Assignment 2

Instructions

- Unless specified otherwise,
  - you can use anything in the Python Standard Library;
  - don't use any third-party library in your code for this assignment;
  - you can assume all user inputs are always valid and require no validation.
- The underlined blue texts in the Sample Runs section of each question refers to the user inputs. It does NOT mean that your program needs to underline them.
- Please follow closely the format of the sample output for each question. Your program should produce **exactly** the same output as the sample (same text, symbols, letter case, spacing, etc.). The output for each question should end with a single newline character, which has been provided by the `print()` function by default.
- If you see fit to define your own functions for specific tasks in any question, you may do so, but this is not required.
- Name your script files as instructed in each question (case-sensitive). Using other names may affect our marking and may result in deduction.
- Your source files will be tested in script mode rather than interactive mode.

Question 1 – ASCII Art (30%)

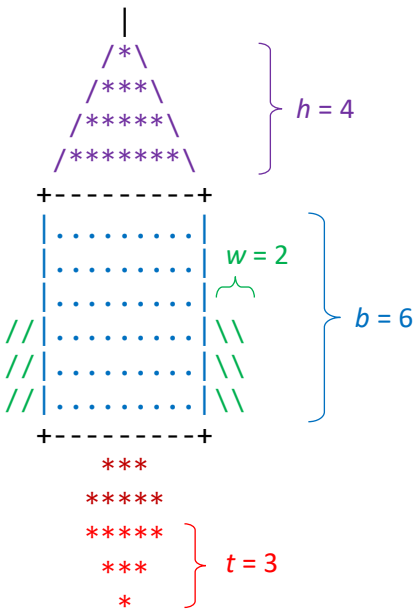
Write a Python script `rocket.py` to print a rocket pattern using symbols in the ASCII character set as console output such as the following:

```
      |
     /*\
    /***\
   /*****\
  /*******\
  +---------+
  |.........|
  |.........|
  |.........|
//|.........|\\
//|.........|\\
//|.........|\\
  +---------+
     ***
    *****
    *****
     ***
      *
```

There are four parameters to control the overall shape of the rocket:

1. ***h* (head)**: controls the height of the rocket's head.
2. ***b* (body)**: controls the height of the rocket's body.
3. ***w* (wing)**: controls the width of the rocket's wings.
4. ***t* (tail)**: controls the height of the rocket's tail (or flame).

To ease visualizing which part of the shape each of these parameters refers to, please look at the following drawing with each parameter annotated:



Part 1: Rocket Head

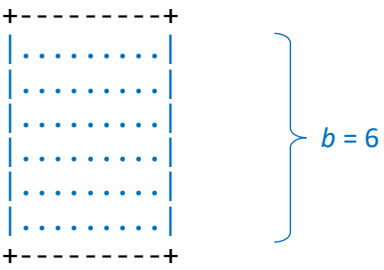
For the first line to print, the rocket pattern begins with a single pipe symbol "|", which acts as the tip of the rocket's head. Then the program prints a triangular pattern which has its left and right edges formed by forward slashes "/" and backslashes "\" respectively, and is filled with asterisks "*". For example, it looks like:

```
   /*\
  /***\
 /*****\
/*******\
```

The number of "*" per line is always an odd number. The height of this triangular shape (a.k.a. the height of the head) is denoted by *h*, which is entered by the user via standard input. The above example refers to the case of *h* = 4, so this triangular shape is composed of four lines. The height of the head refers to the number of lines of asterisks "*" only, excluding the "|" tip and the line that follows, which looks like "+----------+".

Part 2: Rocket Body

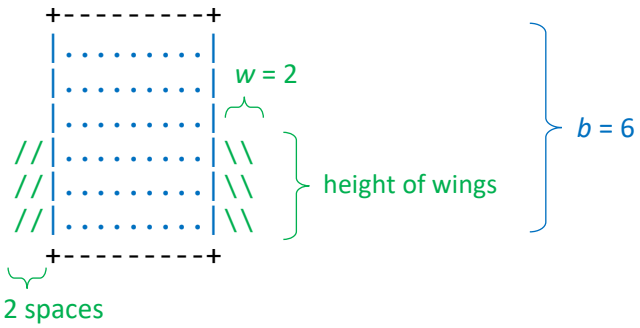
There are two lines "+----------+" at the top and bottom of the rocket's main body. They are formed by hyphens and a plus at the start and end positions of the line ("+" 's are printed at the four corners of the rocket's body).



