CISS450: Artificial Intelligence
Assignment 1

OBJECTIVES

1. Write procedural python programs.
2. To be acqainted with several standard AI problems.

FALL2022: I'm upgrading to Python 3. The virtual machine is Fedora 31. Let me know if you see errors in the given python code. This is mostly likely due to the change from Python 2 to Python 3.

The goal is to practice (procedural) python programming. At the same time, I will introduce briefly some problems studied in AI. Therefore some of the problems here will actually appear in later actual AI assignments. You won't be solving any of these problems, but you will be writing some functions which will be useful in the future.

For each question, you will get a skeleton code that looks like this:

```
# File  : main.py
# Author: smaug (replace this with your nickname)

# Your code

if __name__ == '__main__':
    # Code to test your code above
```

Make sure the filename is `main.py`.

Remove (or comment out) unnecessary input/output before turning in your work.

IMPORTANT NOTE:

- You should use our Fedora 31 virtual machine.
- As mentioned in class, we are using Python 3. To check the version of your python, at the bash shell do `python --version`. If the output is of the form `Python 3.*.*`, i.e., the major version is 3, you are fine.
- For later assignments, we will be using pygame. If you are using any of our fedora virtual machines, then pygame should already be installed. To verify that pygame is installed, run the python interpreter in your bash shell by doing `python` and then at the python prompt execute `import pygame`. If pygame not is installed, you will get an error message.

Q1. Write a function `matrix` that accepts an integer n and a string of comma-separated values and return a 2-dimensional array (i.e. a list of lists) where each row, except possibly for the last, has n elements using the “values” in the string.

For instance

```
matrix(3, "a,b,c,d,e,f,g,h,i")
```

returns the list

```
[["a","b","c"],["d","e","f"],["g","h","i"]]
```

and

```
matrix(3, "a,b,c,d,e,f,g,h")
```

returns the list

```
[["a","b","c"],["d","e","f"],["g","h"]]
```

Once you’re done with the above, improve on it so that it accepts a separator string which is used to separate values in the second parameter. For instance:

```
matrix(3, "a|b|c|d|e|f|g|h|i", "|")
```

returns the list

```
[["a","b","c"],["d","e","f"],["g","h","i"]]
```

Of course the earlier cases must still work, i.e., the third parameter must have a default value of `','`:

```
def matrix(n, s, separator=','):
    # To be completed
    pass
```

Skeleton code:

```
# File  : main.py
# Author:

# Your matrix function here

if __name__ == '__main__':
    n = int(input())     # for instance enter 3
    s = input()          # for instance enter "a,b,c,d,e,f,g,h,i"
                         # (without quotes)
    separator = input()  # for instance enter "," (without quotes)
                         # or enter "" (without quotes) for default case

    if separator == '':
        print(matrix(n=n, s=s))
    else:
        print(matrix(n=n, s=s, separator=separator))
```

NOTES.

1. Python's input function `input` accepts a string. Try this:

```
i = input("enter i: ") # enter 3
print(i, type(i))
i = input("enter i: ") # enter 3.14
print(i, type(i))
i = input("enter i: ") # enter abc
print(i, type(i))
```

2. The Python statement `pass` means "don't do anything". It's like a C++ empty statement (i.e., just a semicolon). The `pass` in the above function is necessary because a Python block (function, if-else, loops) must have at least one statement.

Q2. Write a function that performs the following list computation. If you call

```
move('right', ['0', '3', '4', '2', ' ', '8'])
```

it will return

```
['0', '3', '4', '2', '8', ' ']
```

i.e. the space will "move to the right". If the space is already at the rightmost spot, the same list is returned. If you call

```
move('left', ['0', '3', '4', '2', ' ', '8'])
```

it will return

```
['0', '3', '4', ' ', '2', '8']
```

i.e. the space will "move to the left". Again if the space is already at the leftmost spot, the same list is returned.

This is a simplification of the $n^2 - 1$ problem (see https://en.wikipedia.org/wiki/15_puzzle) which is an example of sliding tile problems (https://en.wikipedia.org/wiki/Sliding_puzzle). Sliding tile problems have been used as testbeds for AI research for decades.

