You should follow the instructions in naming the files, capturing the input and printing the output, otherwise our automated grading procedure will not work correctly and you will lose marks. On top of correct program output, you are also marked for coding style. Make sure you have ticked the Enable code style linting option on Spyder for conformance to the PEP 8 convention. You are NOT ALLOWED to import modules for this coursework. Standard university penalty of **5%** of available marks per day, or part of a day, will apply to late work. Late submissions are acceptable **up to 7 days late**.

Submission: You must submit your work via Gradescope.

Deadline: **2400 CST 13/10/21.**

Weighting: This coursework is worth 30% of the module grade.

1. Calculate annual income based on tax (5 marks), coding style (1 mark)

Write a `main()` function to calculate annual income based on the income tax entered by the user from the command prompt. **Do not include any additional `print` statement before user input.**

The income tax is calculated based on the following formula:

Total Income	Tax rate
0 -12,500	0%
12,501 - 50,000	20%
50,001 - 150,000	40%
Over 150,000	45%

Income tax is paid on the amount of taxable income remaining after allowances have been deducted. The calculated income is rounded to two decimal points. If tax is 0.00, then the default annual income is set to 12500.00. If invalid amount of tax is entered (e.g., abc, 20.ab), the program prints `"Invalid amount!"`. **Make sure to only print out the annual income or `"Invalid amount!"`**. Name the file for this program as cwk1a.py

Annual income	Tax
14000.00	300.00
100000.00	27500.00
180000.00	61000.00

2. Extended Vigenère cipher (5 marks), coding style (1 mark)

Write a `ext_vigenere(text, key, option)` function to implement an extended Vigenère cipher that accept 3 arguments: the `text` (plaintext/ciphertext), the `key`, and the `option` either to "`encrypt`" or "`decrypt`" and returns the encrypted/decrypted `text`. If invalid option is provided, the function returns "`Invalid option!`"

In this cipher, the encryption/decryption works by referring to the Vigenère table.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 1: Vigenère table

Figure 1 shows the table for the original Vigenère table. To encrypt, the `key` refers to the row and the `plaintext/ciphertext` is referred to the column. For example, given a plaintext `VIGENERE` and a key `CIPHER`, the encryption will produce the ciphertext `XQVLRVTM` as shown below. The key "C" refers to row C, and the plaintext "V" refers to column V. The intersection is the ciphertext, in this case, the letter "X". To decrypt, reverse the operations.

Plaintext	V	I	G	E	N	E	R	E
Key	C	I	P	H	E	R	C	I
Ciphertext	X	Q	V	L	R	V	T	M

You are required to write an extended Vigenère cipher covering all printable characters on the ASCII table (i.e., characters with ASCII equivalent value from 32 (space) to 126 (~). You can refer to the ASCII table at http://www.asciitable.com/. Name the file for this program as cwk1b.py. **You do not need to write codes to capture user input or to print the output.**

Table below shows some example of returned results.

Examples	
Calling the function	**Returns**
`ext_vigenere ("VIGENERE", "CIPHER", "encrypt")`	`yrwmswun`
`ext_vigenere ("<UcY6dK`", "testing", "decrypt")`	`GoodLuck`
`ext_vigenere ("<UcY6dK`", "testing", "endecrypt")`	`Invalid option!`

For more details regarding this cipher, please refer to https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher and read section headed "Description".

3. Highest count items (5 marks), coding style (1 mark)

Write a `count_occurences(input_string)` function to return a list of item(s) with highest count in `input string`. Each item in `input_string` is comma-separated. The output of the program is a `list` in the format [[item, count]]. If there is a tie, return all items with the highest count (i.e., [[item1, count], [item2, count], etc...]). **Make sure to only return the list and nothing else is printed.** Name the file for this program as cwk1c.py.

