CS 3035, Fall 2022
In-Lab Exercise 18

Due on: October 26, 2022 (11:59 PM)

List comprehensions are used in Python and Haskell. We discussed Python List Comprehensions in prior classes. Review the syntax for Python and Haskell List Comprehensions. Use List Comprehensions to do the following tasks in the language specified in **bold** in the question. All these tasks can be done using the Python or Haskell Interactive Shell. However, you may do them using a text editor.

For each task, copy-paste the lines of your code and take a screenshot of your output for your submission.

1. In **Python:** Write a list comprehension to take a list of numbers and generate a new list consisting of cubes of the numbers in the input list. Print the output of your list comprehension.

```
>>> list = [1,2,3,4,5]
>>> cube = [x**3 for x in list]
>>> print(cube)
[1, 8, 27, 64, 125]
```

2. In **Python:** Write a list comprehension to apply the following function (fv) to a list comprising items in the form of tuples. Tuples in Python follow the same syntax as we discussed for Haskell. For example, a list comprising two tuples with 3 values each would look like [(1,2,3), (3,4,5)].

   The function fv() calculates the future value of an amount of money (***principal)*** that is earning interest at a given ***rate*** over a given period of ***time.*** Note that this function accepts the interest rate as a floating point number. For example: an interest rate of 5% is written as 0.05.

   Each tuple in the input list should consist of three numbers. Use fv() to calculate the future value for each tuple, assuming that the first number in each tuple is a principal amount (for example, the number of dollars deposited), the second is the interest rate, and the third is the number of periods.

```
def fv(principal, rate, time):
     curr = principal * ((1 + rate) ** time)
    return curr
```

   Print the output of your list comprehension.

```
>>> def fv(principal, rate, time):
...     curr = principal * ((1 + rate)** time)
...     return curr
...
```

```
>>> input = [(500, 0.10, 2), (1000, 0.05, 4), (1500, 0.07, 5)]
>>>
>>> amountCalculator = [fv(principal, rate, time) for principal, rate, time in input]
>>> print (amountCalculator)
[605.0000000000001, 1215.5062500000001, 2103.8275960500005]
>>>
```

3. In **Python:** Write a function abbreviator() that takes an input string and only keeps the upper-case characters from the original string, creating the abbreviation. For example, for an input string "Your Mileage Might Vary", the output of abbreviator() is YMMV. Print the input and output string.

   You must use list comprehension to create a list of upper-case characters. Then, you must convert the list returned by the list comprehension to an output string. One way to do this is to create an empty string and use a loop to append upper-case characters to this string. In python, you can use "+=" to append a character to a string.

   Also, note the following:
   - You can apply list comprehensions to strings just as if they were lists of characters
   - A character is an upper-case letter if and only if it is greater than or equal to upper-case A and less than or equal to upper-case Z.
   - Python's logical AND operator is just the all-lower case word 'and'.

```
>>> def abbreviator(input):
...         abbreviation = [character for character in input if character >='A' and
character <= 'Z']
...     output_string = ""
...     for item in abbreviation:
...             output_string += item
...     return output_string
...
>>> input_string = "Your Mileage Might Vary"
>>> print(abbreviator(input_string))
YMMV
>>>
```

4. **In Haskell:** Write the list comprehension to only output upper-case characters from an input string. You may input your string directly into the list Comprehension without storing it in a variable prior to creating the list comprehension. Note the following:
   - A character is an upper case letter if it lies within the range of letters between upper-case A and upper-case Z.
   - The predicate or condition in your list comprehension can create such a range and use the elem function to find if the input string contains an upper-case element.

```
Prelude>  [c | c <- "Your Mileage Might Vary", c `elem` ['A'..'Z']]
"YMMV"
```

5. **In Haskell:** Write the list comprehension from Task 3 in Haskell. Print the output of your list comprehension.

```
Prelude> [x^3 | x <- [1,2,3,4,5]]
[1,8,27,64,125]
Prelude> [x**3 | x <- [1,2,3,4,5]]
[1.0,8.0,27.0,64.0,125.0]
```