The body part is filled with dot "."; the left and right edges are formed by the pipe "|" symbol. Note that *b* refers to the height of the body; it counts the lines of dots only, excluding the top and bottom lines "+----------+". So, for the above example, *b* is 6 only.

Part 3: Rocket Wings

The rocket has two wings: the left wing and right wing are denoted by patterns of forward slashes "/" and backslashes "\" respectively.

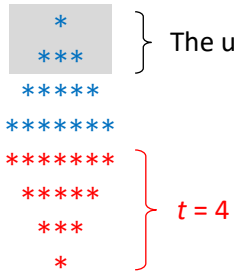


The two wings always have equal widths and heights. The height of the wings is set to half of the rocket body's height. For indivisible cases, we take the ceiling of the division, i.e., $\left\lceil \frac{b}{2} \right\rceil$.

For the above example, the wings have a width of 2, i.e., $w = 2$, and a height of 6 / 2 = 3.
Let's say if $b = 7$, then 7 / 2 = 3.5, taking the ceiling gives 4, so the wings' height is also 4.

Part 4: Rocket Tail (Flame)

The tail or flame part is the most complicated part in this assignment. It looks like a diamond shape but with some subtle differences. It is formed in the following way. Think of it as if two triangles are stacking on top of another, with the bottom one inverted:



What's more, the upper half of the top triangle is chopped (grey out in the above diagram). The parameter t refers to the height of the bottom triangle. For the above example, $t = 4$. For odd t, the number of lines of asterisks to keep for the top triangle is the ceiling of the division of t by 2, i.e., $\left\lceil \frac{t}{2} \right\rceil$.

The following table shows the shapes of the tails for different values of t. For example, for $t = 5$, we have 5 / 2 = 2.5, taking the ceiling gives 3. So, we keep/print the lower 3 lines and chop the upper 2 lines from the top triangle for the case of $t = 5$.

t	2	3	4	5	6	7	8
Shape of the tail	*** *** *	*** ***** ***** *** *	***** ******* ******* ***** *** *	***** ******* ********* ********* ******* ***** *** *	******* ********* *********** *********** ********* ******* ***** *** *	******* ********* *********** ********** ********** *********** ********* ******* ***** *** *	********* *********** ********** ********** ********** *********** ********** ********* ******* ***** *** *

Requirements on Input Validation

For the drawing to look like a rocket, the following bounds on the input values are enforced:

- $h \geq 1$
- $b \geq 2$
- $w \geq 1$
- $t \geq 2$

There is no upper bound validation necessary for the parameters *h*, *b* and *w*. But due to the visibility constraints of a console screen, we assume only small values for *h*, *b* and *w* for testing your program, practically up to around 30 for each of these parameters.

However, for *t*, its size should have an upper bound since we don't want the tail to look wider than the rocket body. Let's do some analysis below:

- The width (number of "*") of the base of the triangle for the head part is $2h - 1$.
- The width (including all characters "|.....|") of the body is $2h + 3$.
- The width of the tail (flame), i.e., $2t - 1$, should NOT be wider than the width of the rocket body. So, the user inputs must satisfy the following inequality:

$$2t - 1 \leq 2h + 3, \text{ or } t \leq h + 2$$

You may assume that the user always enters positive integer values. However, if the user inputs for the 4 parameters violate any of the above requirements, your program will prompt the user again until a valid set of values is received.

