

BAN401
“Applied Programming and Data Analysis for Business”

FINAL GROUP-BASED PROJECT

SUBMISSION DEADLINE:

DATE: 15 November 2019

TIME: 14.00

I. REPORT PREPARATION:

You should submit TWO separate files:

File 1: One Report file (only PDF format) that includes the following:

1.1 Problems 1-3 (Python - part). For each problem:

- a. Python code
 - Copy your code and past it into you report
 - Keep your code formatting when pasting it into your report (for the best readability)
- b. Explanation of your solution
 - Describe the idea of your designed solution
 - How your code solves the problem
- c. Screenshot of the results from the PyCharm
- d. Required:
 - Comment in your code explaining core moments. For example:

```
s = [5, 35, 26, 44, 4, 1] # list s is created
```
 - Code should be implemented in Python 3 (i.e., not in Python 2)

1.2 Problem 4-5 (R - part). For each problem:

- a. R code
 - Copy your code and past it into you report
 - Keep your code formatting when pasting it into your report (for the best readability)
- b. Explanation of your solution
 - Describe the idea of your designed solution
 - How your code solves the problem
- c. Screenshots of the results from the RStudio
- d. Required:
 - Comment in your code explaining core moments. For example:

```
y <- c(1:10) # create a data vector with elements 1-10
```

1.3. Problem 6 (SQL – part)

- a. SQL code
 - *Copy your code and past it into you report*
 - *Keep your code formatting when pasting it into your report (for the best readability)*
- b. Screenshot of the results

1.4. Problem 7: performed analysis

GENERAL REQUIREMENTS:

- The report should be typed (not handwritten)
- The report should be written in English
- It should be possible to copy text from your report. This does not apply to figures (if any) and screenshots.

File 2: One compressed file (ZIP or RAR format) including the following:

2.1 Problems 1-3 (Python):

- a. 4 code files (.py format) - for problems #1.1, #1.2, #2, and #3, respectively
 - *Names of the .py files should include problem numbers (in the following format):
problem_1.1.py; problem_1.2.py ; problem_2.py; problem_3.py*

2.2 Problems 4-5 (R):

- a. 2 code files (.r format) - for problems # 4 and # 5, respectively
 - *Names of the .r files should include problem numbers (in the following format):
problem_4.r; problem_5.r*

2.3 Problem 6 (SQL):

- a. The .txt file with the SQL query code
 - *Name of the .txt file should include the problem number:
problem_6.txt*

II. SUBMISSION PROCEDURE:

You must submit your final group-based project [via WISEFlow](#).

Please, remember, that you must submit two files prepared according to “I. REPORT PREPARATION” procedure (as described above):

File 1: One Report file (only PDF format)

File 2: One compressed file (ZIP or RAR format)

NOTES

1. Code snippets (in MS Word): format-preserving

To display code snippets in MS Word preserving format and syntax highlighting, the following online tools can be helpful:

For Python 3 and SQL: <http://hilite.me/>

For R: https://emn178.github.io/online-tools/syntax_highlight.html

On the final step, please adjust “Font Size” (in MS Word) of the pasted code snippets (for the best readability).

2. ETHICS & DATA PROTECTION:

- It is **forbidden** to discuss solutions with anyone outside your group.
- It is **NOT allowed** to distribute/post any parts of the given group-based project in the Internet (or any other way) in any format during and after the exam period.
- YOUR SUBMISSION WILL BE CHECKED FOR PLAGIARISM!

PROBLEM 1

Problem 1.1

Write a Python code to solve the following problem:

1. Create a dictionary of numbers and their associated words in English following Table 1:

Table 1 Numbers

0	10	20	30	40	50	60	70	80	90	100
zero	ten	twenty	thirty	forty	fifty	sixty	seventy	eighty	ninety	hundred

2. Use `input ()`-method to ask user for any number from Table 1.
3. Write a code that prints the word in English for the entered number. For example, if the input is 40, the code should print *"The English word for 40 is forty."* If the user gives any other inputs (including numbers that are not presented in the dictionary), the code should print *"Sorry, the input was not valid"* and ask again for another input until the user enters a number from Table 1

Apply your code to any user-input. Provide the screenshot of your results from the PyCharm.

