# COSC4315/COSC6345: Programming Languages
# Functional Programming: Set Operations

## 1 Introduction

You will write a program to compute set operations on files with words and numbers. This is a primitive storage mechanism used everywehre, including search engines and word processors. The input will be two files (bags) and one set operation (difference, union, intersection). Let the two sets of unique words in each file be $A$ and $B$. The goal is to compute $A \cup B$, $A \cap B$, $A - B$.

Since this homework requires functional programming you should not worry about memory efficiency and time complexity (of course, avoid exponential time or too many quadratic functions).

## 2 Input

The input are text files, with at most 1000 different words or numbers, separated by symbols (comma, period, parentheses, operators, etc). Repeated spaces or symbols should be considered as one space. Skip symbols (i.e. anything that is not letter or digit, like common separators and arithmetic operators). Input words will be simple strings from regular expressions, English words, either from a dictionary (including abbrevations), or names (city, person, acronyms). You can assume that words will be at most 20 characters long and numbers 20 digits. Assumption: The input files will fit in main memory.

## 3 Program and input specification

The main program should be called "setops" (exactly like this, in lower case, avoid other names). Notice the input files will not be the same when your program is called.

Call syntax:

```
python3 setops.py "set1=[filename];set2=[filename];operation=[difference|union|intersection]"
```

Call examples

```
# spell checker
python3 setops.py "set1=a.txt;set2=b.txt;operation=difference"

# common words
python3 setops.py "set1=a.txt;set2=b.txt;operation=intersection"
```

Review: A word is regular expression of letters, starting in letter combined with digits (like variable names or product names), where letters can be lower or upper case. Names and common words are all letters. Similary, a number is a regular expression of digits, perhaps having a decimal part. Therefore, strings mixing letters+numbers should be treated as separate words. Notice that words can be followed by punctuation signs and they can be capitalized (uppercase first letter).

The output should be sent to the output file "result.txt" *exactly* in the format below with one word per line. Avoid changing the output format, adding spaces/symbols or adding any information messages since this will prevent automated testing. Any important information message should go to the screen.

Output example with union of two files with one word in each file:

```
city
house
```

Intersection would be an empty file (display "empty set").

# 4 Requirements

- Programming languages:

  Python.

  JavaScript.

- Functional programming is the main requirement, as detailed below. Basically, this means all functions should be recursive, there should be no mutation and all computations will happen via lists. You should use a combination of lambda expressions (for short computations) and recursive functions (for recursive algorithms).

- Functional expressions: Your program should internally use lambda expressions to convert keywords, instead of built-in lower and upper case functions. All your search and sort functions should be recursive. Your program should use functional constructs (map, reduce, combine) where possible.

- Lists, as provided by the language, are required to store a set of words: input, temporary and output. You cannot existing Python libraries with advanced arrays, dictionary or hash tables data structures. Therefore, you cannot their associated functions either. Why? They defeat a functional programming approach.

  Lists need to be passed as function argument and/or function return value, but cannot be a global variable. The program must be prepared to handle zeroes or simple errors like spaces or missing numbers extra credit for developing both forward and backward recursion starting from either side of the list.

- Words should be recursively converted to lower case. We will treat words as generic regular expressions.

- Output lists should be sets, sorted with standard string comparisons. Treat numbers as strings.

- Notice input files are not necessarily sets: words can be repeated (i.e. they are bags). However, the output must be a set, without duplicates elements.

- Recursive functions are required to process all lists. It is unacceptable to have loops (while/for) to process the list(s). While loops are acceptable only to read the input file into a list, but recursive functions are encouraged.

- You cannot use text or language processing libraries to parse input files or construct lists, especially language processing libraries. You will get zero if you do so.

- Search and sort algorithms: These lists should have recursive sort and search functions, without mutation. Notice Languages like Python and JavaScript follow C semantics in function parameters: arrays are passed by reference. The input array cannot be mutated (changed). It is acceptable to have an $\Theta(n^2)$ sort algorithm, but $\Theta(n \log(n))$ is encouraged. Specify which algorithm you are using. Search can be sequential or binary, with binary being preferred. Keep in mind a recursive sort algorithm with deep recursion will use significant RAM.

- Correctness is the most important requirement: TEST your program well. Your program should not crash or produce exceptions.

- Create a README file explaining how to run your program and explaining any issues or incomplete features.

- Your program will be tested for originality, comparing it to other students source code, source code students from past semesters and top source code sites from the Internet (e.g. github, stackoverflow). You must disclose any source code that was not written by you. Keep in mind several students may get source code from the same site, especially if it comes as a top hit in Google.

- Your program should display a small "help" if not input parameters are provided and must write error messages to the screen. Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, blank lines and so on.

- Unix system (UH Linux server). You can develop and test your code on any computer or operating system (e.g. Windows, Mac), BUT your program will be tested only on our course server. Therefore, test it carefully on the UH server before the deadline.

- Test cases:

  Your program will be tested with 10 test cases, going from easy to difficult. You can assume 70% of test cases will be clean, valid input files. The remaining 30% will test correctness and efficiency with harder input.

- Grading:

  A program not submitted by the deadline is zero points (any exception must be discussed with the instructor at least 2 days before the deadline). A non-working program is worth 10 points (i.e. not producing ouput for the easiest test case). A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 60 or higher. In general, correctness is more important than speed.