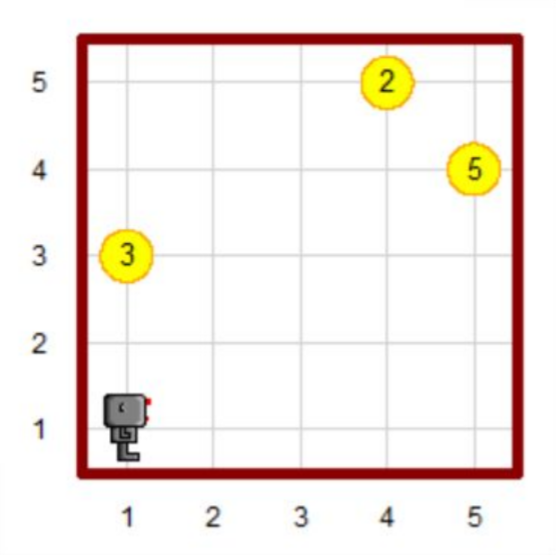
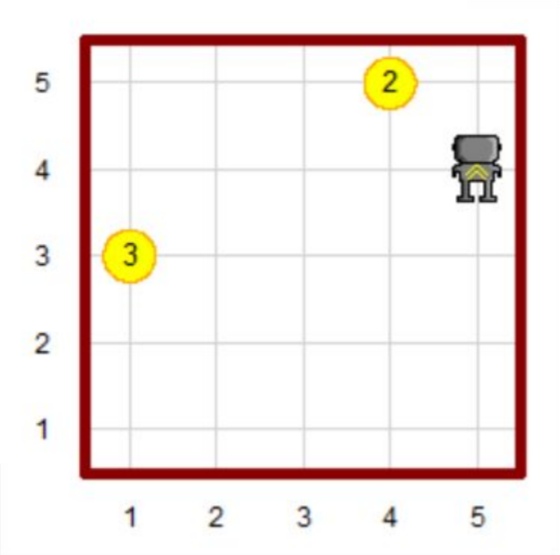
1. **Pick Up the Biggest Beeper Stack (3 pts)**

Given a world that has more than one beeper dropped on the ground, write a code that sends a robot to the position where there are the maximum number of beepers and pick up all beepers at that position.**Example:**

**Assumptions:**

1. The world size is n x m, where both n and m are integers between 1 and 10.
2. No two stack will have the same number of beepers.
3. There will be at least one stack in each world.
4. The world will not have walls in the inside (rectangular shaped wall with no wall inside).

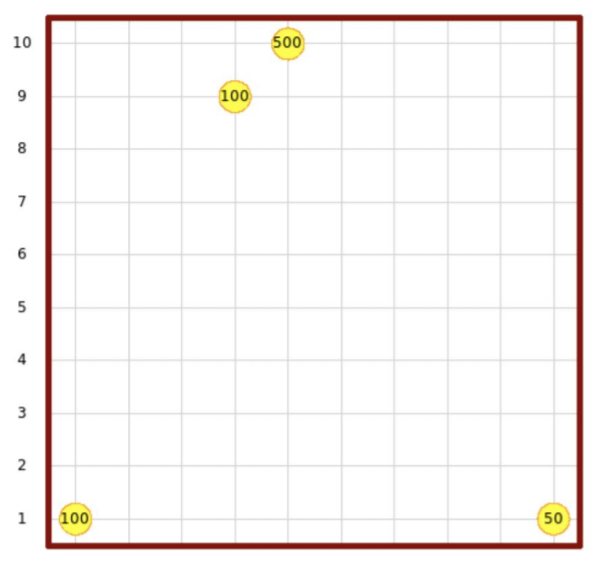
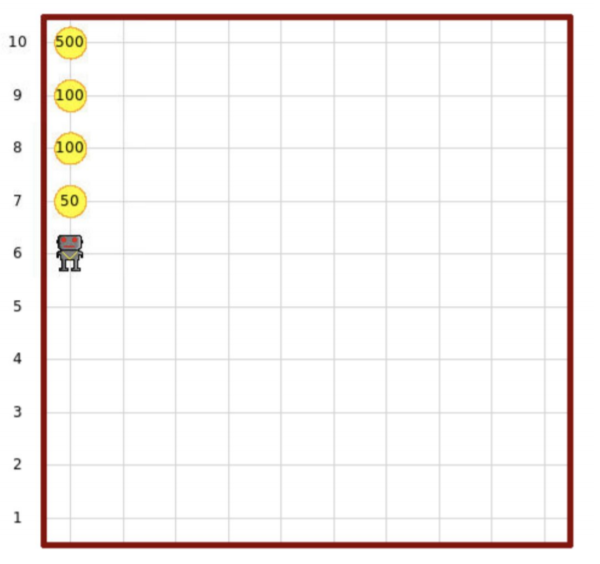
**List of Functions of Robot**