Problem 1.2

Write Python code that reads a sequence of words, where each word can be separated from another word by any number of white spaces. Use `input ()`-method to ask user for a sequence of words.

Your code should remove all the repeated words, sort the resulting sequence of words alphabetically, and then print it. All printed words should be separated from each other by just one white space. Words that start with uppercase should be printed first (sorted alphabetically). Words that start with uppercase and words that start with lowercase are to be considered as different words.

Example output is as follows:

Enter your sequence of words:

Write Python Code a sequence word each word can be Separated another word Number of white spaces a write
Code Number Python Separated Write a another be can each of sequence spaces white word write

Requirements:

- You are not allowed to use/import any modules.
- Code must be implemented in Python 3
- Apply your code to any sequence of words. Provide the screenshot of your results from the PyCharm.

PROBLEM 2

You learned how to create functions using **def** keyword in Python. Write a Python code to solve the following problem:

Write a Python function `get_the_chain(numbers)` that finds the longest chain of numbers in an unsorted list of numbers:

- Argument `numbers` is a list of integers
- Argument `numbers` can be of any length

For example, `numbers = [45, 56, 8, 92, 43, 5, 6, 44]` is the list of eight numbers, where the longest chain is `[43, 44, 45]`.

If there are multiple chains of the same length in `numbers`, the function should return the chain that starts with the highest number. For example, `[2, 3, 4]` and `[6, 7, 8]` are two chains of the same length equal to three, but the chain `[6, 7, 8]` starts with the higher number. Thus, the chain of numbers `[6, 7, 8]` should be returned by the function.

Your code should display the initial `numbers`-list and the resulting list (the longest chain of numbers without duplicates) returned by the function `get_the_chain(numbers)`

Apply your function to the following list (and display the result):

```
numbers = [0, 7, 4, 8, 1, 3, 8, 10, 11, 2, 5, 12, 9]
```

Requirements:

- You must create your own mechanism to solve the problem; solving the problem you are allowed to use the following built-in functions:
 - `len()`
 - `print()`
 - `list()`
 Any other built-in functions are not allowed¹.
- You are not allowed to use/import any modules.
- You are not allowed to use any sorting mechanisms integrated in Python (like method `.sort()`)
- Code must be implemented in Python 3.
- Provide the screenshot of your results from the PyCharm.

¹ See "Built-in functions"-table: <https://docs.python.org/3/library/functions.html>

PROBLEM 3

Consider the following table with information about fictitious NHH Master's students²:

Table 2 "NHH students"

<i>First Name</i>	<i>Last Name</i>	<i>Student ID</i>	<i>GPA</i>	<i>Major Profile</i>	<i>NHHS groups</i>
Mike	Wheeler	19710	3.5	FIE	it.gruppen
Nancy	Wheeler	19670	3.6	ENE	K7 Bulletin, NHHS Opptur, NHHS Energi
Steve	Harrington	19660	2.4	STR	
Mike	Wazowski	18119	2.9	BAN	
Jeffrey	Lebowski	69420	4.2	BLZ	NHHI Bowling, NHHI Vinum
Ivan	Belik	12345	1.8	BAN	it.gruppen, NHHS Consulting
Sterling	Archer	11007	2.7	MBM	NHHI Lacrosse

In Table 2 above:

- Column "First Name" lists first names of the students.
- Column "Last Name" lists last names of the students.
- Column "GPA" specifies current GPA for each student.
- Column "Major Profile" contains an abbreviation of each student's major profile in NHH's Master's program.
- Column "NHHS groups" provides information about each student's membership in NHH's interest groups, sports clubs, and NHHS' subcommittees.

PROBLEM DESCRIPTION:

- (1) Create an object in .py file (Python 3) that would contain all of the data in the Table 2.
- You may use any of the data structures (or any combination thereof) to which you have been introduced in this course.
 - When deciding on which data structure(s) to use, consider
 - efficiency of different ways of storing data in Python;
 - the types of data manipulations that will be performed by the end-user of the program you will write for this problem (see (2) for details).

