STUDYDADDY

Get Homework Help From Expert Tutor

Get Help

Foundations of Programming 2016-2017

Assignment 2 – 100 Points (PASS mark: 40 points)
(Counts towards 17.5% of the final grade for this module)

Part (A) of this assignment contains 40 marks. Part (B) contains 60 marks. You are required to complete Part (A) before tackling Part (B). The different marks are reported in square brackets, under each question. **"Penalty marks"** (explained below) **are reported in RED**.

NOTE: For this coursework **you should produce 2 scripts, one for Part A and one for Part B.** If you prefer to produce a single script that addresses **both Part A and part B,** please submit **two copies of the same script**. Please refer to the guidelines provided on the V.L.E. (learn.gold.ac.uk, "Term2 Assignment") for the format and labelling of the submitted scripts.

IMPORTANT: **beware of the "forbidden" methods listed below, which carry penalty marks.** Each use of a "forbidden method" in your code will cause a **penalty of 20 marks (i.e., 20 points will be subtracted from your final mark).** Repeated usage of a forbidden method will lead to multiple penalties (e.g., if your script uses a forbidden method in *two* separate points, your grade will be penalized by 2*20 = 40 points). **Penalties apply to both Part (A) and Part (B)**.

The forbidden methods / functions are:

- **`pop, insert, del, index, find, remove, sort, sorted`**
- **slicing** (i.e., any usage of the '**:**' operator inside square brackets [ ]) is also forbidden.

Only "`append`" and "`reverse`" can be used freely and without incurring in penalties.

NOTE: ***You are not allowed to use any Python modules other than the "turtle" module*** *(i.e., your program may only contain the "import turtle" statement, and no other imports).* ***A PENALTY of 80 marks will be applied for using any other external modules or libraries*****.**

Queuing at the box office

A "FIFO" (First-In First-Out) queue is a data structure in which the first element that "enters" is also the first one to "leave". This is the most familiar type of queue (also known as "first-come, first-served" system), which you (should) normally experience, e.g., when queuing at a tickets office.

(A) Write a Python program that uses the "list" data type to create and manage a FIFO queue of strings.

Your program should create, initialize and maintain a queue in which each element is a string. The program should interact with the user by means of a "menu" of 6 possible options. To make a choice, the user should be asked to enter a one-character string, followed by "Enter". For each choice the program should react with a different behavior, as specified below:

CHOICE	PROGRAM BEHAVIOUR
"P"	Print all the elements currently in the queue; the strings should be printed *on the same line*, separated by a single space (and not be included in square brackets). **[5 marks]**
"A"	"Add" or "Append": when this option is chosen, the program should ask the user to enter a (second) string, and append this new element at the end of the current queue (you may assume the second string entered by the user is non-empty). **[5 marks]**

"N" "Next": remove the first element in the queue – the new queue should contain all the elements of the original one except the first one; if the current queue is empty, choosing this option should leave the queue as is.

[10 marks]

"L" ("Leave" the queue): when this option is chosen, the program should ask the user to enter a second string s, and remove the first occurrence of s from the queue. If the queue contains no occurrences of s, or if the queue is empty, the queue should be left unchanged, and no error should occur. (You can assume that the string s entered by the user is non-empty).

[15 marks]

"M" print the memory location where the list containing the current queue is stored.

[3 marks]

"Q" Quit (end the program).

[2 marks]

- After each choice, the program should ***not*** print the set of possible options again, but just prompt the user for the next choice (e.g., print "Make another choice:"). **This behaviour should be repeated *until the user chooses option "Q"*. Any string entered by the user not amongst those listed above should be disregarded and have no effects on the queue –** for any such incorrect input, the program should just prompt the user again for another choice.

[15 penalty marks if your code does not behave as specified by the above bullet point]

(B) Write ANOTHER Python program which builds upon the script you have written for Part (A), and which (in addition to *ALL* of Part A) implements ALSO the following additional features (you can choose to implement *ALL* or *ANY SUBSET* of the following four questions B.1— B.4):

B.1) Ensure that the queue never grows longer than 10 – i.e., if the current queue contains 10 elements and the user chooses option "A" above, the program should not ask the user for a second string but just output the message: "*The queue is full – please try later*".

[5 marks]

B.2) Enable a further option ("S", for Sort) which, when chosen, will cause the existing queue to be sorted in alphabetical order (i.e., the new queue should contain the same elements as the original one, but sorted alphabetically). Sorting an empty queue has no effects.