Sample Runs

```
Enter h, b, w, t: 1 2 1 2↵
   |
  /*\
 +---+
 |...|
/|...|\
 +---+
  ***
  ***
   *
```

```
Enter h, b, w, t: 2 2 3 3↵
      |
     /*\
    /***\
   +-----+
   |.....|
///|.....|\\\
   +-----+
     ***
    *****
    *****
     ***
      *
```

```
Enter h, b, w, t: 4 6 2 3↵
       |
      /*\
     /***\
    /*****\
   /*******\
  +---------+
  |.........|
  |.........|
  |.........|
//|.........|\\
//|.........|\\
//|.........|\\
  +---------+
     ***
    *****
    *****
     ***
      *
```

```
Enter h, b, w, t: 5 5 4 7↵
         |
        /*\
       /***\
      /*****\
     /*******\
    /*********\
   +-----------+
   |...........|
   |...........|
////|...........|\\\\
////|...........|\\\\
////|...........|\\\\
   +-----------+
      *******
     *********
    ***********
   *************
   *************
    ***********
     *********
      *******
       *****
        ***
         *
```

```
Enter h, b, w, t: 1 1 1 1↲      ← b and t are too small!
Invalid input!
Enter h, b, w, t: 1 2 1 1↲      ← t is too small!
Invalid input!
Enter h, b, w, t: 1 1 2 2↲      ← b is too small!
Invalid input!
Enter h, b, w, t: 3 4 5 8↲      ← t is too big!
Invalid input!
Enter h, b, w, t: 3 4 5 6↲      ← t is too big!
Invalid input!
Enter h, b, w, t: 3 4 5 5↲
          |
         /*\
        /***\
       /*****\
      +-------+
      |.......|
      |.......|
/////|.......|\\\\\
/////|.......|\\\\\
      +-------+
        *****
       *******
      *********
      *********
       *******
        *****
         ***
          *
```

Question 2 – Sudoku Checker (15%)

Write a Python script called `sudoku.py` to implement a Sudoku checker. As you may know, a Sudoku is a 9x9 grid (see an example below) that is completed when every 3x3 square, row and column contain all of the digits from 1 to 9.

4	3	5	2	6	9	7	8	1
6	8	2	5	7	1	4	9	3
1	9	7	8	3	4	5	6	2
8	2	6	1	9	5	3	4	7
3	7	4	6	8	2	9	1	5
9	5	1	7	4	3	6	2	8
5	1	9	3	2	6	8	7	4
2	4	8	9	5	7	1	3	6
7	6	3	4	1	8	2	5	9

Write a function `is_sudoku()` that receives a completed 9x9 grid, in the form of a two-dimensional (2D) list, and returns True if the grid represents a valid Sudoku and False otherwise. You may define other functions as callees for this function if it can help decompose the overall task better.

Note: we will import your sudoku module into our test scripts for testing against our test cases. You should have your own client script that imports the module for using and testing the function against your own test cases such as the following:

```
A = [
    [4,3,5,2,6,9,7,8,1],
    [6,8,2,5,7,1,4,9,3],
    [1,9,7,8,3,4,5,6,2],
    [8,2,6,1,9,5,3,4,7],
    [3,7,4,6,8,2,9,1,5],
    [9,5,1,7,4,3,6,2,8],
    [5,1,9,3,2,6,8,7,4],
    [2,4,8,9,5,7,1,3,6],
    [7,6,3,4,1,8,2,5,9],
]
print(is_sudoku(A))
True

B = [
    [4,3,5,2,6,9,7,8,1],
    [6,8,2,5,7,1,4,9,3],
    [1,9,7,8,3,4,5,6,2],
    [8,2,6,1,9,5,3,4,7],
    [3,7,4,6,8,2,9,1,5],
    [9,5,1,7,4,3,6,2,8],
    [5,1,9,3,2,6,8,7,4],
    [2,4,8,9,5,7,1,3,6],
    [7,6,3,4,1,8,2,5,0],
]  # not a Sudoku; the last element is 0
print(is_sudoku(B))
False

C = [
    [5,3,4,6,7,8,9,1,2],
    [6,7,2,1,9,5,3,4,8],
    [1,9,8,3,4,2,5,6,7],
    [8,5,9,7,6,1,4,2,3],
    [4,2,6,8,5,3,7,9,1],
    [7,1,3,9,2,4,8,5,6],
    [9,6,1,5,3,7,2,8,4],
    [2,8,7,4,1,9,6,3,5],
    [3,4,5,2,8,6,1,7,9],
]
print(is_sudoku(C))
True
```

