Homework 3

MPCS 51042 – Python Programming

Due: April 30th 2019, 11:59 pm

Initial Setup

Make sure to perform a pull upstream inside your repository. This will grab the distribution code for hw3. The command is the following:

```
$ git pull upstream master
```

Homework Video

Before beginning the assignment, please watch the short video on Panopto:

3.7: Nested Functions

Style Guide

For this homework and all future homework assignments, we will follow the style guide used by the undergraduate Python course. It's located here: `https://classes.cs.uchicago.edu/archive/2018/fall/12100-1/style-guide/index.html`

Requirements for Problems 1-4

Problems 1-4 will have you implement a single function using the functional programming paradigm. The purpose of these problems is to provide you exposure to programming in a different style then normal. Specifically, these problems will expose you to how you would program if you were in a functional programming language. In Problems 1-4, you must adhere to these restrictions:

- You are only allowed to use the `map`, `filter`, or `functools.reduce` functions if you are iterating over an iterable objects (e.g., lists, dictionaries, sets, etc.)
- **You can not use recursion or any looping structure (e.g., for loop, while loop, etc.)**.
- You cannot use any type of comprehension (i.e., list, dictionary, or set comprehension). However, you are allowed to use `list(...)` to convert an iterable object into a list.
- No importing other modules for help with the exception of the `functools` module. However, you can only use the `reduce` function inside `functools` and no other function.
- Specific problems may have additional restrictions so please read the problem.
- You may write helper functions but they must adhere to the restrictions above.

Please ask on Piazza if you are unsure about using certain functions. I will be happy to clarify anything so feel free to ask questions.

Problem 1

Implement the `satisfy` and `satisfy_all` functions:

- `satisfy`: takes in two predicate functions (`func1` and `func2`) and a list of integers (`values`). This function returns a list of tuples where each tuple contains two components. The first component is the integer from `values`, and the second component is `True` if both predicate functions (`func1` and `func2`) return `True` when given the integer; otherwise the second component is `False`.

```
In [1]: import problem1

In [2]: problem1.satisfy(lambda x: x > 10, lambda y: y < 100, [1,20,200])
Out[2]: [(1, False), (20, True), (200, False)]

In [3]: problem1.satisfy(lambda x: x > 10, lambda y: y < 100, [])
Out[3]: []

In [4]: problem1.satisfy(lambda x: x > 10, lambda y: y < 100, [2])
Out[4]: [(2, False)]
```

- `satisfy_all`: takes in a list of predicates (`funcs`) and a list of integers (`values`). This function returns a list of tuples where each tuple contains two components. The first component is an integer from `values`, and the second component is a list of the return values from calling each predicate function with the integer:

```
In [5]: import problem1

In [6]: funcs = [lambda x: x > 10, lambda y: y < 100]

In [7]: problem1.satisfy_all(funcs,[1,10,200])
Out[7]: [(1, [False, True]), (10, [False, True]), (200, [True, False])]

In [8]: problem1.satisfy_all(funcs,[])
Out[8]: []

In [9]: problem1.satisfy_all(funcs,[2])
Out[9]: [(2, [False, True])]

In [10]: problem1.satisfy_all(funcs,[20])
Out[10]: [(20, [True, True])]
```

There is a pytest test file (`hw3/problem1/test_problem1.py`) with additional tests. Make sure you pass those tests.

Problem 2

Implement a function `max_sum` that takes in a list of integer lists (`values`) and returns the total count of all the integers inside the lists within `values`.

```
In [1]: import problem2
```

```
In [2]: problem2.max_sum([[1,2,3],[2],[3,4,5]])
Out[2]: 7

In [3]: problem2.max_sum([])
Out[3]: 0

In [4]: problem2.max_sum([[1],[2],[3]])
Out[4]: 3
```

Restriction:

- You cannot use the `len` function. However, think about reimplementing the `len` function using `functools.reduce`.

There is a pytest test file (`hw3/problem2/test_problem2.py`) with additional tests. Make sure you pass those tests.