Examples of the input and returned list:

input_string	**Returned list**
3, 13, 7, 9, 3, 3, 5, 7, 12, 13, 11, 13, 8, 7, 5, 14, 15, 3, 9, 7, 5, 9, 14, 3, 8, 2, 5, 5, 8, 14, 11, 11, 12, 8, 5, 3, 3, 10, 3, 10, 7, 7, 10, 10, 2, 7, 4, 8, 1, 5	[['3', 8]]
British Gas, British Gas, Ovo, Igloo, Igloo, Octopus, Bulb, Shell, E.ON, Npower, British Gas, Octopus, Igloo, Npower, Igloo, Shell, Scottish Power, Bulb, EDF, Bulb, EDF, Bulb, Scottish Power, Octopus, Scottish Power, Octopus, Octopus, EDF, Ovo, Shell, Octopus, E.ON, British Gas, Bulb, Npower, Shell, Scottish Power, Npower, Scottish Power, Npower	[['Octopus', 6]]
aac, ctt, gat, ccc, gcc, ctg, gtc, tcg, ccg, cca, ata, cca, tat, ata, cac, gcg, cca, gta, gtg, gac, taa, ata, gtc, aat, gct, gta, ggc, tgc, tca, ctt, tgt, ata, acc, tgc, gac, cgc, atc, cgt, tac, agg, ctt, cgc, cgc, tgg, gcg, tgg, taa, cta, aaa, tgc, cgt, tgc, gac, tta, aag, taa, act, cca, tac, agg, cgc, gtg, cca, gcg, gtt, gag, tca, aca, tct, gta, ata, ctt, gat, tcg, tcg, cac, cgt, tac, caa, aac, ctg, tgt, aag, ttc, ccc, tcc, ctc, cct, aga, gtt, tga, gaa, cct, ctc, tct, ggt, gcc, tct, ccc, agt, caa, gac, ccc, cgc	[['cca', 5], ['ata', 5], ['cgc', 5]]

4. Game of Life (5 marks), coding style (1 mark)

Write a `gameoflife(x_cycle)` function to simulate the game of life based on a given initial population, `origin`, represented as a list of lists and returns the final population(list) after `x_cycle` number of cycles. Please refer to the Wikipedia page on Game of Life at https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life and read the sections headed "Rules" and "Examples of patterns". For a cell on the edge of the world, the neighbours outside the world are considered dead. In the end, return final population (the `list`). **Make sure to only return the list and nothing else is printed.** Name the file for this program as cwk1d.py.

```
origin = [[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
          [0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0],
          [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1],
          [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

5. Word Search (5 marks), coding style (1 mark)

Write a `wordsearch(search_string)` function for a word search game based on a given puzzle represented as a list of lists. If you are unfamiliar with world search puzzles, refer to https://en.wikipedia.org/wiki/Word_search and read the section headed "Strategies". The words may be placed horizontally, vertically, or diagonally. If the word is found, return the coordinates of all the characters as a `list of tuples`; else print "`Not found!`" If a word is found at multiple coordinates, return all coordinates, as a list of lists**. Make sure to only return the list nothing else is printed.** Name the file for this program as cwk1e.py.

```
puzzle = [['R', 'U', 'N', 'A', 'R', 'O', 'U', 'N', 'D', 'D', 'L'],
          ['E', 'D', 'C', 'I', 'T', 'O', 'A', 'H', 'C', 'Y', 'V'],
          ['Z', 'Y', 'U', 'W', 'S', 'W', 'E', 'D', 'Z', 'Y', 'A'],
          ['A', 'K', 'O', 'T', 'C', 'O', 'N', 'V', 'O', 'Y', 'V'],
          ['L', 'S', 'B', 'O', 'S', 'E', 'V', 'R', 'U', 'C', 'I'],
          ['B', 'O', 'B', 'L', 'L', 'C', 'G', 'L', 'P', 'B', 'D'],
          ['L', 'K', 'T', 'E', 'E', 'N', 'A', 'G', 'E', 'D', 'L'],
          ['I', 'S', 'T', 'R', 'E', 'W', 'Z', 'L', 'C', 'G', 'Y'],
          ['A', 'U', 'R', 'A', 'P', 'L', 'E', 'B', 'A', 'Y', 'G'],
          ['R', 'D', 'A', 'T', 'Y', 'T', 'B', 'I', 'W', 'R', 'A'],
          ['T', 'E', 'Y', 'E', 'M', 'R', 'O', 'F', 'I', 'N', 'U']]
```

Some of the words in the puzzle are `runaround`, `scalar`, `tray`, `gazebo`, `convoy`, `chaotic`, and `sleepy`.

Example	
search_string	**Returned list**
`runaround`	`[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8)]`
`chaotic`	`[(1, 8), (1, 7), (1, 6), (1, 5), (1, 4), (1, 3), (1, 2)]`