SKELETON CODE:

```
# File  :
# Author:

def move(direction=None, m=None):
    # To be completed
    if direction == 'left':
        pass
    elif direction == 'right':
        pass
    else:
        raise ValueError("ERROR in move: invalid direction %s" % direction)

if __name__ == '__main__':
    direction = input() # for instance enter "right" (without quotes)
    s = input()         # for instance enter "0342 8" (without quotes)
    m = list(s)
    print(move(direction=direction, m=m))
```

You may assume that the input for `s` contains exactly one space.

Notes.

1. You can convert a string to a list of characters. (In Python a character is just a string of length 1. Python does not have a character type.) Try this:

```
s = "0342 8"
m = list(s)
print(m, type(m))
```

2. To throw an exception in Python you use the `raise` command. Python programmers would say "raise an exception" rather than "throw an exception". This is similar to OCAML. You can raise any value or object. Python has a collection of standard classes already created for exception handling. The exception class `ValueError` is used whenever there's something wrong with a value. Try this:

```
def is_even(x):
    if not isinstance(x, int):
        raise ValueError("is_even error: argument %s is not int" % x)
    return x % 2 == 0

print(is_even("spam"))
```

   Instead of letting the exception crash your program, you can catch it, do whatever you need to do, and then halt the program yourself in a graceful way. Try this:

```
def is_even(x):
    if not isinstance(x, int):
        raise ValueError("is_even error: argument %s is not int" % x)
    return x % 2 == 0

try:
    print(is_even("spam"))
except ValueError as e:
    print(e)
    print("too bad ... have to stop now ...")
```

   The concept of exceptions in Python is very similar to the concept of exceptions in C++. Review your C++ notes/books on exceptions if necessary.

3. In the above, the function call `isinstance(x, int)` returns `True` exactly if the type of value `x` refers to is an integer. In general you can use `isinstance` to test is an object is from a class. For instance `isinstance(x, Person)` returns `True` if `x` is a `Person` object.

Q3. The following is called the $n^2 - 1$ problem. I'm going to describe it for the case when $n = 3$.

Suppose you're given the following 3-by-3 grid of tiles numbers 1 to 8:

	0	1	2
0	1	2	3
1	4	5	6
2	7		8

Note that there's one square with no tile – the empty space in the grid above. You can move the tiles, or, analogously, you can "move" the space. So if I move the space in the north direction, the above becomes

	0	1	2
0	1	2	3
1	4		6
2	7	5	8

The goal, when given something similar to the above, is to find a sequence of moves so that the tiles are moved one at a time (or analogously, the space is moved), until you get

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	

Of course one also asks a similar question except that there are $n^2 - 1$ tiles laid out in an n-by-n grid. The goal for this question is just to make a single move.

Write a function

```
move2(direction, m, target=' ')
```

such that it moves the string `target` in `m`, a 2-dimensional array, according to the given direction. For instance if direction is `'N'` (i.e. north) and

```
m = [['1', '2', '3'],
     ['4', '5', '6'],
     ['7', ' ', '8']]
```

calling

```
move2('N', m)
```

will move the `' '` in `m` in the north direction, swapping its place with the string northward of `' '`, resulting in this `m`:

```
m = [['1', '2', '3'],
     ['4', ' ', '6'],
     ['7', '5', '8']]
```

On calling

```
move2('W', m)
```

with the original `m`

```
m = [['1', '2', '3'],
     ['4', '5', '6'],
     ['7', ' ', '8']]
```

will change `m` to this:

```
m = [['1', '2', '3'],
     ['4', '5', '6'],
     [' ', '7', '8']]
```

Note that in certain cases, the `' '` can have fewer directions of motion. For instance, if you call

```
move2('S', m)
```

with the original `m`,

```
m = [['1', '2', '3'],
     ['4', '5', '6'],
     ['7', ' ', '8']]
```

the `m` is unchanged since `' '` cannot move south.

