CPSC 3400 Languages and Computation
Winter 2018

Homework 2
Due: Monday, January 29 at 10:55am

Part 1 – Generator Function

Write a Python program (`hw2-1.py`) that contains the following function:

`genLetters(string)`: Takes a string as an input and operates as a generator function. Each call will yield the next letter in the string until it is out of letters. It always yields the letter as a lowercase letter. For instance, if the input string is "CPSC 3400 is fun", it will yield these letters in order: c, p, s, c, i, s, f, u, n.

In addition to this function, you must do the following steps outside the function:

1. Ask the user for a string. You may not make any assumptions regarding the string.

2. Using a for loop and the function `genLetters`, create a dictionary that has lowercase letters for keys and the number of times a letter occurs in the string (lower or uppercase) as values. Assign the dictionary the name `letterTable`.

3. Create a *generator expression* that will yield tuples from the dictionary `letterTable`. The tuples should be in the form (key, value). The tuples should be yielded in alphabetical order. Assign the generator expression the name `genPairs`.

4. Use `genPairs` to print out the tuples. Except for the input prompt in part 1, this is the only step in the program that generates any output.

Sample Run

```
Enter a string: Anna loves to eat bananas!
('a', 6)
('b', 1)
('e', 2)
('l', 1)
('n', 4)
('o', 2)
('s', 2)
('t', 2)
('v', 1)
```

Part 2 – Binary Search Trees

Write a Python program (`hw2-2.py`) that uses Python lists to store a binary search tree of integers. A tree consists of a single list with either three elements [value of root node, left subtree, right subtree] or zero elements [] (represents an empty tree).

Implement the following functions:

- `insert(tree, value)`: Inserts the value into the tree and returns 0 if successful and -1 on error. An error occurs if the value is already in the tree; the tree remains unmodified in this situation. The resulting tree does NOT need to be balanced.

- `search(tree, value)`: Returns true if the value is in the tree and false otherwise.

- `inorder(tree)`: Does an inorder traversal of the nodes. This is to be implemented using a generator function. It yields the next node in the traversal.
    - If you use a recursive function (recommended), you need to use `yield from` when calling `inorder` recursively. You can also implement a non-recursive version of the function.

In addition to these functions, you must do the following steps to create a simple test driver. This code should reside outside the functions. The formatting should be identical to the sample output provided below.

1. Get the name of an input file from the command line (using `sys.argv`). WARNING: Do not prompt the user for a file name.
2. Read the file. Each line will contain a single integer that is to be inserted into the tree. Insert the integers in the order specified in the file. If a duplicate value is encountered, print a message indicating the duplicate value.
3. Print the tree data structure. Simply print the variable, do not try to format the printing in anyway.
4. Create a loop that ranges from 1 to 9 (including both 1 and 9). Each iteration, print whether the number is in the binary search tree or not.
5. Create a for loop that does an inorder traversal of the tree and prints the value of the node each iteration (each on its own line).
6. Using *a list comprehension*, create a list that contains an inorder traversal of the integer values in the list. Print the list to the screen.

Sample Input File and Output

A sample input file is provided on `cs1` at:

`/home/fac/elarson/cpsc3400/hw2/sample.txt`

Here are the contents of that file:

```
20
10
30
5
35
8
25
2
37
8
27
15
22
```

If your program is working correctly, it should have the following output:

```
Duplicate value detected: 8
Step 3:
[20, [10, [5, [2, [], []], [8, [], []]], [15, [], []]], [30, [25, [22, [],
[]], [27, [], []]], [35, [], [37, [], []]]]]
Step 4:
1 NO
2 YES
3 NO
4 NO
5 YES
6 NO
7 NO
8 YES
9 NO
Step 5:
2
5
8
10
15
20
22
25
27
30
35
37
Step 6:
[2, 5, 8, 10, 15, 20, 22, 25, 27, 30, 35, 37]
```

IMPORTANT! Do not assume that testing is complete if your program produces the correct output for this provided input file. During grading, your assignment will be tested with other input files.

Grading

Grading is based on the following rubric and is primarily based on functionality based on how many tests pass or fail. Some parts of the program have additional restrictions that can incur further penalties in addition to any deductions due to functionality (your total deduction cannot exceed the number of points for that part).

Part 1 (18 points)

genLetters	*8 points*
• Works in all tests	(no penalty)
• Fails in one test	-2
• Fails in two tests	-4
• Fails in three or more tests	-8

Additional restriction:

• Is not a function generator	-5

Step 2 (dictionary created using genLetters):	*4 points*
• Works in all tests	(no penalty)
• Fails in one test	-2
• Fails in two or more tests	-4

Step 3 (generator expression to produce tuples):	*6 points*
• Works in all tests	(no penalty)
• Fails in one or two tests	-2
• Fails in three or more tests	-6

Additional restriction:

• Did not yield tuples of the proper form.	-3
• Not a generator expression	-4

Part 2 (32 points)

insert	*8 points*
• Works in all tests	(no penalty)
• Fails in one test	-2
• Fails in two tests	-4
• Fails in three or more tests	-8

Additional restriction:

• Does not return -1 for duplicate values	-2

search	*8 points*
• Works in all tests	(no penalty)
• Fails in one test	-2
• Fails in two tests	-4
• Fails in three or more tests	-8

inorder	*8 points*
• Works in all tests	(no penalty)
• Fails in one test	-2
• Fails in two tests	-4
• Fails in three or more tests	-8

Additional restriction:	
Is not a function generator	-5

Test driver	*8 points*
• Works in all tests	(no penalty)
• Formatting issues	-1 (or more)
• Minor functionality issue	-2 (each)
• Major functionality issue	-5 (or more)
• Fails in three or more tests	-8
• Did not use list comprehension in step 6	-3

In addition, programs may lose additional points as follows:

- Up to 10 points may be deducted if programs if they exhibit poor readability or style. See the programming assignment expectation handout.
- 6 point deduction if the file name is not obtained from the command line for part 2.
- Programs that contain syntax errors will receive a zero.

Submitting your Assignment

On `cs1`, run the following script in the directory with your program:

```
/home/fac/elarson/submit/cpsc3400/hw2_submit
```

This will copy the files `hw2-1.py` and `hw2-2.py` to a directory that can be accessed by the instructor. Please be sure to keep the same file names or the submission program will not work. Only the last assignment submitted before the due date and time will be graded. ***Late submissions are not accepted and result in a zero.***