NOTE: you do **not** need to store the *names* of the columns (e.g. "First Name", "GPA" etc.) explicitly in your Python object.

² All names and characters portrayed in this problem are fictitious. No identification with actual persons (living or deceased) and groups is intended or should be inferred.

(2) In the same .py file, write a simple “search engine” that would allow user to search for a specific student and display information about that student. Your “search engine” must have the following functionality:

- Ask user to specify which student she/he is looking for. Your program should be able to find the relevant student if the search query was entered as one of the following:
 - First name
 - Last name
 - Both first and last name, in any order, as part of the same search query
 - Student ID
- When given a query that does not match any students in Table 2, the program should display the following message to the user: "No matches found."
- When a query matches one and only one student in Table 2, your program should display message "One match found." and then print out that student's first and last name and student ID followed by that student's GPA, Major profile, and any groups of which that student is a member (in the format seen in Figure 1). If no information is stored about a student's group membership, your program should display "N/A" (in the "NHHS group membership" field) when showing the results for such student.

```

-----
Retrieving data for student Jeffrey Lebowski (student ID 69420).
- GPA: 4.2
- Major: BLZ
- NHHS group membership:
  - NHHI Bowling
  - NHHI Vinum

```

Figure 1 Output format

- When a query matches several entries (e.g., user searched for a first *or* last name that is present in the Table 2 more than once), your program should:
 - step i.* Display message "Several results matched your query:" followed by a numbered list of matches (from 1 to n , where n is the number of results that matched the query) with the first name, last name, and ID number for each matched student (in the format seen in the screenshot below (Figure 2)).
 - step ii.* Ask user to *either* specify the number of the matching search result (from the numbered list of matches in *step i*) for which she/he wants to retrieve the information *or* enter the word **all** to indicate that she/he would like results to be displayed at once for all matches. Specifically, user should be given the following opportunities:
 - input a new number from the resulted numbered list of multiple matches. This should result in displaying information about the student (in the format presented in Figure 1) *OR*

- input a single request to retrieve and display information (in the format specified in Figure 1) about each student that was displayed in *step i*