[15 marks]

B.3) Ensure that the parts of the code which provide the functionalities required by **Part (A)** maintain *one and the same* list-object in memory throughout the interaction with the user. (E.g., applying either the "N" or "L" options should leave the memory location at which the current list is stored unaltered).

[20 marks]

(*continues on the next page*)

(continued from Part B)

B.4) Use Python's turtle module to provide a graphical display of the current state of the queue.

Each element (string) in the queue should be represented by a different turtle object on the graphic window (canvas). All turtles should be on the same line and face the same direction (see examples below). Every time the user modifies the queue, the canvas should be revised so as to reflect the new situation, as explained below:

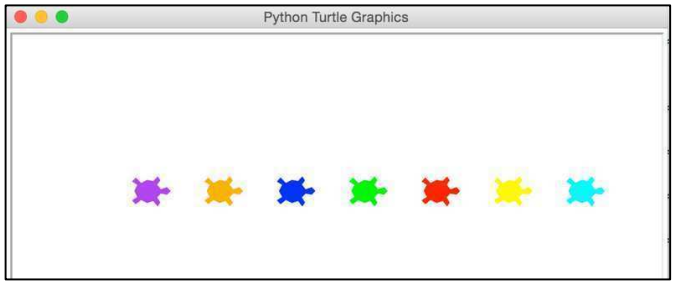
- if a new element is added to the queue (option "A" in part **(A)**), a new turtle should be created and positioned at the end of the line, after the last turtle (you can choose the distance between two consecutive turtles to be what you find most appropriate);
- if the first element in the queue is removed (option "N" in part **(A)**), the first turtle in line should disappear from the screen and the remaining turtles shift forward by one position (without leaving any traces!);
- if an element is removed from the "middle" of the queue (option "L" in part **(A)**), the corresponding turtle should disappear from the screen and the turtles "behind it" in the queue should move forward by one place (see examples below).

Each turtle should have a colour, chosen amongst 8 different ones. For a queue containing up to 8 elements, all corresponding turtles should have different colours; if the queue grows longer, there can be different turtles of the same colour.

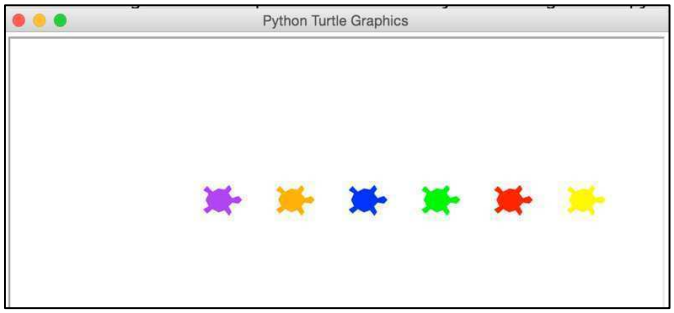
[20 marks]

Example:

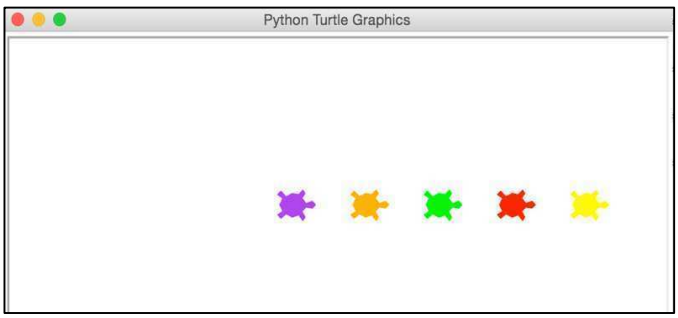
A queue containing 7 elements (strings) is visualised as a line of 7 turtles, each of a different colour.
(Note that the strings – which can be thought of as corresponding to the turtles' "names" – should *not* be visualised here).



The user chooses option "N" (see part **(A)**) and the first string in the queue is deleted. As a result, the corresponding turtle (the first in line, in cyan) is no longer displayed, and all other turtles have stepped forward by one position:



The user chooses option "L" (see part **(A)**) and types in a string which is identical to the fourth element in the queue: the corresponding turtle (in blue) is no longer visible, and the two turtles behind it (orange and purple) have moved forward by one position in the queue:



STUDYDADDY

Get Homework Help From Expert Tutor

Get Help