

Function practice.py: Implement python functions

Your task is to implement a few python functions to get some practice. If you do not understand certain terms, be sure to Google them.

Name	Arguments	Returns
bubble_sort	number_list	Returns the sorted number_list. Implement the bubble sort algorithm as shown below that sorts from smallest to largest . Note: your implementation should not modify the original list number_list.
third_min	number_list	Returns the third smallest number in in the given list number_list. Hint: use your implementation of bubble sort to do this easily.
second_max	number_list	Returns the second largest number in in the given list number_list. Hint: use your implementation of bubble sort to do this easily. Also use negative indexing feature in python.

Algorithm 1: Bubble sort

Data: Input array $A[]$

Result: Sorted $A[]$

$int\ i, j, k;$

$N = length(A);$

for $j = 1$ **to** N **do**

for $i = 0$ **to** $N-j$ **do**

if $A[i] > A[i+1]$ **then**

$temp = A[i];$

$A[i] = A[i+1];$

$A[i+1] = temp;$

end

end

end

class practice.py: Implement python classes

Your task is to implement a simple Courses class that stores courses, their instructor, and the number of students. The following are the details:

Class name		Inherits from
Courses		object
Courses Constructor		
Constructor arguments		Constructor body
course_name_list, instructor_name_list, num_students_list		Create a class dictionary variable self.courses_dict where the key is the name of the course and value is a pair (instructor, num_students_list). Use the list arguments to populate this dictionary.

Courses Functions			
Name	Arguments	Returns	implementation
addCourse	course_name, instructor_name, num_students	Returns nothing.	Add the course to self.courses_dict. If the course already exists, then do nothing.
addStudents	course_name, num_students	Returns nothing.	Adds to the student count of the given course in self.courses_dict. If course does not exist, then do nothing.
adjustInstructor	course_name, instructor_name	Returns nothing.	Adjusts/changes the instructor of the given course in self.courses_dict. If course does not exist, then do nothing.
getInstructor	course_name	Returns instructor of course_name.	If course does not exist, then return None.
getNumStudents	course_name	Returns student count of course_name.	If course does not exist, then return None.

ac_simulation.py: Simulate simple reflex agent

Your task is to simulate an environment where a simple air conditioner (AC) reflex agent that will turn on the AC when a min temperature is reached and turn off the AC when a max temperature is reached. To do this, we are going to implement two classes: SimpleACReflexAgent and SimpleACEnvironment.

Class name		Inherits from
SimpleACReflexAgent		object
SimpleACReflexAgent Constructor		
Constructor arguments		Constructor body
min_threshold, max_threshold		Add arguments to self. These are the min and max temperature thresholds.

SimpleACReflexAgent Functions			
Name	Arguments	Returns	implementation
select_action	percept	Returns action selected by agent. Possible actions are "TurnOn", "TurnOff", None.	percept is a list of size 2 where the first element is the current temperature and the second element is a Boolean indicating if the AC is on (true = AC on, and false = AC off). There are three cases to consider: <ul style="list-style-type: none">- The agent should return a string "TurnOn" if the AC is off and current temp is \leq min_threshold.- The agent should return a string "TurnOff" if the AC is on and current temp is \geq max_threshold.- Otherwise agent returns None

Class name		Inherits from
SimpleACEnvironment		object
SimpleACEnvironment Constructor		
Constructor arguments	Constructor body	
ac_agent, starting_temp=28	<p>Add the following four variables to self:</p> <ul style="list-style-type: none"> - self.ac_agent = ac_agent is the AC reflex agent. - self.temperature=starting_temp is the current temperature in the room. - self.num_agent_actions=0 is the number of times the agent performed “TurnOn” and “TurnOff” actions (note this does not count None actions). Initially, 0 actions performed. - self.is_ac_on=False is a Boolean indicating if AC is on or off. Initially, the AC is off. 	

SimpleACEnvironment Functions			
Name	Arguments	Returns	implementation
tick	No arguments	Returns nothing.	<p>This function progresses the world by a single timestep. Each call to tick will do the following in order:</p> <ul style="list-style-type: none"> - Call the AC agent’s select_action function with the current percept: temperature and AC status. With the returned action, we will simulate the effect of the action on the environment - self.num_agent_actions is incremented by 1 if and only if the action was “TurnOn” or “TurnOff”. - update the status of the AC via self.is_ac_on depending on the returned action. - If the AC is on, then increment the current temperature by 1. If the AC is off, decrement the current temperature by 1.
simulate	num_timesteps	Returns nothing.	<p>This function is meant to simulate the environment-agent interaction for num_timesteps iterations. So just implement a simple for-loop where self.tick() is called in each iteration.</p>

server_simulation.py: Simulate model-based agent

Your task is to simulate an environment where a server agent serves water bottles of three sizes small, medium, and large to customers. However, the agent only has a limited stock of each size, and needs to maintain the quantities in memory. In terms of operation, a customer gives the agent a number from 0 to 100 indicating their hydration level. The agent then gives the customer a single bottle whose size depends on the hydration level (more on this later). To simulate this, we are going to implement two classes: `ServerAgent` and `ServerEnvironment`.

