Handin 4 B2

### Part 1

This week, we are going to create a program that compares the words in American English and British English. Specifically, we provide the lists of words that begins with a to e (in both upper and lower cases) in separate files, british-english\_a-e and american\_english-a-e. You will calculate how many entries are different between the two files.

The way to do the task is to put the words from the two files into two separate containers, and run through a loop to check if each word in one file is present in the other. There are a lot of ways to implement this idea, and we will try three of them, each under a different representation of the data.

In addition to the main data files british-english\_a-e and american-english\_a-e,  we also provide two small test files so you can test whether your methods work before actually running on the large files. These files are called british-english-test and american-english-test.  The files can be downloaded at: [british-english\_a-e](https://absalon.ku.dk/files/6725666?wrap=1), [american-english\_a-e](https://absalon.ku.dk/files/6725665?wrap=1), [british-english-test](https://absalon.ku.dk/courses/59965/files/6719679?wrap=1), and [american-english-test](https://absalon.ku.dk/courses/59965/files/6719680?wrap=1" \t "_blank).

1. Create a file called handin4.py. Inside this file, create a function called wordfile\_to\_list, which takes a single argument called filename. This function should read the file, and return a list of words (as strings). You can assume that each line in the file only contains a single word. Please remember to the remove newlines at the end of each line.

Create a file called handin4\_test.py. Inside this file, call your wordfile\_to\_list function on the british-english\_a-e file and save the result in a variable called wordlist\_british. Do the same for the american word list and save it in a variable called wordlist\_american.

1. We will now write the first comparison function. Add a function to the handin4 module called wordfile\_differences\_list\_search, which takes two filenames as arguments, and calls wordfile\_to\_list to create a list for each of these files. The function should contain a loop that for each word in the first list looks through the second list to see if there is a match. It should return a list of words that are in the first file but not in the second. Matches should be case-sensitive, so e.g. "Gnu" and "gnu" are not considered identical.

In the file called handin4\_test.py, call the wordfile\_differences\_list\_search on the input files (british-english\_a-e as file1 and american-english\_a-e as file2), and save the result in a variable called differences\_list\_search.

Python has a built-in functionality for measuring execution times within a programme using the time module, which can be used like this:

import time   
start\_time = time.time()   
# write code you want to measure execution time for here   
time\_spent = time.time() - start\_time

Use this technique to measure how long it takes to call the wordfile\_differences\_list\_search function, and save the result in a variable called time\_spent\_list\_search.

1. Next, we will test the speed of lookups in a Python dictionary. First, we need functionality to read a file into a dictionary instead of a list. Create a function called wordfile\_to\_dict in the handin4 module. This function should be identical to wordfile\_to\_list, but it should save the results as keys in a dictionary rather than in a list (you can choose whatever you like for the values - for instance None).

Add a function to the handin4 module called wordfile\_differences\_dict\_search, which takes two filenames as arguments, and calls wordfile\_to\_list on the first file and wordfile\_to\_dict on the second file. The function should contain a loop that for each word in the list looks in the dictionary to see if there is a match. It should return a list of words that are in the first file but not in the second file.

The test code in handin4\_test.py should call the wordfile\_differences\_dict\_search on the input files. Save the result in a variable called differences\_dict\_search and use the time module to measure the time it takes to do the calculation, saving it in a variable called time\_spent\_dict\_search.

1. Finally, we will test the speed of directly applying Python set operations. First, we need functionality to read a file into a set. Create a function called wordfile\_to\_set in the handin4 module. This function should be identical to wordfile\_to\_list, but it should save the results in a set instead .

Add a function to the handin4 module called wordfile\_differences\_set\_search, which takes two filenames as arguments, calls wordfile\_to\_set to get the word sets, uses set operators or methods to get the set of words in british-english\_a-e but not in american-english\_a-e, and finally converts them to a list using the type conversion function list(). Similarly, write test codes in handin4\_test.py but now call the wordfile\_differences\_set\_search on the input files, and save the result and time it takes in variables differences\_set\_search and time\_spent\_set\_search, respectively.

You are very welcome to print the time cost and result out in handin4\_test.py so you can check your results, although this output will not be used by the code-checker (we check the variables directly).

### Part 2: Project

We will continue with processing the Land\_and\_Ocean\_summary.txt file.

1. Make a new file called handin4\_project.py, start with the read\_data3 function for Hand-in3, and rename it read\_data4 (feel free to copy & paste).
2. Create a new local dictionary variable name decade\_dict.
3. We need to expand the function to iterate over the year keys in the data from the file and save the dictionary values occurring on the first year of a decade (i.e., for 1850, 1860, 1870, ...) to this new dictionary in the same format as the original (i.e., years as keys, and a list of floats as values).
4. Just as in Handin3, year\_range should be optional; if not specified, values for all years should be returned. Additionally, you will need a third argument decade , which will output the decade's first year if it is set to true.
5. Create a new file called  handin4\_project\_test.py, to test your code in the following way:

decade\_anomaly\_data=handin4\_project.read\_data4(filename="Land\_and\_Ocean\_summary.txt", year\_range=(1970,2000), decade=True)  
​  
value=decade\_anomaly\_data[1970][1]

1. You should test your code by calling the decade\_anomaly\_dat function and return the result (in  handin4\_project\_test.py,)

**Hints**:

* The relevant year values are those that have zero remainders when divided by 10. Remember the Modulo Operator. It **returns the remainder of dividing the left-hand operand by the right-hand operand**.
* Additionally, you will have to use a conditional statement that makes conditions for when you have a specific time period you want to output and if you want the decade years.