Question 3 – Alphabet Grids (25%)

Write a Python script called `algrid.py`, which implements the following functions for creating and manipulating 2D arrays of alphabets, which we call "alphabet grids":

(a) `random_grid(m, n, unique)` that takes two positive integers, `m` and `n`, and a Boolean, `unique`, (default to `False`) as arguments, and returns an *m*-by-*n* array (implemented using a nested Python list) containing random alphabets between 'A' and 'Z' or 'a' and 'z' as elements.

For example, calling `random_grid(3, 4)` may return a nested list like this:

```
[['p', 'O', 'v', 'U'], ['f', 'V', 'Y', 'k'], ['F', 'b', 'U', 'k']]
```

If the `unique` parameter is set to `True`, every element of the array must be different from one another (note: uppercase 'A' and lowercase 'a' are considered different).

For example, calling `random_grid(3, 4, True)` may return:

```
[['V', 'F', 'a', 'x'], ['T', 'o', 'Z', 't'], ['R', 'z', 'M', 'Q']]
```

Note: the 1st example has 'k' and 'U' duplicated while the elements in the 2nd example are all unique. The elements are randomly picked; the dimension is 3 rows by 4 columns.

When `unique` is `True`, the size `m` × `n` must be ≤ 52. If the combination of arguments violates this, raise a `ValueError` exception and handle it gracefully by printing the message: "The grid is too big for having unique items!" and exiting the program immediately.

For example, executing the following code

```
grid = random_grid(10, 10, True)
```

will cause the program to end prematurely (`grid` still unassigned) with the following output:

```
The grid is too big for having unique items!
```

(b) `print_grid()` that receives a nested list argument and prints it with each row on a separate line. For example, suppose that the following alphabet grid is passed to the function:

```
grid = [['p', 'O', 'v', 'U'], ['f', 'V', 'Y', 'k'], ['F', 'b', 'U', 'k']]
```

`print_grid(grid)` will output the following to the screen:

```
[['p', 'O', 'v', 'U'],
 ['f', 'V', 'Y', 'k'],
 ['F', 'b', 'U', 'k']]
```

Note: every line has a newline character immediately following the last character (',' or ']').

(c) `sorted_grid(grid, row_wise)` that receives an alphabet grid and a Boolean argument, `row_wise` (default to `True`), and returns a new alphabet grid with elements alphabetically sorted in a row-wise (or column-wise) manner. For example, suppose that the following alphabet grid is passed to the function:

```
grid = [['W', 'I', 'I', 't'], ['e', 'q', 'A', 'f'], ['f', 'i', 'y', 'e']]
```

`sorted_grid(grid)` will return the following list:

```
[['A', 'e', 'e', 'f'], ['f', 'i', 'I', 'I'], ['q', 't', 'W', 'y']]
```

whereas `sorted_grid(grid, False)` will return the following list:

```
[['A', 'f', 'q'], ['e', 'i', 't'], ['e', 'I', 'W'], ['f', 'I', 'y']]
```

For better visualization, we show the outputs of printing the above arrays using the `print_grid()` function developed in part (b) as follows:

Random alphabet grid:
```
[['W', 'I', 'I', 't'],
 ['e', 'q', 'A', 'f'],
 ['f', 'i', 'y', 'e']]
```

Sorted grid (row-wise):
```
[['A', 'e', 'e', 'f'],
 ['f', 'i', 'I', 'I'],
 ['q', 't', 'W', 'y']]
```

Sorted grid (column-wise):
```
[['A', 'f', 'q'],
 ['e', 'i', 't'],
 ['e', 'I', 'W'],
 ['f', 'I', 'y']]
```

Note: The original grid should remain unchanged after any call of `sorted_grid()`. Another point to note is that for the same alphabet, we specially require lowercase to precede uppercase. So, in the above example, 'i' appears before 'I'. Hint: think about how to set the `key` parameter if you would use the `sort()` or `sorted()` method of the `list` class to achieve this ordering.