Note that this is a model-based agent since there is a need to maintain the store quantities of water bottles in memory (model of the world); this is because the agent does not receive the quantity updates through percepts.

Class name		Inherits from	
ServerAgent		object	
ServerAgent Constructor			
Constructor arguments		Constructor body	
small_count=10, medium_count=10, large_count=10		Add arguments to self. These are the initial quantities of the small, medium, and large water bottles.	
ServerAgent Functions			
Name	Arguments	Returns	implementation
select_action	percept	Returns action selected by agent. Possible actions are “small”, “medium”, “large”, None.	percept is a number between 0 and 100 indicating the hydration level of the customer. There are four cases to consider: - If hydration level is in [0, 33], give a “large” bottle. If there are no more large bottles, return None. - If hydration level is in [34, 66], give a “medium” bottle. If there are no more medium bottles, return None. - If hydration level is in [67, 99], give a “small” bottle. If there are no more small bottles, return None. - If hydration level is >= 100, then return None. Note: when agent gives a bottle, decrement the corresponding count.
storage_empty	No arguments	Returns true if all size quantities are 0, otherwise false.	Hint: can be done in one conditional statement.

For the environment, in each timestep, we are going to draw a random number from 0 to 100 to simulate customer hydration levels. One simple way to do this is through python's `random.randint(a=0, b=130)` function; **import random to use this**. Note we are drawing numbers between 0 and 130 so that we have a higher chance of getting > 100 .

Class name			Inherits from
ServerEnvironment			object
ServerEnvironment Constructor			
Constructor arguments		Constructor body	
server_agent		Add the following four variables to self: - self.server_agent = server_agent is the server agent. - self.num_agent_actions=0 is the number of times the agent performed any action (including None). Initially, 0 actions performed.	
ServerEnvironment Functions			
Name	Arguments	Returns	implementation
tick	No arguments	Returns nothing.	This function progresses the world by a single timestep. Each call to tick will do the following in order (be sure to import random): - Randomly draw a number between 0 and 130 using random.randint(a=0, b=130). - Call the server agent’s select_action function with the current percept (hydration level). With the returned action, we will simulate the effect of the action on the environment. - self.num_agent_actions is incremented by 1 regardless of action taken.
simulate	No arguments	Returns nothing.	This function is meant to simulate the environment-agent interaction until the agent runs out of all size quantities (hint use storage_empty() to check this). So just implement a simple while-loop where self.tick() is called in each iteration.

You can use [test_pal.py](#) file to test your code. Here is the output from my implementation.

```
[3, 2, 1, 0, -1, -2, -3] [-3, -2, -1, 0, 1, 2, 3]
-1 2

[79, 31, -73, -47, -18, -25, 45, 89] [-73, -47, -25, -18, 31, 45, 79, 89]
-25 79

ICS-501, Moayad, 28
ICS-502, Irfan, 16
ICS-104, Ahmed, 36
MATH-101, Khalid, 45
-----Making changes-----
ICS-501, Moataz, 28
ICS-502, Irfan, 32
ICS-104, Mohammed, 136
MATH-101, Khalid, 45

AC simulation #1 starting conditions:
min-max thresholds: 0, 100
env temperature: 50, num_agent_actions: 0, is_ac_on: False
-----simulating for 60 timesteps-----
env temperature: 10, num_agent_actions: 1, is_ac_on: True

AC simulation #2 starting conditions:
min-max thresholds: 15, 25
env temperature: 20, num_agent_actions: 0, is_ac_on: False
-----simulating for 48 timesteps-----
env temperature: 18, num_agent_actions: 5, is_ac_on: True

Server simulation #1 starting conditions:
small, medium, large counts: 5, 5, 10
env num_agent_actions: 0
-----simulating until storage is done-----
env num_agent_actions: 45

Server simulation #2 starting conditions:
small, medium, large counts: 100, 50, 50
env num_agent_actions: 0
-----simulating until storage is done-----
env num_agent_actions: 352
```