CSE 101: Introduction to Computational and Algorithmic Thinking

Stony Brook University

Lab Assignment #5

Spring 2018

Assignment Due: March 2, 2018 by 11:59 pm

Assignment Objectives

This lab assignment will give you practice with while-loops and Boolean operators.

Getting Started

Visit Piazza and download the "bare bones" file `lab5.py` onto your computer. Open `lab5.py` in PyCharm and fill in the following information at the top:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)
4. the course number (CSE 101)
5. the assignment name and number (Lab #5)

Submit your final `lab5.py` file to Blackboard by the due date and time. Late work will not be graded. Code that crashes and cannot be graded will earn no credit.

Part I: FizzBuzz (20 points)

Write the function `fizz_buzz()`, which takes three arguments in the following order:

- `max_fizz_buzz`: a positive integer
- `fizz`: a prime number
- `buzz`: a prime number

The function returns a list of integers and strings until a condition described below is met. Suppose this list is called `result`. The following rules are applied:

- If a number is a multiple of both `fizz` and `buzz`, then append `'fizzbuzz'` to `result`.
- If a number is a multiple of `fizz` only, then append `'fizz'` to `result`.
- If a number is a multiple of `buzz` only, then append `'buzz'` to `result`.
- If a number is not a multiple of either `fizz` or `buzz`, then simply append the number to `result`.

Strings and integers are appended to `result` until `'fizz_buzz'` has been appended `max_fizz_buzz` times. You will want to use a while-loop to repeat applying the above rules while a counter is less than `max_fizz_buzz`.

Examples:

Function Call	**Return Value**
`fizz_buzz(3, 2, 7)`	`[1, 'fizz', 3, 'fizz', 5, 'fizz', 'buzz', 'fizz', 9, 'fizz', 11, 'fizz', 13, 'fizzbuzz', 15, 'fizz', 17, 'fizz', 19, 'fizz', 'buzz', 'fizz', 23, 'fizz', 25, 'fizz', 27, 'fizzbuzz', 29, 'fizz', 31, 'fizz', 33, 'fizz', 'buzz', 'fizz', 37, 'fizz', 39, 'fizz', 41, 'fizzbuzz']`
`fizz_buzz(11, 3, 2)`	`[1, 'buzz', 'fizz', 'buzz', 5, 'fizzbuzz', 7, 'buzz', 'fizz', 'buzz', 11, 'fizzbuzz', 13, 'buzz', 'fizz', 'buzz', 17, 'fizzbuzz', 19, 'buzz', 'fizz', 'buzz', 23, 'fizzbuzz', 25, 'buzz', 'fizz', 'buzz', 29, 'fizzbuzz', 31, 'buzz', 'fizz', 'buzz', 35, 'fizzbuzz', 37, 'buzz', 'fizz', 'buzz', 41, 'fizzbuzz', 43, 'buzz', 'fizz', 'buzz', 47, 'fizzbuzz', 49, 'buzz', 'fizz', 'buzz', 53, 'fizzbuzz', 55, 'buzz', 'fizz', 'buzz', 59, 'fizzbuzz', 61, 'buzz', 'fizz', 'buzz', 65, 'fizzbuzz']`
`fizz_buzz(11, 2, 2)`	`[1, 'fizzbuzz', 3, 'fizzbuzz', 5, 'fizzbuzz', 7, 'fizzbuzz', 9, 'fizzbuzz', 11, 'fizzbuzz', 13, 'fizzbuzz', 15, 'fizzbuzz', 17, 'fizzbuzz', 19, 'fizzbuzz', 21, 'fizzbuzz']`
`fizz_buzz(8, 5, 5)`	`[1, 2, 3, 4, 'fizzbuzz', 6, 7, 8, 9, 'fizzbuzz', 11, 12, 13, 14, 'fizzbuzz', 16, 17, 18, 19, 'fizzbuzz', 21, 22, 23, 24, 'fizzbuzz', 26, 27, 28, 29, 'fizzbuzz', 31, 32, 33, 34, 'fizzbuzz', 36, 37, 38, 39, 'fizzbuzz']`
`fizz_buzz(5, 2, 3)`	`[1, 'fizz', 'buzz', 'fizz', 5, 'fizzbuzz', 7, 'fizz', 'buzz', 'fizz', 11, 'fizzbuzz', 13, 'fizz', 'buzz', 'fizz', 17, 'fizzbuzz', 19, 'fizz', 'buzz', 'fizz', 23, 'fizzbuzz', 25, 'fizz', 'buzz', 'fizz', 29, 'fizzbuzz']`
`fizz_buzz(8, 5, 5)`	`[1, 2, 3, 4, 'fizzbuzz', 6, 7, 8, 9, 'fizzbuzz', 11, 12, 13, 14, 'fizzbuzz', 16, 17, 18, 19, 'fizzbuzz', 21, 22, 23, 24, 'fizzbuzz', 26, 27, 28, 29, 'fizzbuzz', 31, 32, 33, 34, 'fizzbuzz', 36, 37, 38, 39, 'fizzbuzz']`

Please note that the examples in the driver class are different from the ones provided in the above table.