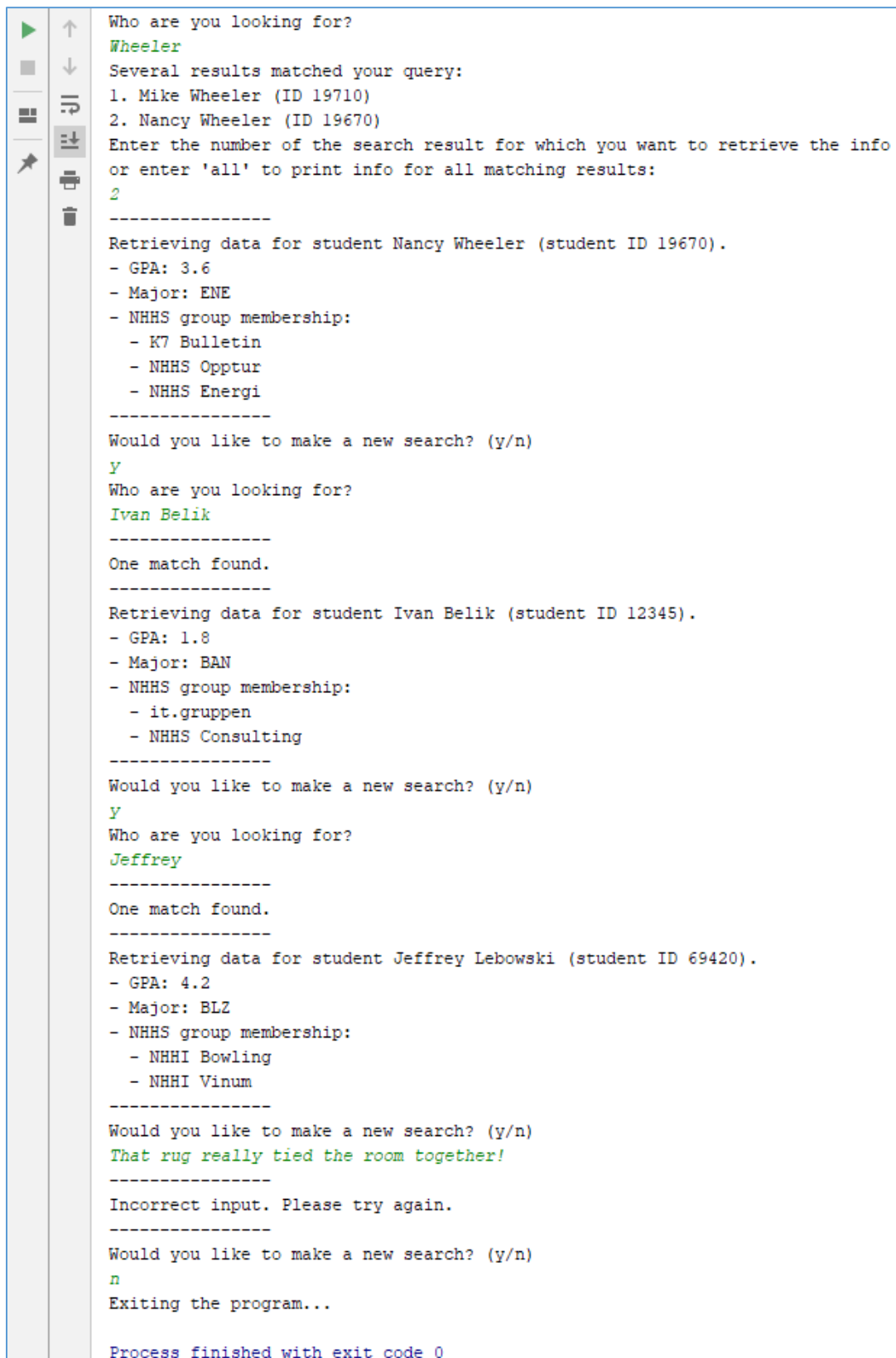
step iii. Display message "Incorrect input. Please try again" when during *step ii* user enters a number that does not correspond to any of the items in the numbered list of matches presented to her/him in *step i*.

User should then be redirected to *step ii*. This should continue as long as user does not correctly specify the number of the match from *step i* or asks for all matching results to be displayed.

NOTE: your program **does not** need to handle situations in which user's input in *step ii* is neither a number nor a request to display all results. E.g., if your Python script returns a `ValueError` message during *step ii* after receiving user input that is not a number or a request to display all results, it will **not** be considered a mistake.

- After the search yielded one or more matches and information about matching student(s) was retrieved *or* after the program reported that no matches were found for the search query, your script should ask user if she/he wants to perform a new search or exit the program. Depending on user's response, the program should then either ask for a new search query or finish. If user provided a response that the program cannot interpret as a request to perform either of these two options (i.e. perform a new search or stop), your script should display the message "Incorrect input. Please try again." and ask user to repeat her/his response.

(3) Run your script and retrieve information for *Nancy Wheeler*, *Ivan Belik*, and *Jeffrey Lebowski* by performing the search queries and navigating your search engine's prompts exactly as shown in Figure 2.



```

Who are you looking for?
Wheeler
Several results matched your query:
1. Mike Wheeler (ID 19710)
2. Nancy Wheeler (ID 19670)
Enter the number of the search result for which you want to retrieve the info
or enter 'all' to print info for all matching results:
2
-----
Retrieving data for student Nancy Wheeler (student ID 19670).
- GPA: 3.6
- Major: ENE
- NHHS group membership:
  - K7 Bulletin
  - NHHS Opptur
  - NHHS Energi
-----
Would you like to make a new search? (y/n)
y
Who are you looking for?
Ivan Belik
-----
One match found.
-----
Retrieving data for student Ivan Belik (student ID 12345).
- GPA: 1.8
- Major: BAN
- NHHS group membership:
  - it.gruppen
  - NHHS Consulting
-----
Would you like to make a new search? (y/n)
y
Who are you looking for?
Jeffrey
-----
One match found.
-----
Retrieving data for student Jeffrey Lebowski (student ID 69420).
- GPA: 4.2
- Major: BLZ
- NHHS group membership:
  - NHHI Bowling
  - NHHI Vinum
-----
Would you like to make a new search? (y/n)
That rug really tied the room together!
-----
Incorrect input. Please try again.
-----
Would you like to make a new search? (y/n)
n
Exiting the program...

Process finished with exit code 0