There are four possible values for directions: `'N'` (for north), `'S'` (for south), `'E'` (for east), and `'W'` (for west),

Besides modifying the `m` that is passed into the function `move2`, your `move2` function must also return `m`.

```
def move2(direction, m, target=' '):
    # To be complete
    return m
```

SKELETON CODE:

```
# File  :
# Author:

def move2(direction, m, target=' '):
    # To be completed
    pass

if __name__ == '__main__':
    direction = input()     # for instance enter "N" (w/o quotes)
    s = input()             # for instance enter "1,2,3,4,5,6,7, ,8" (w/o
                            # quotes)
    n = input()             # for instance enter 3
    m = matrix(n, s)
    target = input()        # for instance enter " " (w/o quotes)
                            # Enter "" (w/o quotes) for default case.

    if target == '':
        print(move2(direction=direction, m=m))
    else:
        print(move2(direction=direction, m=m, target=target))
```

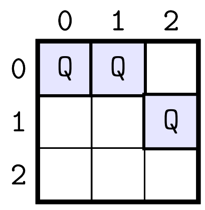
You may assume that the input for `s` has the right form, i.e., it has length n^2 and has exactly one space and n is the value entered for variable `n`.

Q4. The following is a standard problem in AI called the n-queens problem.

The following 2-dimensional array:

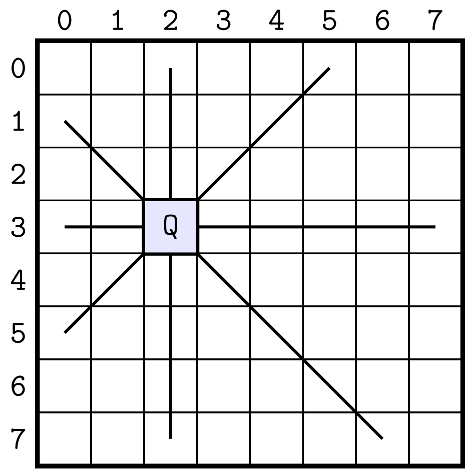
```
[['Q', 'Q', ' '],
 [' ', ' ', 'Q'],
 [' ', ' ', ' ']]
```

when viewed as a 3-by-3 chessboard looks like this

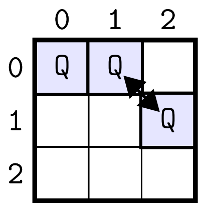


It has three queens, one in each column.

Note that in the standard 8-by-8 chessboard, queens can capture a piece on the board if that piece is horizontally or vertically or diagonally in clear sight (i.e., not obstructed) from the queen. In the following, there's a queen at $(3, 2)$ and a piece on a square on the lines shown can be captured by the queen:

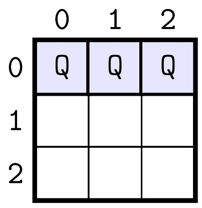


Therefore, going back to our first 3-by-3 example, the following is a pair of attacking queens, i.e., they are attacking and can capture each other:

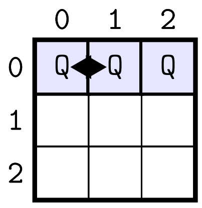
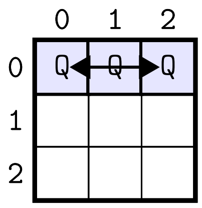
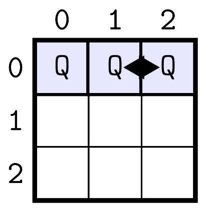


The queen in the first column and the queen in the third column do *not* form an attacking pair.

Note that when considering whether a pair of queens are attacking each other, you should just look at that pair of queens – you ignore the pieces in between. In other words, do not consider the more complex case of other queens blocking the pair. So for instance something like this:



contains 3 attacking pairs:

The goal of the n-queens problem is to find a placement of n queens in an n-by-n chessboard so as to minimize the number of attacking pairs (of course the best is to have 0 attacking pairs.) The goal of this question is just to compute the number of pairs of attacking queens.

Write a function

```
attacking_pairs(m)
```

that returns the number of attacking pairs for the given 2-dimensional array `m`.

Skeleton code:

```
# File  :
# Author:

# matrix function from earlier question here

# attacking_pairs function here

if __name__ == '__main__':
    # For this:
    # +-+-+-+
    # |Q|Q| |
    # +-+-+-+
    # | | |Q|
    # +-+-+-+
    # | | | |
    # +-+-+-+
    s = input()      # enter "Q,Q, , , ,Q, , , " (without double-quotes)
    n = int(input()) # enter 3
    m = matrix(n, s) # m is a 3-by-3 2D array
    print(attacking_pairs(m))
```

You may assume that the argument `m` passed in is correct: There is exactly one `'Q'` for each column and any non-`'Q'` value in the matrix must be a space and the matrix `m` is a square.

