Create a file called 'lab9-takehome-ID.py' where ID is your actual BCIT id, e.g. lab9-takehome-A01002131.py

IMPORTANT NOTES

- If your filename does not contain your student id, and does not follow the naming convention specified above, you will lose marks
- Ensure all of your code is uncommented. I will not be making any edits to your file in order to get it to run, so ensure you run and test your script multiple times before submitting, otherwise you will lose marks
- If your script contains syntax or runtime errors, you will lose marks
- For each exercise below I have given you an example of correct output for various inputs. **Your submission should contain the exact function calls that I have included with each question, so that I can see correct output for each exercise**
- Solutions to any of the exercises cannot use
  - list comprehensions
  - functions not discussed in class
  - libraries/modules not discussed in class
- You can still get marks for incomplete submissions, but copied code will result in a zero

Exercise 1

Create a Python function called matrix2 that takes no arguments.

The function must:

- Reads the included file 'lab9-inclass.txt' line by line (use .readline()) - ensure you understand the data in the file before proceeding
- Returns a dictionary:
    - The key is the name of an instructor
    - The value is the number (not the course names!) of courses that instructor teaches

RULES:

- Instructors can only appear in the list once
- Instructor names should not contain whitespace
- Ignore empty lines in the file
- lab9-inclass.txt cannot be modified by you, and your script will be tested against my own copy of the file

Correct output:

```
>>> print(matrix2())
{'Tim': 2, 'Akila': 2, 'Chris': 1, 'Johnny': 1}
```

Exercise 2

Create a function called instructor2. The function must re-use the matrix2 function, and takes no arguments.

The function must:

- Convert the value returned by matrix2 into an alphabetically sorted list of tuples, where each tuple contains the instructor's name and the number of courses they teach
- Return the sorted list of tuples

Correct output (note the data types):

```
>>> print(instructor2())
[('Akila', 2), ('Chris', 1), ('Johnny', 1), ('Tim', 2)]
```

Exercise 3

Create a function called instructors2 that follows the same logic as above, but returns a tuple where the count of courses for each instructor is a string of asterisk characters instead of an integer

Correct output:

```
>>> print(instructors2())
[('Akila', '**'), ('Chris', '*'), ('Johnny', '*'), ('Tim', '**')]
```

Exercise 4

Create a function called instructors2_format that takes no arguments.

The function must call instructors2 and print each instructor one-per line.

Correct output:

```
>>> instructors2_format()
Akila **
Chris *
Johnny *
Tim **
```

Exercise 5

Create a function called leet. The function accepts a single string argument.

The function must return a modified version of the string argument using re.sub() (i.e. the sub() function of the re, 'regular expression' built-in python module)

The function must perform the following substitutions on the argument:

- The character 'e' or 'E' is changed to the character '3'
- The character 'l' or 'L' is changed to the character '1'
- The character 't' or 'T' is changed to the character '7'
- The character 's' or 'S' is changed to the character '5'
- The character 'a' or 'A' is changed to the character '4'

Rules:

- The case of any non-modified characters must not be changed (i.e. anything other than the characters noted above)

Example correct output:

```
>>> print(leet('Leetspeak leetSpeaK'))
'13375p34k 13375p34K'
```

Exercise 6

Create a function called separate_and_sort. The function accepts a single string argument.

The function must return a string with the alphabet characters sorted and separated from the numeric characters.

Rules:

- All alphabet characters must be lowercased
- All alphabet characters must be sorted a-z
- All numeric characters must be sorted in ascending order
- Duplicate characters (irrespective of case) must be included in the output

Example correct output:

```
>>> print(separate_and_sort('Chris Harris ACIT1515'))
'aacchhiiirrrsst1155'
```

Exercise 7

Create a function called list2dict. The function accepts a single list argument.

The function must return a dictionary of key/value pairs of adjacent values (i.e. values that are *next to each other* in the list).

