# CptS355 - Assignment 3 - Spring 2023

## Python Warm-up

**Assigned:** Saturday, March 4, 2023

**Weight:** This assignment will count for 7% of your final grade.

**This assignment is to be your own work. Refer to the course academic integrity statement in the syllabus.**

### Turning in your assignment

All the problem solutions should be placed in a single file named **`HW3.py`**. At the top of the file in a comment, please include your name and the **names of the students with whom you discussed any of the problems in this homework**. This is an individual assignment and the final writing in the submitted file should be *solely yours*. You may NOT copy another student's code or work together on writing code. You may not copy code from the web, or anything else that lets you avoid solving the problems for yourself. **You are not allowed to use ChatGPT or similar tools to create solutions or part of the solutions for the HW problems.**

In addition, you will write unit tests using `unittest` testing framework. You will write your tests in the file **`HW3tests.py`** – the template of this file is available on the HW3 assignment page. You will edit this file and provide additional tests (add at least one test per problem).

To submit your assignment, please upload both files (`HW3.py` and `HW3tests.py`) on the Assignment3 (Python) DROPBOX on Canvas (under Assignments).

Please don't zip your code; directly attach the .py files to the dropbox. You may turn in your assignment up to 3 times. Only the last one submitted will be graded. Implement your code for Python3.

### Grading

The assignment will be marked for good programming style as well as thoroughness of testing and clean and correct execution. **6% of the points will be reserved for the test functions.** Turning in "final" code that produces debugging output is bad form, and points will be deducted if you kept the debugging output in your code. We suggest you the following:

- Near the top of your program write a debug function that can be turned on and off by changing a single variable. For example,

```
debugging = True
def debug(*s):
     if debugging:
        print(*s)
```

- Where you want to produce debugging output use:

```
debug("This is my debugging output",x,y)
```

  instead of `print`.

(How it works: Using * in front of the parameter of a function means that a variable number of arguments can be passed to that parameter. Then using *s as print's argument passes along those arguments to print.)

**Problems:**

## 1(a) aggregate_log – 5%

Assume you keep track of the number of hours you study for each course you are enrolled in daily. You maintain the log of your hours in a Python dictionary as follows:

```
log_input = {'CptS355':{'Mon':3,'Wed':2,'Sat':2},
             'CptS360':{'Mon':3,'Tue':2,'Wed':2,'Fri':10},
             'CptS321':{'Tue':2,'Wed':2,'Thu':3},
             'CptS322':{'Tue':1,'Thu':5,'Sat':2}}
```

The keys of the dictionary are the course numbers and the values are the dictionaries which include the number of hours you studied on a particular day of the week.

Define a function, **aggregate_log** which adds up the number of hours you studied on each day of the week and returns the summed values as a dictionary. Note that the keys in the resulting dictionary should be the abbreviations for the days of the week and the values should be the total number of hours you have studied on that day. **aggregate_log** would return the following for the above dictionary:

```
{'Fri': 10, 'Mon': 6, 'Sat': 4, 'Thu': 8, 'Tue': 5, 'Wed': 6}
```

**Important Notes:**

1. Your function should not change the input dictionary value.
2. You should not hardcode the keys and values (language and course names) in your solution.
3. You are not allowed to import additional Python libraries we haven't covered in class.

## 1(b) combine_dict – 6%

Define a function **combine_dict** which combines two given study logs and returns the merged dictionary. The values of the common keys should be summed in the resulting dictionary. For example:

```
log1 = {'Mon':3,'Wed':2,'Sat':2}
log2 = {'Mon':3,'Tue':2,'Wed':2,'Fri':10}
combine_dict(log1,log2)
```

will return:

```
{'Mon': 6, 'Wed': 4, 'Sat': 2, 'Tue': 2, 'Fri': 10}
```

**Important Notes:**

1. Your function should not change the input dictionary value.
2. You should not hardcode the keys and values (language and course names) in your solution.
3. You are not allowed to import additional Python libraries we haven't covered in class.

## 1(c) merge_logs – 12%

Now assume that you kept the log of number of hours you studied for your courses throughout the semester and stored that data as a list of dictionaries. This list includes a dictionary for each week you recorded your log.

Define a function **merge_logs** which takes a list of course log dictionaries and returns a dictionary which includes the combined logs for each class, i.e., the logs of each class should be merged to a single dictionary. You should use **combine_dict** (from part (b)) in your solution.

```
log_list = [{'CptS355':{'Mon':3,'Wed':2,'Sat':2},'CptS360':{'Mon':3,'Tue':2,'Wed':2,'Fri':10},
            'CptS321':{'Tue':2,'Wed':2,'Thu':3},'CptS322':{'Tue':1,'Thu':5,'Sat':2}},
           {'CptS322':{'Mon':2},'CptS360':{'Thu':2, 'Fri':5},'CptS321':{'Mon':1,'Sat':3}},
           {'CptS355':{'Sun':8},'CptS360':{'Fri':5},'CptS321':{'Mon':4},'CptS322':{'Sat':3}}]
```

```
merge_logs(log_list)
```
will return:

```
{'CptS355': {'Mon': 3, 'Wed': 2, 'Sat': 2, 'Sun': 8},
 'CptS360': {'Mon': 3, 'Tue': 2, 'Wed': 2, 'Fri': 20, 'Thu': 2},
 'CptS321': {'Tue': 2, 'Wed': 2, 'Thu': 3, 'Mon': 5, 'Sat': 3},
 'CptS322': {'Tue': 1, 'Thu': 5, 'Sat': 5, 'Mon': 2}}
```

**Important Notes:**

1. Your function should not change the input dictionary value.
2. You should not hardcode the keys and values (language and course names) in your solution.
3. You are not allowed to import additional Python libraries we haven't covered in class.