NOTES.

1. Python has an absolute function. Try this:

```
print(abs(42))
print(abs(-42))
print(abs(3.14159))
print(abs(-3.14159))
```

Q5. This and the last two questions are related to probability theory which is extremely important when it comes to probabilistic/heuristic AI algorithms.

Write a function `chars()` such that when the function is called with a URL, it performs character analysis on the web page at the given URL. Specifically, when you call

```
chars("http://news.yahoo.com")
```

the function analyzes and prints all the characters 0-9, a-z, A-Z that appears on the web page with frequencies and probabilities of occurrence. For instance if the web page looks like this: `Abc,,,,,abc?????` then the program prints the following:

```
A 1 0.16666666666666666
a 1 0.16666666666666666
b 2 0.33333333333333331
c 2 0.33333333333333331

b 2 0.33333333333333331
c 2 0.33333333333333331
A 1 0.16666666666666666
a 1 0.16666666666666666
```

There are two sections in the printout, separated by a blank line. First, let me talk about the first section of the output:

```
A 1 0.16666666666666666
a 1 0.16666666666666666
b 2 0.33333333333333331
c 2 0.33333333333333331
```

The characters are printed according to the order prescribed by their ASCII code. Note also that the comma (i.e. `','`) and the question mark (i.e. `'?'`) are not counted. Therefore the total number of characters counted is 6. The character `'A'` appears once, therefore the probability is 1/6 which is 0.16666666666666666. This explains the first line of output. Note that each row, the character, the frequency, and the probability are separated by a single space. Also, note that characters that do not appear are not reported. For instance the character `B` is not reported.

The second section of the output:

```
b 2 0.33333333333333331
c 2 0.33333333333333331
A 1 0.16666666666666666
a 1 0.16666666666666666
```

is the same as the output of the first section except that it is sorted by the probabilities in reverse order, i.e., the higher probabilities appear earlier. ../a01q08/question/main.tex Your program gets a URL from the user and then call `chars()` with the URL. Here's a test case.

(Parts of the output are not shown.)

```
http://www.gutenberg.org/files/1661/1661-h/1661-h.htm
0 2870 0.0059789175451
1 4147 0.00863922336569
2 12112 0.0252322819883
... snipped ...
x 571 0.00118953376943
y 9281 0.0193346110579
z 165 0.00034373567768

e 54728 0.11401191617
t 39446 0.0821757426774
a 35400 0.0737469272114
... snipped ...
Q 21 4.37481771593e-05
X 10 2.08324653139e-05
Z 2 4.16649306279e-06
```

It's not too surprising that, for a sample space that is large enough (the above webpage has 647817 characters), you would expect `e` to be the most frequently occurring character since `e` is the most commonly used character in the English language. (The URL points to *The Adventures of Sherlock Holmes*.)

NOTES.

1. There are many ways to download a webpage using python. Try this:
   ```
   import requests
   s = requests.get('http://news.yahoo.com').text
   print(s)
   ```
2. You must use a dictionary for the frequency count of a character. In other words if `a` appears 100 times, then put the key-value pair `('a', 100)` into your frequency count dictionary.

Q6. The following is a simple exercise of computations on lists that is used in genetic AI algorithms.

Write a crossover function, called `crossover`, such that if `xs` and `ys` are lists (if the first two inputs are lists) or strings (if the first two are strings), the following function call:

```
crossover(xs, ys, index0, index1)
```

returns two lists or strings `us`, `vs` such that `us` is a copy of `xs` and `vs` is a copy of `ys` except that at index positions `i = index0, index0+1, ..., index1-1`, the values `xs[i]` and `ys[i]` are swapped. For instance suppose

```
xs = 'abcdefghi'
ys = 'ABCDEFGHI'
```

and we call

```
us, vs = crossover(xs, ys, 2, 5)
```

then `us` is `'abCDEfghi'` and `vs` is `'ABcdeFGHI'`.

