classes `Player` and `Tile`.

2.1 Player [25 marks]

A class `Player` is created with one input: the player colour (`str`). This will create the player with the given colour and having 3 wood and 3 brick resources. The class `Player` must support the following methods:

- `get_colour()` : Returns the colour (`str`) of the player.
- `get_resources()` : Returns a pair of number (`tuple` of `int`) with the following format `(wood, brick)` where `wood` is the number of wood and `brick` is the number of brick the player has.
- `robbed()` : Returns a string (`str`) based on the following condition:
  - If the player has fewer than 8 total resources (*i.e.*, sum of both wood and brick) then the string `"<player colour> is safe from robber"` is returned where `<player colour>` refers to the player colour.
  - Otherwise, the number of wood and brick the player owns is halved (*rounded down*) then the string `"<player colour> is robbed of <total resources removed> resources "` is returned where `<player colour>` refers to the player colour and `<total resources removed>` refers to the sum of wood and brick that was robbed.
- `trade(other_player, woods, bricks)` : Takes in as input the other player (`Player`) and two integers (`int`) corresponding to the number of wood to be traded with the number of brick. The function returns a string (`str`) based on the following conditions:
  - If the player has fewer woods than `woods` then the string `"<player colour> does not have enough wood"` is returned where `<player colour>` refers to the player colour.
  - If the other player has fewer bricks than `bricks` then the string `"<other player colour> does not have enough brick"` is returned where `<other player colour>` refers to the colour of `other_player`.
  - Otherwise, the trade happens then the string `"<player colour> trades with <other player colour>"` is returned where `<player colour>` refers to the player colour and `<other player colour>` refers to the colour of `other_player`.
- `build(tile)` : Takes in as input a tile (`Tile`) and returns a string (`str`) based on the following conditions:
  - If the player has fewer woods than 2 wood or fewer than 2 brick then the string `"<player colour> does not have enough resources to build"` is returned where `<player colour>` refers to the player colour.
  - If there are already 3 buildings in `tile` *regardless* of the colour, then the string `"Tile <tile number> does not have any space"` is returned where `<tile number>` refers

to the tile number (*see Question 2.2*).

- Otherwise, 2 wood and 2 brick are removed from the player then the string `"<player colour> builds on Tile <tile number>"` is returned where **`<player colour>`** refers to the player colour and **`<tile number>`** refers to the tile number (*see Question 2.2*).

Question

Implement the class `Player`.

2.2 Tile [25 marks]

A class `Tile` is created with two inputs: the tile number (`int`) and a resource (`str` which is either `"wood"` for wood or `"brick"` for brick). This will create a tile without any building. The class `Tile` must support the following methods:

- `get_number()` : Returns the tile number (`int`) of the tile.
- `get_resources()` : Returns the resources produced by the tile (`str` which is either `"wood"` for wood or `"brick"` for brick).
- `get_building()` : Returns the number of buildings (`int`) on the tile.
- `produce()` : Returns a string (`str`) based on the following condition:
  - If there are no buildings on the tile, then the string `"Tile <tile number> does not produce any <tile resource>"` is returned where **`<tile number>`** refers to the tile number and **`<tile resource>`** refers to the resource the tile produce.
  - Otherwise, the string `"Tile <tile number> produces <number of building> <tile resource>"` is returned where **`<tile number>`** refers to the tile number, **`<number of building>`** refers to the number of building (*of any colour*) on the tile and **`<tile resource>`** refers to the resource the tile produce.

Question

Implement the class `Tile`.

continue on the next page...

Sample Run

```
>>> red    = Player("Red")
>>> blue   = Player("Blue")
>>> yellow = Player("Yellow")
>>> brick1 = Tile(1, "brick")
>>> wood1  = Tile(2, "wood")
>>> brick2 = Tile(3, "brick")
>>> wood2  = Tile(4, "wood")
>>> red.build(brick1)
Red builds on Tile 1
>>> brick1.produce()
Tile 1 produces 1 brick
>>> red.build(wood1)
Red does not have enough resources to build
>>> blue.build(wood1)
Blue builds on Tile 2
>>> yellow.build(brick1)
Yellow builds on Tile 1
>>> brick1.produce()
Tile 1 produces 2 brick
>>> brick1.produce()
Tile 1 produces 2 brick
>>> wood2.produce()
Tile 4 does not produce any wood
>>> wood1.produce()
Tile 2 produces 1 wood
>>> wood1.produce()
Tile 2 produces 1 wood
>>> wood1.produce()
Tile 2 produces 1 wood
>>> blue.get_resources()
(4, 1)
>>> red.get_resources()
(1, 4)
>>> blue.trade(red, 5, 1)
Blue does not have enough wood
>>> blue.trade(red, 1, 5)
Red does not have enough brick
>>> blue.trade(red, 2, 2)
Blue trades with Red
>>> blue.get_resources()
(2, 3)
>>> red.get_resources()
(3, 2)
```

– End of Paper –