## 2(a) most_hours – 15%

Consider the combined log output in problem 1(c). Assume you would like to find the course with the maximum total study time. Write a function "most_hours" that takes the log_input data as input and returns the course having the maximum total hours. For example,

```
log_input = {'CptS355': {'Mon': 3, 'Wed': 2, 'Sat': 2, 'Sun': 8},
             'CptS360': {'Mon': 3, 'Tue': 2, 'Wed': 2, 'Fri': 20, 'Thu': 2},
             'CptS321': {'Tue': 2, 'Wed': 2, 'Thu': 3, 'Mon': 5, 'Sat': 3},
             'CptS322': {'Tue': 1, 'Thu': 5, 'Sat': 5, 'Mon': 2}}
most_hours(log_input)
```
returns
```
('CptS360', 29)
```

**Your function definition should not use loops or recursion but use the Python map, reduce, and/or filter functions**. You may define and call helper (or anonymous) functions, however your helper functions should not use loops or recursion. You will not get any points if your solution (or helper functions) uses a loop. If you are using reduce, make sure to import it from functools.
You are not allowed to use Python libraries we haven't covered in class.

**2(b) filter_log – 15%**

Consider the `log_input` data in problem 1(a). Assume you would like to find the courses that you studies for on a particular day of the week for more than some number of hours. Write a function "`filter_log`" that takes the `log_input` data and returns the courses that has the given day in its log with more than or equal to the required number of hours. For example,

```
log_input = {'CptS355': {'Mon': 3, 'Wed': 2, 'Sat': 2, 'Sun': 8},
             'CptS360': {'Mon': 3, 'Tue': 2, 'Wed': 2, 'Fri': 20, 'Thu': 2},
             'CptS321': {'Tue': 2, 'Wed': 2, 'Thu': 3, 'Mon': 5, 'Sat': 3},
             'CptS322': {'Tue': 1, 'Thu': 5, 'Sat': 5, 'Mon': 2}}

filter_log(self.log_input,"Mon", 3)
```

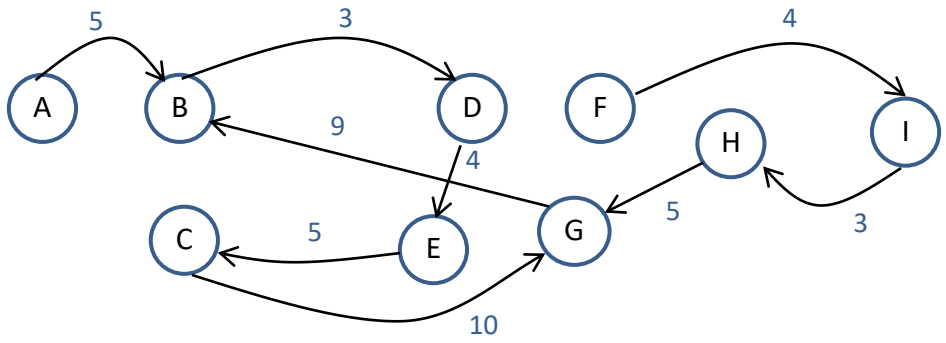
returns

```
['CptS355', 'CptS360', 'CptS321']
```

**Your function definition should not use loops or recursion but use the Python `map`, `reduce`, and/or `filter` functions**. You may define and call helper (or anonymous) functions, however your helper functions should not use loops or recursion. You will not get any points if your solution (or helper functions) uses a loop. If you are using `reduce`, make sure to import it from `functools`.
You are not allowed to use Python libraries we haven't covered in class.

## 3. graph_cycle – 12%

Consider the following directed graph where each node has at most one outgoing edge and each edge has a weight. Assume the graph nodes are assigned unique labels. The edges of this graph can be represented as a Python dictionary where the keys are the starting nodes of the edges and the values are pairs of ending nodes and weights of the edges.




```
graph = {'A':('B',5),'B':('D',3),'C':('G',10),'D':('E',4),'E':('C',5),
      'F':('I',4),'G':('B',9),'H':('G',5),'I':('H',3)}
```

Write a recursive function, `graph_cycle`, which takes a graph dictionary and a starting node as input and returns the sequence of nodes that form a cycle in the graph. It returns the first cycle that exists in the path beginning at node "start". The function returns the list of the cycle node labels where the starting node of the cycle should be included both in the beginning and at the end. If the graph doesn't have any cycles, it returns None.