If

```
xs = [5,3,1,2,3,5]
ys = [5,2,1,4,6,3]
```

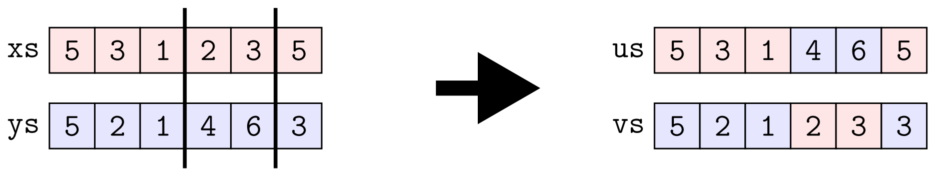
and we call

```
us, vs = crossover(xs, ys, 3, 5)
```

then `us` is `[5,3,1,4,6,5]` and `vs` is `[5,2,1,2,3,3]`.

You can think of `xs` and `ys` as chromosomes from two parents and `us`, `vs` as children chromosomes. `us` inherits all the chromosomes from the first parent except for some which have crossed over from the second parent. Likewise for the second child `vs`.

Diagrammatically, the behavior of the second example above looks like this:



Note that your crossover function must work with strings and arrays.

The above crossover is said to be a 2-point crossover – there are two cuts. In general you can have 3-point crossover, 4-point crossover, etc.

SKELETON CODE:

```
# File  :
# Author:

def crossover(xs=None, ys=None, index0=0, index1=0):
    # To be completed
    if isinstance(xs, str) and isinstance(ys, str):
        pass
    elif isinstance(xs, list) and isinstance(ys, list):
        pass
    else:
        raise ValueError("ERROR in crossover: xs=%s, ys=%s" % (xs, ys))


if __name__ == '__main__':
    i = int(input())       # enter 0 for string xs and 1 for int list xs

    if i == 0:
        xs = input()       # for instance enter "abcdefghi" (w/o quotes)
        ys = input()       # for instance enter "ABCDEFGHI" (w/o quotes)
    else:
        xs = input()               # for instance enter "5,3,1,2,6,5"
        xs = xs.split(",")         # xs becomes ['5','3','1','2','6','5']
        xs = [int(x) for x in xs] # xs becomes [5,3,1,2,6,5]

        ys = input()               # for instance enter "5,2,1,4,3,3"
        ys = ys.split(",")         # ys becomes ['5','2','1','4','3','3']
        ys = [int(y) for y in ys] # ys becomes [5,2,1,4,3,3]

    index0 = int(input())
    index1 = int(input())
    us, vs = crossover(xs, ys, index0, index1)
    print(us)
    print(vs)
```

Notes.

1. In C++, you can have two functions with the same name but with different signature because in C++ function calls are determined by function name and the signature and not just the function name. That's function overloading. Right?

   ```
   int f(int x)
   {
        return x + 1;
   }
   double f(double x)
   {
        return x + 2.0;
   }
   ```

   (The same idea applies to method overloading.) Python does not support function overloading. If you do this:

   ```
   def f(x):
       return x + 1

   def f(x):
       return x + 2.0
   ```

   then the second function overwrites the first (the first function disappears). This is similar to OCAML. To achieve the above C++ function overloading in Python, you perform type checking within a single function like this:

   ```
   def f(x):
       if isinstance(x, int):
           return x + 1
       elif isinstance(x, float):
           return x + 2.0
   ```

2. Here are some types in Python:

   a) `int`: integer type
   b) `bool`: boolean type
   c) `float`: floating point type
   d) `str`: string type
   e) `list`: list type
   f) `tuple`: tuple type
   g) `dict`: dictionary type

   You can easily find more information with google.

3. You can return two values (or more) in Python. Try this:

   ```
   def f(x, y):
       return x + 1, y + 2
   ```

```
a, b = f(2, 3)
print(a, b)
```

Technically speaking you're still returning one value: the tuple value `(3, 5)` was returned. The statement `(a, b) = (3, 5)` then assigns `3` to `a` and `5` to `b`. This last step is called unpacking.

4. Actually, without using the `if` statement, it's possible to write one block of code that works for both lists and strings.