```

Figure 2. Output

(4) Finally, run your script to process the following requests (i.e. user inputs) in the same order as shown below:

- Mike
- all
- y
- 11007
- y
- Copernicus
- N

(5) Provide screenshots of your results from parts (3) and (4) (of the “PROBLEM DESCRIPTION” in the given Problem 3) in the PDF report file.

REQUIREMENTS:

1. You are not allowed to use/import any modules.
2. Code must be implemented in Python 3 (i.e., not in Python 2).

PROBLEM 4

We have a deck of an unlimited number of cards, where each card has only one of the following values stamped on it:

“1 point”, “2 points”, “5 points”, “10 points”, “20 points”, “50 points”, “1 megapoint”, or “2 megapoints”.

Notations:

1 megapoint = 100 points

2 megapoints = 200 points

Based on the given specification, a possible scenario to buy one “2 megapoints”-card is to pay with the following set of cards:

- One “1 megapoint”-card
- Two “20 points”-cards
- One “50 points”-card
- Three “1 point”-cards
- One “2 points”-card
- One “5 points”-card

How many possible scenarios exist to buy one “2 megapoints”-card?

Write an R code (one .r script) to solve this problem. Your code should display a result in the RStudio console.

Requirements:

- Solving this problem, you are not allowed to buy one “2 megapoints”-card paying with a “2 megapoints”-card
- Solving this problem, you are not allowed to have any packages loaded into your R script (no `library()` allowed)
- Provide the screenshot of your results (from the RStudio console)

PROBLEM 5

I. DATA DESCRIPTION:

Consider two tables describing publicly traded companies in the pharmaceutical/biomedical healthcare sector³:

- *Table 3 “Share prices dataset”:*

Company name	Share price	Founded	Number of employees
Bayer	74.08	1952	116998
Biogen	220.47	2007	7800
GlaxoSmithKline	206.47	1999	96851
Hoffmann-La Roche	272.97	1989	88509
Novartis	89.77	1996	125161
Orion	37.06	2006	3154
Pfizer	35.51	1849	92400
Pharma Mar SA	2.2	1989	599
Rigel	1.73	1996	158
Takeda	33.78	1781	49578

Table 3 contains information about companies in terms of four variables:

- Column “Company name” contains names of the companies.
- Column “Share price” lists companies’ share prices.
- Column “Founded” specifies the year each company was established.
- Column “Number of employees” provides an estimate of the total number of workers employed by a company.

- *Table 4 “Patent Application Registry”:*

Patent Application ID	Status	Applicant
19931853	Ceased	PFIZER HEALTH AB
19941619	Refused	PFIZER HEALTH AB
20002977	Granted	PFIZER HEALTH AB
20006323	Granted	ADVANCED MEDICINE INC
20010863	Granted	NOVARTIS AG (CH)
20012424	Granted	NOVARTIS AG (CH)
20012853	Granted	NOVARTIS AG (CH)
20013463	Granted	BAYER HEALTHCARE LLC (US)
20014524	Granted	SMITHKLINE
20016071	Granted	SANOFI-AVENTIS DEUTSCHLAND GMBH
20021920	Granted	ORION CORP (FI)
20026197	Granted	F HOFFMANN LA ROCHE AG
20043632	Granted	RIGEL PHARMACEUTICALS INC (US)
20055465	Granted	F HOFFMANN LA ROCHE AG
20060992	Pending	RIGEL PHARMACEUTICALS INC (US)
20062515	Granted	TAKEDA PHARMACEUTICAL COMPANY LIMITED (JP)
20063739	Pending	NOVARTIS AG (CH)
20071584	Granted	BAYER HEALTHCARE LLC (US)
20076687	Pending	ORION CORP (FI)
20084114	Granted	ORION CORP (FI)

³ All names and data portrayed in this problem are fictitious. No identification with actual names and data is intended or should be inferred.