(d) `save(grid, filename)` that writes the content of the argument `grid` to a text file specified by the `filename` argument.

Suppose that `grid` refers to the above example in part (b). Calling `save(grid, "mygrid.txt")` will create a text file named mygrid.txt in the current directory. The file content is as follows:

```
p O v U\n
f V Y k\n
F b U k\n
```

Note: a single space is between every two letters; each line ends with a newline ('\n') character.

(e) `restore(filename)` that takes a string argument `filename`, reads the content of the text file with the specified file name, creates an alphabet grid using the file content as its elements, and returns the grid to the caller.

Suppose that a text file named mygrid.txt storing the file content as shown in part (d) is in the current directory. Calling `restore("mygrid.txt")` will return the grid in part (b).

Question 4 – Weather Forecast (20%)

The Hong Kong Observatory provides real-time weather data in JSON format. The following URL is one of their data APIs:

https://data.weather.gov.hk/weatherAPI/opendata/weather.php?dataType=rhrread&lang=en

There are two types of weather information available: 1. Rainfall, 2. Temperature. The data of either type can be different for different places.

Write a Python script `weather.py` to fetch a current weather report from the above link and print the regional weather information the user chooses from a console menu (see Sample Runs below).

Hints:
To fetch a resource (e.g. a JSON file) from an URL, you may use the `urllib.request.urlopen()` function. For example,

```
import urllib.request
response = urllib.request.urlopen(URL)    # URL: the above link
```

You can call `response.read()` to get a `bytes` object of the HTTP response, which contains the JSON document. It is better to open the link under a `with` statement. Read this page to see an example.

Then you can use `json.loads()`, which supports `bytes` type, to load the bytes object of the JSON data into a Python dictionary.

You may see some timestamp data looking like this: "2021-08-19T23:15:00+08:00". To print the expected output, the function call `datetime.strptime(s, '%Y-%m-%dT%H:%M:%S%z')` will be useful for parsing the input string `s` into a `datetime` object. Then you can call the method `strftime(format)` on the `datetime` object to get time strings in specific format for printing.

There is an HTML entity written as `&amp;` in the place name "Central & Western District". You can import the html package and call `html.unescape()` on the string to print "Central & Western District" instead.

For temperature information, the date time in the output refers to the 'recordTime' field in the JSON document.

Sample Runs

```
1. Rainfall
2. Temperature
Which type of weather info? 1↵
 1. Central & Western District
 2. Eastern District
 3. Kwai Tsing
 4. Islands District
 5. North District
 6. Sai Kung
 7. Sha Tin
 8. Southern District
 9. Tai Po
10. Tsuen Wan
11. Tuen Mun
12. Wan Chai
13. Yuen Long
14. Yau Tsim Mong
15. Sham Shui Po
16. Kowloon City
17. Wong Tai Sin
18. Kwun Tong
Which place? 9↵
The rainfall is 0 mm in Tai Po from 2021-10-16 20:45:00 to 2021-10-16 21:45:00.
```

```
1. Rainfall
2. Temperature
Which type of weather info? 2↵
 1. King's Park
 2. Hong Kong Observatory
 3. Wong Chuk Hang
 4. Ta Kwu Ling
 5. Lau Fau Shan
 6. Tai Po
 7. Sha Tin
 8. Tuen Mun
 9. Tseung Kwan O
10. Sai Kung
11. Cheung Chau
12. Chek Lap Kok
13. Tsing Yi
14. Shek Kong
15. Tsuen Wan Ho Koon
16. Tsuen Wan Shing Mun Valley
17. Hong Kong Park
18. Shau Kei Wan
19. Kowloon City
20. Happy Valley
21. Wong Tai Sin
22. Stanley
23. Sham Shui Po
24. Kai Tak Runway Park
25. Yuen Long Park
26. Tai Mei Tuk
Which place? 23↵
The temperature is 24 C in Sham Shui Po at 2021-10-16 22:00:00.
```

Question 5 – Word Count (20%)

Write a Python script `word_counter.py` to parse a text document (such as an essay or newspaper article) and report the top 10 words having the highest frequencies of occurrence in the text.