Q7. Recall that the `range` function works as follows:

```
list(range(5))         returns  [0, 1, 2, 3, 4]
list(range(-2, 5))     returns  [-2, -1, 0, 1, 2, 3, 4]
list(range(-2, 5, 3))  returns  [-2, 1, 4]
```

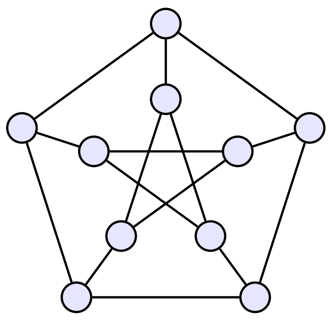
Write a `floatrange` function that does the same but for floats. For instance

```
floatrange(5)            returns  [0.0, 1.0, 2.0, 3.0, 4.0]
floatrange(2, 5)         returns  [2.0, 3.0, 4.0]
floatrange(1, 1)         returns  []
floatrange(1, 2, 1)      returns  [1.0]
floatrange(1, 2, 0.5)    returns  [1.0, 1.5]
floatrange(1, 2, 1.5)    returns  [1.0]
floatrange(-2, 2, 0.5)   returns  [-2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5]
```

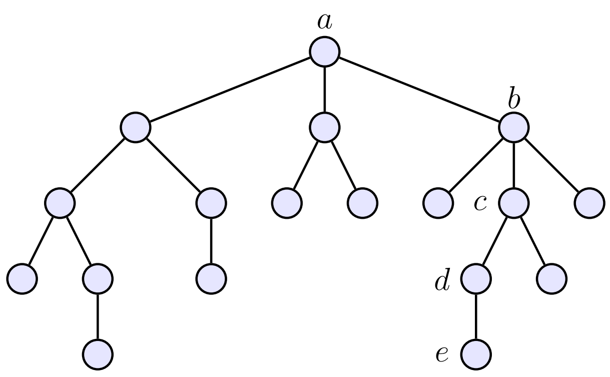
SKELETON CODE:

```
File  :
Author:

def floatrange(a, b=None, c=1):
    # To be completed
    pass

if __name__ == '__main__':
    a = float(input())
    b = input() # enter "" (w/o quotes) for default b and c
    c = input() # enter "" (w/o quotes) for default c

    if b == '':
        print(floatrange(a))
    else:
        b = float(b)
        if c == '':
            print(floatrange(a, b))
        else:
            c = float(c)
            print(floatrange(a, b, c))
```

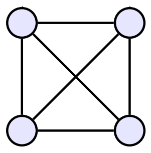
Q8. Recall that graphs (or undirected graphs) are just dots and lines. Here's one – it's the famous Petersen graph:



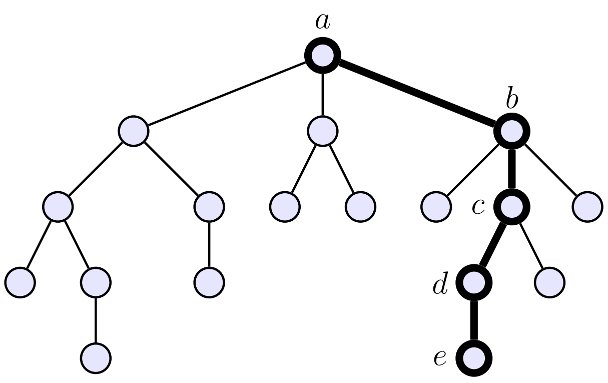
Formally the dots are called nodes or vertices and the lines are called edges. A graph that does not have "cycles" (cycle = a sequence of edges starting and ending at the same node) is called a tree. Here's a tree:



However this is not a tree:



A path in a graph is just a sequence of edges. For instance in the tree above you have this path that goes through a, b, c, d, e:



Sometimes one would say the path is a, b, c, d, e (i.e., listing the nodes that the path goes through) and sometimes one would say the path is $(a, b), (b, c), (c, d), (d, e)$ (i.e., listing the edges that the path contains).