Table 4 draws on data from a patent registry. It contains information about patent applications along with the names of the applicant companies:

- Column “Patent Application ID” holds the numerical identifiers assigned to each patent application.
- Column “Status” indicates status of a patent application: “*Granted*”, “*Refused*”, “*Ceased*”, or “*Pending*”.
- Column “Applicant” specifies the name of the company that applied for a given patent.

Your goal in this problem is to use data from Table 3 and Table 4 to create Table 5, which would contain all the columns from Table 3 and an additional column (“Total number of patents”) with the *total* number of patents *granted* (i.e. patents with value “*Granted*” in column “Status”) to each company.

Note that most, but not all, companies are present in both Table 3 and Table 4. Some companies that filed a patent application listed in Table 4 are not mentioned in Table 3. As Table 3 provides most of the data in which you are interested, patent applications by companies not listed in Table 3 should not be included in Table 5.

For example, *ADVANCED MEDICINE INC* has filed patent application 20006323 according to Table 4, but is not present in Table 3 and should therefore not be a part of Table 5.

Note also that each row in Table 4 lists a separate patent application and that each application has a status associated with it. Some companies from Table 3 have multiple patent applications listed, while others have none. You are only interested in active patents, which means that patent applications with any status other than “*Granted*” should not count towards the grand total of patents in possession of a given company in Table 5.

For example, *ORION CORP (FI)* has three patent applications listed in Table 4, but the patents have only been granted for applications 20084114 and 20021920, so in Table 5 there should be value “2” recorded in the column “Total number of patents” for the row associated with *Orion*.

Biogen, on the other hand, does not have any patent applications according to Table 4, and so value “0” should be stored as its number of patent applications in Table 5.

Finally, note that Table 3 and Table 4 have different conventions for how the names of the companies are stored.

E.g., a company listed as *GlaxoSmithKline* in Table 3 can also be found in the patent registry (Table 4), but under the name *SMITHKLINE*.

You will need to write R code that would perform the process of matching entries that relate to the same company, but have its name stored differently in Table 3 and Table 4. The most efficient way to do this is to use fuzzy matching.

II. FUZZY MATCHING

Fuzzy matching, or *approximate string matching*, is the technique of finding imperfect (i.e. not exact) matches between pairs of strings. Fuzzy matching is widely used for automated spell checking, computer-assisted translation, spam filtering, and analysis of unstructured data.

Fuzzy matching calculates *edit distance* between two given strings, allowing us to identify strings that approximately match the pattern we specified (i.e. are “not too far” from our benchmark pattern).

Edit distance refers to the number of *transformations* required to turn a sequence of characters in string B into an identical match with string A. Transformations are one-character changes to a string. There are three types of transformations: *deletion*, *insertion*, and *substitution*.

For example, edit distance between strings “they’re” and “their” is three, since that is the minimum number of transformations required to make the strings a perfect match:

1. their → theyr (substitution of “i” for “y”)
2. theyr → they’r (insertion of ’)
3. they’r → they’re (insertion of “e” at the end)

Within the general technique of fuzzy matching, there exist many specific algorithms whose accuracy varies depending on the context in which they are applied. To implement fuzzy matching in this problem, you should use built-in function `agrep()`, which is a part of the base R installation (please, do not load any packages with the `library()` function when working on this problem).

Note: you should not create a mechanism for finding a good match or develop its algorithm. To perform fuzzy matching between company names in Table 3 and Table 4, you simply need to employ the function `agrep()`.

The official R documentation for the function `agrep()` can be found here:

<https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/agrep>

Hint 1.