Problem 3

Implement a function `duplicate` that takes in a list of integers (`values`) and an integer (`num`). This function returns a list of tuples where each tuple contains an integer from `values` duplicated `num + 1` times.

```
In [1]: import problem3

In [2]: problem3.duplicate([1,2,3],3)
Out[2]: [(1, 1, 1, 1), (2, 2, 2, 2), (3, 3, 3, 3)]

In [3]: problem3.duplicate([],3)
Out[3]: []

In [4]: problem3.duplicate([1],0)
Out[4]: [(1,)]

In [5]: problem3.duplicate([100],1)
Out[5]: [(100, 100)]
```

Restriction:

- You cannot use the repetition operator ($*$) for sequence types. Think about writing a helper function that performs that functionality.

There is a pytest test file (`hw3/problem3/test_problem3.py`) with additional tests. Make sure you pass those tests.

Problem 4

Implement a function `list_range` takes in a range that is represent by an integer tuple (`rangeTup`), and a list of integers lists (`values`). The function returns a list that includes each integer list inside `values` if all their integers are within the range specified by `rangeTup` (inclusive on both ends). You can assume the lower-bound is the first component and the upper-bound is the second component. Assume the lower-bound integer is always less then or equal to the upper-bound integer.

```
In [2]: problem4.list_range((1,3),[[1,3,2],[9,87,1],[2,1]])
Out[2]: [[1, 3, 2], [2, 1]]

In [3]: problem4.list_range((1,3),[])
Out[3]: []

In [4]: problem4.list_range((1,300),[[1,3,2],[9,87,1],[2,1]])
Out[4]: [[1, 3, 2], [9, 87, 1], [2, 1]]
```

There is a pytest test file (`hw3/problem4/test_problem4.py`) with additional tests. Make sure you pass those tests.

Problem 5

Write a function that mimics the Unix `grep` command. The function definition should look like:

```
def grep(pattern, lines, ignore_case=False):
    ...
```

`pattern` should be a string that represents a "pattern" that we want to find within each line. The function checks whether "pattern" is a substring in any of the provided lines. `lines` should be an iterable of strings (this could be a single string, a list of strings, or tuple). Finally, the `ignore_case` argument indicates whether we want to search for the pattern in a case-sensitive or case-insensitive manner. The matching lines should not be returned as a list. Instead, the function should yield the matched lines. If the function has yielded all then matched lines then it returns `None` for every subsequent call.

Note: You must implement grep as a generator function. This means it should generate the next value on demand and not upfront as with lists.

Here is an example of using `grep` based on the description above:

```
In [1]: import problem5

In [2]: lines = ['I went to Poland.',
   ...: ...          'He went to Spain.',
   ...: ...          'She is very happy.']

In [3]: t = problem5.grep('went', lines)

In [4]: for x in range(5):
   ...:     print(next(t))
   ...:
I went to Poland.
He went to Spain.
None
None
None

In [5]: t = problem5.grep('i', lines, ignore_case=True)

In [6]: for x in range(5):
   ...:     print(next(t))
```

```
   ...:
I went to Poland.
He went to Spain.
She is very happy.
None
None
```

Notice in the first example, only two lines matched the pattern. Thus, the third call (and all subsequent calls) to the function object (i.e., `t`) return `None`.

Place your solution in **hw3/problem5/problem5.py**

Problem 6

Rewrite Problem 5 with `grep` not using a generator. The usage and functionality will be the same as Problem 5. In this implementation, you should think about creating the results up front instead of on demand. The sample run should work the same for this version of `grep`.

Hint: Think about using a nested function.

Here is an example of using `grep` based on the description above:

```
In [1]: import problem6

In [2]: lines = ['I went to Poland.',
   ...: ...           'He went to Spain.',
   ...: ...           'She is very happy.']

In [3]: t = problem6.grep('went', lines)

In [4]: for x in range(5):
   ...:     print(t())
   ...:
I went to Poland.
He went to Spain.
None
None
None

In [5]: t = problem6.grep('i', lines, ignore_case=True)

In [6]: for x in range(5):
   ...:     print(t())
   ...:
I went to Poland.
He went to Spain.
She is very happy.
None
None
```

Place your solution in **hw3/problem6/problem6.py**