Write a function `f` that takes a list of nodes (as strings) and return a list of edges where each edge is a list of two nodes. For instance

```
f(["a", "b", "c", "d", "e"])
```

returns

```
[["a", "b"], ["b","c"], ["c", "d"], ["d", "e"]]
```

Write another function `g` that does the opposite, i.e.,

```
g([["a", "b"], ["b","c"], ["c", "d"], ["d", "e"]])
```

returns

```
["a", "b", "c", "d", "e"]
```

SKELETON CODE:

```
# File  :
# Author:

# matrix function here

# f function here

# g function here

if __name__ == '__main__':
    option = int(input())
    if option == 1:       # test f
        s = input()       # enter "a,b,c,d,e" (without double-quotes)
        xs = s.split(",") # xs is ["a", "b", "c", "d", "e"]
        print(f(xs))
    else: # test g
        s = input()       # enter "a,b,b,c,c,d,d,e" (without double-quotes)
        xs = matrix(2, s) # xs is [["a","b"],["b","c"],["c", "d"],["d", "e"]]
        print(g(xs))
```

The `matrix` function refers to the `matrix` function from an earlier question.

Q9. Mergesort

Complete the mergesort function.

SKELETON CODE:

```
# File  :
# Author:

def mergesort(xs, verbose=False):

    # Put merge function here

    def mergesort_(xs, start, end, t, verbose=False):

        # To be completed

        if verbose:
            print(xs)

    t = [] # temporary array
    n = len(xs)
    mergesort_(xs, 0, n, t, verbose)

if __name__ == '__main__':
    s = input()             # for instance enter "51,32,1,23,47" (w/o
                            # quotes) to sort [51, 32, 1, 23, 47]
    xs = [int(_) for _ in s.split(',') if _ != '']
    mergesort(xs, verbose=True)
    print(xs)
```

Q10. [OPTIONAL]

Now we perform a word analysis on the web page (see previous question) with their frequencies and probabilities and write them to a file `words.txt`. This is done through a function called `words` and the parameter is the URL. In other words

```
words("http://news.yahoo.com")
```

will analyze the words on the web page at `http://news.yahoo.com`. Note that you should only analyze the words that appear on the web page. You should therefore remove HTML tags. Here is an examples of HTML tags:

```
This word is in <b>bold</b>.
```

In other words tags look like this:

```
<...>
```

should not be included in computation of frequencies and probabilities.

- Each (word, frequency, probability) triple should be on a separate line, with a space separating them. Only lowercase should be printed – uppercase should be replaced by lowercase. For simplicity strings containing the apostrophe, example `don't`, counts as a word. There must be no blank line in the text file except for the last line which contains only the newline character.
- The listing must be sorted by frequency and then the word, with the highest frequency appearing first. This means that if the word `ham` and `eggs` appear with the same frequency, then `eggs` must appear first.

For instance if the web page has only two words spam appearing 3 times and ham appearing one like this:

```
Spam! SPAM?"Spam", ...,ham!
```

then the output should be

```
spam 3 0.75
ham 1 0.25
```

Q11. [OPTIONAL]

Now we perform a link analysis. This is similar to the above (see previous questions) except that you process all the URL links and print the information to the file `urls.txt`. Here's a fragment of a web page containing a link:

```
Click <a href="http://www.nowhere.com/a/b?c=1">here</a> to nowhere.
```

In this case the URL is

```
http://www.nowhere.com/a/b?c=1
```

Note that HTML is not case sensitive. For instance you should also harvest the following link:

```
Click <a HREF="http://www.somewhere.com/a/b">here</a> to somewhere.
```

and

```
Click <A HREF="http://www.somewhere.com/a/b">here</a> to somewhere.
```

etc. Furthermore HTML also allows spaces. For instance the above is really the same as:

```
Click <a href="http://www.somewhere.com/a/b">here</a> to somewhere.
```

and

```
Click < a href = "http://www.somewhere.com/a/b">here</a> to somewhere.
```

and

```
Click < a href= " http://www.somewhere.com/a/ b " >here</a> to somewhere.
```

etc.

You should write the urls to the file `urls.txt`, one url per line. Do not repeat. The urls should be sorted.

If the link does not begin with `http://`, you must append it. The output format is similar to (a): The information in the file must be sorted by frequency and then by the URL. For instance if the web page has two links, `http://abc.com` appearing 3 times and `http://def.com` appearing once, then `urls.txt` must be

```
http://abc.com 3 .75
http://def.com 1 .25
```