In your R code, function `agrep()` *must* be given the following two arguments:

1. A **pattern** string that will be used as the “benchmark” to which the comparisons are made.
2. Character vector **X** in which the function will try to find approximate matches to the **pattern**.

Additionally, `agrep()` can take argument **max.distance**, which allows user to specify maximum allowed *distance* between the *pattern* and a *string* in vector X. Distance here is expressed either as an integer (total number of edits allowed) or as the maximum fraction of the length of the *pattern* that is allowed to be modified.

The default value for **max.distance** is 0.1. In order to solve this problem, **you do not need to modify this value** or even pass **max.distance** argument to `agrep()`, as its default value is already at the level that will allow 100% accuracy given the data in Table 3 and Table 4.

Finally, by passing the argument **ignore.case**, user may indicate whether pattern matching should be case sensitive or not. **ignore.case=TRUE** forces R to ignore case. Passing **ignore.case=FALSE** will make string comparisons case sensitive. If this argument is not passed when calling the function **agrep()**, it will default to **FALSE**.

Note that there are a few other arguments that **agrep()** can take (**costs**, **value**, **fixed**, and **useBytes**), but understanding and/or using any arguments not discussed above is not necessary for solving this problem. Also, remember that you should not pass argument **max.distance** to the function.

Hint 2.

Note that in the following case **agrep()** will return a vector with the value 1:

```
> s1 <- "Bayer"
> s2 <- "Bayer Healthcare LLC"
> agrep(s1, s2)
[1] 1
```

When calling **agrep()**, we passed **s1** as the **pattern** argument and **s2** as the **X** argument. Since **s2** is a character vector of length 1 and its first (and only) element contains the pattern specified in **s1**, R found one match and returned a vector containing its index in vector **X**.

Remember that in terms of R, when we assign a single value (e.g., "Bayer Healthcare LLC") to a variable (e.g., **s2**), we create a vector of length 1.

Note that this is a perfect match: even though the string in **s2** contains more characters than **s1**, our pattern (**s1**) is fully contained in **s2**.

The opposite does not hold. Note that if we pass **s2** as the pattern instead and look for matches in **s1**, R will not find any matches, as string "Bayer Healthcare LLC" is not contained in **s1**:

```
> s1 <- "Bayer"
> s2 <- "Bayer Healthcare LLC"
> agrep(s2, s1)
integer(0)
```

Vector **X** can contain multiple elements:

```
> s3 <- "Bayer"
> s4 <- c("Biogen", "Bayer Healthcare LLC", "BAYER", "Bayer")
> agrep(s3, s4)
[1] 2 4
```

In this case, R identifies only two matches with our pattern (**s3**) and outputs a vector with their indices (in **s4**). "Bayer Healthcare LLC" (2nd element of **s4**) and "Bayer" (4th element of **s4**) have been matched to the pattern "Bayer", but notice that R failed to recognize that the third element of **s4**, "BAYER", should also be a match, as it clearly refers to the same company. The code you write to solve this Problem 5 should be able to correctly identify matches in such cases.

III. PROBLEM DESCRIPTION:

Your goal in this problem is to create Table 5 that would contain all of the data from Table 3 plus a new column, *Total number of patents*, in which you should store the *total number of granted patents* (i.e. patents with values of “Status” equal to “*Granted*”) that each company from Table 3 owns. The data for this new column should be derived from Table 4.

Table 5 should be structured as follows:

Company name	Share price	Founded	Number of employees	Total number of patents
...
...
...

1. If a company is absent from Table 3, but present in Table 4, you should not add it to Table 5.
2. If a company is present in Table 3, but absent from Table 4, you should add it to Table 5 and consider its total number of patent applications to be equal to zero.
3. If a company has filed multiple patent applications according to Table 4, you should set its number of patent applications in Table 5 to the grand total (i.e. Table 5 should not contain multiple entries for one company).
4. Patent application should only count towards the grand total of a given company's patents if its status is “*Granted*”.