Part II: Filthy Rich (20 points)

As a Real Estate Tycoon you are planning to purchase properties in different blocks around the city. However, there is a maximum number of properties you can buy based on your budget. Write a function `mass_purchase()` that takes arguments in the following order:

- `city`: A *list of lists* of integers representing prices of properties. Each integer represents the price of a building in millions of dollars ($M).
- `budget`: An integer that represents the maximum amount in dollars (in $M) you can spend on purchasing bulidings.

The function iterates over the list of lists, visiting the values in this order: `city[0][0]`, `city[0][1]`, `city[0][2]`, ..., `city[1][0]`, `city[1][1]`, `city[1][2]`, ... Note that it is not necessarily the case that every "row" (e.g., `city[i]`) has the same number of properties.

As the function "visits" each integer value, it adds the value to a running total, provided that doing so would not cause the total to exceed the budget. Adding an integer to the total therefore corresponds with purchasing a building. Each time a building is purchased, the indexes of that building in the list are appended to a list (let's call it `result`). For example, suppose that the function decides to purchase the buildings `city[0,2]`, `city[1,0]` and `city[3,7]`. The return value of the function (`result`) would be the list of lists `[[0,2], [1,0], [3,7]]`. **Important:** the buildings that are purchased **must** be added in the order specified in the previous paragraph.

Hint: use nested for-loops, where the outer loop iterates over the list `city` and the inner loop iterates over a sub-list of `city` (e.g., a particular `city[i]`). Part of the code you will need to write is given below:

```
result = []
for i in _________ :
    for j in _________ :
        if _________ :
            result.append([i, j])
```

Examples:

Function Call	Return Value
`mass_purchase([[12], [2, 1, 6, 7, 14, 11], [1, 3, 4], [8, 9, 1], [5, 13, 4]], 46)`	`[[0, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [2, 0], [2, 1]]`
`mass_purchase([[13, 10, 5, 9, 14], [14, 11, 11, 8], [13, 9], [12, 11, 13, 9], [12, 7, 6, 7, 3, 6]], 33)`	`[[0, 0], [0, 1], [0, 2], [4, 4]]`
`mass_purchase([[12, 7, 9, 14, 11], [6, 8, 7, 6, 2, 5], [4, 13], [6, 7, 12, 6, 11, 2], [12, 7, 14, 3, 2, 3], [14, 13, 7, 2, 7, 8], [13, 7]], 42)`	`[[0, 0], [0, 1], [0, 2], [0, 3]]`
`mass_purchase([[2, 11, 9, 7, 9], [14, 4, 2, 4], [7, 14, 6, 12], [7, 5, 13], [3, 3, 1, 9, 9], [12, 3, 2, 2], [11, 7, 14], [9, 7], [14, 5]], 46)`	`[[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 1], [1, 2], [4, 2]]`
`mass_purchase([[1, 12, 9], [12, 6, 7, 6], [6]], 48)`	`[[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2]]`
`mass_purchase([[14, 14, 2], [6], [6, 11, 7, 9, 9], [7, 12, 14, 4, 14], [8, 13]], 41)`	`[[0, 0], [0, 1], [0, 2], [1, 0], [3, 3]]`
`mass_purchase([[4, 7, 7, 1, 7], [8, 11, 4, 1], [10, 1, 9, 14, 3], [9, 12, 5], [9, 10, 13, 6], [14, 1, 6, 14, 4], [13, 12]], 36)`	`[[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 0], [1, 3], [2, 1]]`
`mass_purchase([[12, 11], [3, 8, 8, 11, 8, 7]], 23)`	`[[0, 0], [0, 1]]`
`mass_purchase([[8, 6, 11], [12, 8], [5, 2, 9, 6, 14], [6, 12], [1, 2, 10, 12, 12, 13]], 48)`	`[[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [2, 1], [4, 0]]`
`mass_purchase([[9, 10], [13], [8, 10], [2, 10, 12, 13], [12, 12, 2, 5, 9, 3], [5, 2, 4]], 36)`	`[[0, 0], [0, 1], [1, 0], [3, 0], [4, 2]]`

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to Blackboard and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this assignment.
3. Click on the link for this assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. Submit only that one `.py` file.
5. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the above directions again. We will grade only your last submission.