You may define helper functions to implement your solution. Either your `graph_cycle` function or your helper should be recursive.

For example:

```
graph = {'A':('B',5),'B':('D',3),'C':('G',10),'D':('E',4),'E':('C',5),
         'F':('I',4),'G':('B',9),'H':('G',5),'I':('H',3)}

graph_cycle(self.graph,'F') returns  ['G', 'B', 'D', 'E', 'C', 'G']
graph_cycle(self.graph,'A') returns  ['B', 'D', 'E', 'C', 'G', 'B']
graph_cycle(self.graph,'C') returns  ['C', 'G', 'B', 'D', 'E', 'C']
```

**Important Note:**

- You are not allowed to use Python libraries we haven't covered in class.
- You should not hardcode the keys ( course names) in your solution.

## 4. Iterators

**`filter_iter` – 15%**

Create an iterator whose constructor takes a function (op) and an iterable value (`it`) as argument and applies op on each value of the input iterator. At each call to the `next` function, the iterator will return the result of (op x) where x is the next value from the input sequence.

**Important Note: Your `filter_iter` implementation should calculate the next `(op x)` value as needed.** An implementation that precomputes all values for the given iterator input and dumps all values to a list ahead of time will be worth only 5 points.

For example:

- ```
  it = iter([-1,-2,7,-3,-2,4,5,3,-2,0,3,6,2,-1])
  filter_iter(it, lambda x: x>0)
  ```
  will represent the sequence:
  `7,4,5,3,3,6,2`

- ```
  filter_iter(Numbers(-20), lambda x: x>0 and x%5==1)
  ```
  will represent the infinite sequence:
  `1,6,11,16,21,26,31,36,41,46,51,56,61,66,71,76,81,86,91,96,….`

  The `Numbers` iterator is defined below.

  ```
  class Numbers():
      def __init__(self,init):
          self.current = init
      def __next__(self):
          result = self.current
          self.current += 1
          return result
      def __iter__(self):
          return self
  ```

**5. `merge` – 10%**

Define a function `merge` that takes 2 iterable values "`it1`" and "`it2`" (which are sorted sequences of increasing numbers), and merges the two input sequences. `merge` returns the first `N` elements from the merged sequence. The numbers in the merged sequence needs to be sorted as well. (Note that the iterator will remember its current position among calls to the `merge` function. It retrieves the next element in the sequence in the subsequent calls to `merge`.)

```
it1 = filter_iter(Numbers(-20), lambda x: x>0 and x%2==0)
it2 = filter_iter(Numbers(1), lambda x: x%3==0)
# first call to merge
merge(it1,it2, 10)        <- returns [2,3,4,6,6,8,9,10,12,12]
# second call to merge ; iterator will remember its position;
# note that 14 and 15 will be skipped since previous merge call retrieved them
# from input iterator by calling next, but didn't include them in the output
merge(it1,it2, 5)         <- returns [16,18,18,20,21]
merge(it1,it2, 5)         <- returns [24,26,27,28,30]
```

If one of the input iterators raise `StopIteration` exception (i.e., reaches to the end of the input iterator), then it should stop merging. For example , in the example below, it1 is an iterator with finite number of elements.

```
it1 = iter([2,3,5,7,11,13,17,19])
it2 = filter_iter(Numbers(1), lambda x: x%3==0)
```

```
# first call to merge
merge(it1,it2,5)          <- returns [2,3,3,5,6]
# note that 7 and 9 will be skipped
merge(it1,it2, 2), [11,12])   <- returns [11,12]
#reached to the end of the first sequence, so merge will return the remaining merged
output.
merge(it1,it2, 5), [17,18,19]   <- returns [17,18,19]
#reached to the end of the first sequence, so merge will return [].
merge(it1,it2, 4 []       <- returns []
```

**Assignment rules – 4%**

Make sure that your assignment submission complies with the following. :

- Make sure that all your debugging print statements are removed or disabled. When we run the tests, only the unittest output should be displayed.
- Make sure to include your own tests in `HW3tests.py` file.
- Make sure that you don't import any third part libraries in your code. In the past semesters, some submissions included several unrelated imports. TAs won't be able to run your code with those dependencies.

### Testing your functions (6%)

We will be using the `unittest` Python testing framework in this assignment. See https://docs.python.org/3/library/unittest.html for additional documentation.

The file `HW3SampleTests.zip` file includes 6 .py files where each one includes the unittest tests for a different HW problem. These files import the `HW3` module (`HW3.py` file) which will include your implementations of the given problems.

You should add your own tests in the `HW3tests.py` file – a template of this file is provided. You are expected to add **at least one more test case** for each problem. Make sure to create your own input dictionaries (or change the given dictionaries extensively) for problems 1,2, and 3.

In Python `unittest` framework, each test function has a "`test_`" prefix. To run all tests, execute the following command on the command line.

```
python -m unittest P1_HW3tests.py
```

You can run tests with more detail (higher verbosity) by passing in the -v flag:

```
python -m unittest -v P1_HW3tests.py
```

If you don't add new test cases you will be deduced at least 6% in this homework.