R PROGRAMMING TASKS:

Create an R code (one .r script) to perform the following tasks:

1. Your code should first recreate Table 3 and Table 4 in R as two separate R objects.
2. Your code should match entries from Table 3 with entries from Table 4 using fuzzy matching (you must use function `agrep()`; see Hints 1 and 2). Specifically, your script should compare company names in Table 3 and Table 4 and correctly match an entry for company A in Table 3 to all (if any) *granted* patent applications by company A in Table 4, despite the fact that different conventions are used for storing company names in Table 3 and Table 4. The pattern matching is not case sensitive.
3. Your code should output a new R object, which should contain all of the data from Table 3 and also information about the *total* number of *granted* patent applications (i.e. patent applications with “Status” equal to “*Granted*”) by each company in Table 3 (please, follow the problem description above).

Specifically, the final R object must include:

- company names (“Company name” from Table 3);
- prices of their shares (“Share price” from Table 3);
- date of establishment (“Founded” from Table 3);
- number of employees (“Number of employees” from Table 3);
- the *total* number of *granted* patent applications (based on Table 4) filed by the respective company.

(Note: in case no entries from Table 4 can be matched with a company in Table 3, assume that it has zero patent applications)

4. Observations in this final R object should be sorted by the total number of patents (in descending order).

5. Finally, your code should display **ONLY** the first FIVE entries from Table 5 as follows:

	Company name	Share price	Founded	Number of employees	Total number of patents
1
2
3
4
5

(Note: the “...” fields should contain actual data)

REQUIREMENTS:

1. Solving this problem, you are not allowed to have any packages loaded into your R-script (no `library()` allowed)
2. You must use function `agrep()` in your R code (see Hints 1 and 2 for an example of use).
3. You are not allowed to use `dplyr`-package in your R code.
4. Provide the screenshot of the first five entries (from Table 5) from the RStudio console.

NOTES:

For your convenience, Table 3 and Table 4 are presented in the .csv format. Please, find the attached “ban401_fuzzy-matching-table3.csv” and “ban401_fuzzy-matching-table4.csv” files in WISEFlow. You are allowed to use the given csv files to recreate Table 3 and Table 4 in R.

All the data presented in the csv files (“ban401_fuzzy-matching-table3.csv” and “ban401_fuzzy-matching-table4.csv”) is fully consistent with the data presented in the Tables (Table 3 and Table 4).

If you import the data from these csv files into R, make sure that all the data (presented in the given csv-files) is consistent with the data presented in Table 3 and Table 4 in this PDF.

If you see that some characters are different (if you import the data from the csv-files) from the numbers presented in Table 3 and Table 4 (in this PDF), this is due to the character encoding settings on your PC. In this case, please make sure that you use the correct symbols presented in Table 3 and Table 4 in this PDF.

PROBLEM 6

Explore the database “Database_Problem.db” available in WISEFlow. Write an SQL-query to retrieve the list of customers who ordered product ‘Lime’. Your SQL-query should return the customers’ names and their addresses:

name	address
.

- Provide the SQL query in the PDF-report.
- Provide the screenshot of the returned results (i.e., the list of customers including names and addresses)
- Please, remember to attach the **.txt** file (with the SQL query code) to your submission
The name of the .txt file should include problem number: problem_6.txt

Tip 1: use *DB Browser for SQLite*

Tip 2: you are free to use any tools to create and save **.txt** file with the SQL query code. The following online tool can be helpful as well:

<https://www.editpad.org/>

PROBLEM 7

Two companies X and Y have their own databases (i.e., databases are independent and not connected to each other). Company X makes a decision to buy Company Y.

PERFORM ANALYSIS:

Discuss three challenges that the integration will face. For each challenge, provide example(s), and/or relative case(s), and/or (hypothetical) real life situation(s).

Minimum (for the problem 7): 1000 words excluding codes (if any), figures (if any), tables (if any), ER-diagrams (if any) and references (if any).

Maximum (for the problem 7): 2000 words excluding codes (if any), figures (if any), tables (if any), ER-diagrams (if any) and references (if any). Text above the maximum limit will be ignored.