Example:

The list ['Chris', 'ACIT1515', 'Akila', 'ACIT1620']

would return the dictionary

{ 'Chris': 'ACIT1515', 'Akila': 'ACIT1620' }

Rules:

- Only lists with an even number of values can be converted. Lists with an odd number of values result in no output
- Keys must be strings or integers. A key/value pair with a key that is not valid are not included in the output
- Keys cannot be blank (i.e. empty strings)
- Values can be any data type
- Duplicate keys are ignored (as are their values)
- You can assume the list passed into the function contains simple values like the examples below

Example correct output:

```
>>> print(list2dict(['Chris', 'ACIT1515', 'Akila']))

>>> print(list2dict(['Chris', 'ACIT1515', ['Akila'], 'ACIT1620', 'Tim', 'ACIT2515']))
{ 'Chris': 'ACIT1515', 'Tim': 'ACIT2515' }
>>> print(list2dict(['Chris', 'ACIT1515', 'Chris', 'ACIT2811']))
{ 'Chris': 'ACIT1515' }
>>> print(list2dict(['Chris', 'ACIT1515', 'Akila', 'ACIT1620', 'Chris', 'ACIT2811']))
{ 'Chris': 'ACIT1515', 'Akila': 'ACIT1620' }
```

Exercise 8

Create a function called count_by_key. The function accepts two arguments, a list of dictionaries that each contain at least one key/value pair, and a string key.

The function must iterate through the list and return a count how many unique values appear for the specified key, for example:

```
data = [
    { 'make': 'honda', 'color': 'red', 'year': 2012 },
    { 'make': 'subaru', 'color': 'blue', 'year': 2003 },
    { 'make': 'toyota', 'color': 'blue', 'year', 2020 },
    { 'make': 'mistubishi', 'color': 'black', 'year', 1997 }
]
```

returns 4 for count_by_key(data, 'make') because there are 4 different makes
returns 3 for count_by_key(data, 'color') because there are 3 different colors (red, blue, and black)

Example correct output:

```
>>> print(count_by_key([{'a': 2}], 'a'))
1
>>> print(count_by_key([{'a': 2, 'b': 4}, {'a': 1}, {'a': 1, 'b': 4}, {'a': 4, 'b': 4}], 'a'))
3
>>> print(count_by_key([{'a': 2, 'b': 4}, {'a': 1}, {'a': 1, 'b': 4}, {'a': 4, 'b': 4}], 'b'))
1
```

Exercise 9

Create a function called compat. The compat function takes two arguments.

The function must:

- Return the concatenation of the two arguments if they are the same data type
- If the data types are different, the function converts the first argument to the data type of the second argument and returns the concatenation of the result
- Handle errors in any cases where the resulting concatenation would result in an illegal operation, e.g. None + None
- Illegal operations result in no output

Example correct output:

```
>>> print(compat(2, 2))
4
>>> print(compat('2', '2'))
'22'
>>> print(compat('2', 2))
4
>>> print(compat('15', ['1', '5']))
['1', '5', '1', '5']
>>> print(compat(None, 10))

>>> print(compat(None, 'such'))
'Nonesuch'
```

Exercise 10

Create a function called operation. The function takes four arguments, lhs and rhs, f (a function), and count (the number of times to execute f)

Create four more functions, all of which take two arguments, lhs and rhs

- add, which returns the sum of lhs and rhs
    - subtract, which returns the difference of lhs and rhs
        - multiply, which returns the product of lhs and rhs
        - divide, which returns the quotient of lhs and rhs, and additionally prevents division by zero

The operation function must:

- Call f exactly 'count' times, using the result as input for any further calls to f
- Return the final result of all calls to f

Example correct output:

```
>>> print(operation(4, 3, add, 2))          # 4+3=7, 7+7=14
14
>>> print(operation(4, 4, subtract, 3))     # 4-4=0, 0-0=0, 0-0=0
0
>>> print(operation(1, 2, multiply, 5))     # 1*2=2, 2*2=4, 4*4=16, 16*16=256, 256*256=65536
65536
>>> print(operation(4, 4, divide, 2))       # 4/4=1, 1/1=1
1
```