* set\_trace(color) : If a color is given with the set\_trace function, then the robot will trace the path it took with the color given. If no color is given, it will turn off tracing.
* set\_pause(delay) : Use this to set a pause to be made after each move of the robot. The delay is in units of seconds/move.
* get\_pos(): Returns current robot position in the form of (x, y).
* turn\_left() : Rotate the robot left by 90 degrees.
* move() : Move one street/avenue in direction where the robot is facing.
* front\_is\_clear() : Returns True if no wall is in front of the robot.
* left\_is\_clear() : Returns True if no wall is to the immediate left of the robot.
* right\_is\_clear() : Returns True if no wall is to the immediate right of the robot.
* facing\_north() : Returns True if the robot is facing north.
* carries\_beepers() : Returns True if the robot is carrying any beepers.
* on\_beeper() : Returns True if any beepers are present at the current robot position.
* pick\_beeper() : The robot picks up one beeper at the current location.
* drop\_beeper() : The robot drops one beeper at the current location.

1. **Rearrange Beeper Stacks (3 pts):**

Given a world, write a code that makes Hubo rearrange the beeper stacks in the world into the left-most column by stack size in a descending order from the top row. (Note, Hubo may not carry more than 1 stack at the same time)

**Example:**

**Assumptions:**

1. The world size is n x m, where both n and m are integers larger than or equal to 1.
2. The number of beeper stacks is not larger than the height of the world.
3. **Top 200 Times World University Rankings (4 pts)**

The Times Higher Education has been providing trusted performance data on universities for students and their families, university academics, university leaders, governments and industry, since 2004.

In this problem, you will be given a csv file containing data about the top 200 universities from 2011 to 2016. Each line of the csv file will have the following format:

<Year>, <World Rank>, <University Name>, <Country>

**Task 1 (1 pt):** Make a dictionary top\_200\_university\_dict using the csv file given with the following requirements:

* The keys of the dictionary are the names of the university.
* The value of each key should be a dictionary which has keys ‘Country’, ‘Ranking’ and ‘Year’.

**Example:**

top\_200\_university\_dict = {

...

'Korea Advanced Institute of Science and Technology (KAIST)': {

'Country':'South Korea’

'Ranking':[79, 94, 68, 56, 52, 148],

'Year':[2011,2012,2013,2014,2015,2016]

},

...

'Yonsei University': {

'Country':'South Korea’

'Ranking':[190, 183, 190],

'Year':[2011,2013,2014]

},

...

}

**Task 2 (1 pt):** Make a function university\_list(dictionary) that takes the dictionary made in task 1 as input and outputs a **list** of all the universities in the dictionary. Also, make a function university\_set(dictionary) that takes the dictionary made in task 1 as input and outputs a **set** of all the universities in the dictionary.

* The university list should not contain duplicate items.
* Please do not use the set(list) function to make a list into a set.

**Example:**

>>> university\_list(top\_200\_university\_dict)

['Harvard University', 'California Institute of Technology', 'Massachusetts Institute of Technology', 'Stanford University', 'Princeton University', 'University of Cambridge', ...

**Task 3 (1 pt**): Make a function check\_ranking(dictionary, name, year) that takes a dictionary, university name, and year as inputs and outputs the ranking of the given university in the given year. This function should be able to handle exceptional cases (given name not in dict, no result for given year, etc…). We assume that dictionary argument is given correctly.

**Example:**

>>> dict = top\_200\_university\_dict

>>> check\_ranking(dict, 'wrong univ name', 2012)

No matching university name.

>>> check\_ranking(dict, 'Harvard University', 2000)

No result for given year.

>>> check\_ranking(dict, 'Harvard University', 2012)

6

**Task 4 (1 pt**): Make a function average\_ranking(dictionary, name [, begin\_year [, end\_year]]) that takes a dictionary, and university name (and optionally begin\_year and/or end\_year) and returns the average ranking of the given university from begin\_year to end\_year. If begin\_year or end\_year are not explicitly given, they are, by default, 2011 and 2016. If the university is out of ranking in a certain year, assume the ranking is 200. Again, this function should handle exceptional cases as in Task 3 and we also assume that dictionary argument is given correctly.

* Which university(s) has the highest average ranking between 2011 and 2013?
* Which university(s) has the lowest average ranking between 2014 and 2016?

**Example:**

>>> dict = top\_200\_university\_dict

>>> average\_ranking(dict, 'wrong univ name', 2012)

No matching university name.

>>> average\_ranking(dict, 'Harvard University', 2011, 2016)

2.8333333333333335

**Submission**

Please submit your code by emailing them to **BOTH**: [steve-kim@kaist.ac.kr](mailto:steve-kim@kaist.ac.kr) and [dkcjd2000@kaist.ac.kr](mailto:dkcjd2000@kaist.ac.kr)