In this script, define a function called `top_words(filename, n=10)`, which reads the specified text file and returns a dictionary containing (at most) n items denoting the top (most frequent) n words in the file. The dictionary keys and values are the words (`str`) and their frequencies (`int`). The dictionary to return is sorted in **descending order** of values (i.e., by frequencies) instead of keys.

Punctuation Stripping
Punctuation marks can complicate our parsing. For example, "happy" and "happy!" will be treated as different words if we have punctuation symbols in the strings. So, the first thing to do is to strip them off. There are many ways of doing so. Some make use of regular expressions (the re module). Another way is to translate every punctation in the string to a space. More self-study on this is needed.

Stop Words Filtering
Normally, an essay has many words like 'a', 'an', 'the', 'of', 'for', 'that', etc. which bear no meaning for our analysis purpose. These are called "**stop words**". Your program must strip them off before carrying out word counting. We have provided a text file called stopwords.txt, which contains common stop words separated by commas. Assume that this file is in the current directory (next to this Python script) and its filename can be hardcoded as a global variable in this module as follows:

```
STOPWORD_FILENAME = "stopwords.txt"
```

Removing Numbers
In case the document contains some numbers, e.g., 1000, or words composed of alphabets and digits, e.g., "footnote1", "3PO", "R2", these numbers or words should be filtered off too. In other words, we want to retain strings whose characters are all alphabetic.

Finally, all single-letter words are removed too.

Combining all the filtering logics above, note the following examples:

"Covid-19" will become "Covid 19" after punctuation translation; it is then split into "Covid" and "19"; "Covid" is counted while "19" is discarded.

"Russia's" will become "Russia s" after punctuation translation; it is then split into "Russia" and "s"; "Russia" is counted while "s" is discarded.

For simplicity, you don't need handle stemming and lemmatization in this question as one would do in natural language processing (NLP). So, "play", "played", "playing" are considered different words; they will become different keys and have their own word counts as values in the output dictionary. The same applies to letter case and plural forms, so "Bitcoin" and "bitcoin" are different, "future" and "futures" are also taken as different words.

For simplicity, you can assume that no Unicode characters exist, so the encoding of the input text files is simply ASCII.

We have provided some text files containing recent news articles for the testing purpose.

Restriction: As stated in the beginning, don't use any third-party library such as NLTK to attempt this question.

Sample Runs

```
print(top_words('covid-russia.txt'))
{'Russia': 8, 'vaccines': 6, 'Covid': 5, 'people': 5, 'vaccinated': 5, 'number':
5, 'vaccination': 4, 'Sputnik': 4, 'doses': 4, 'deaths': 3}
```

```
print(top_words('covid-pill.txt'))
{'countries': 23, 'pill': 19, 'Covid': 12, 'drug': 11, 'people': 11, 'vaccines':
9, 'Asia': 8, 'vaccine': 8, 'molnupiravir': 8, 'access': 7}
```

```
print(top_words('social-security.txt'))
{'savings': 11, 'Social': 10, 'Security': 10, 'more': 10, 'retirees': 8,
'inflation': 8, 'year': 7, 'interest': 7, 'income': 7, 'benefits': 6}
```

```
print(top_words('bitcoin-prices-etf.txt'))
{'bitcoin': 12, 'SEC': 9, 'Bitcoin': 8, 'ETF': 6, 'investors': 4, 'futures': 4,
'fund': 4, 'Friday': 3, 'approve': 3, 'investing': 3}
```

Note: the above outputs show the output dictionary on 2 lines just because of limited width of this document; in the actual output, there is no line break in the middle when printing the dictionary.