COMP1001 – Problem solving in IT
Final Examination

Please read the following instructions carefully.

1. Please put away your phones and calculators.
2. If you need to go to bathroom, please raise your hand and get the approval before going.
3. You may use books, notes and any hardcopy materials, but you cannot share them with others.
4. You will be given three hours to solve eight problems. The full mark of this paper is 90.
    - Q1: 10 marks
    - Q2: 10 marks
    - Q3: 10 marks
    - Q4: 10 marks
    - Q5: 20 marks (10 marks each for (a) and (b))
    - Q6: 10 marks
    - Q7: 10 marks
    - Q8: 10 marks
5. You can only use IDLE and notepad++ for editing and running your program.
6. Unless stated otherwise, you do not need to handle exceptions in your programs.
7. Submission instructions:
    a. Name all files as "<Student ID>_<Question Number>.py". For example, "12345768d_Q1.py".
    b. At the beginning of your code, type your name and student ID. For example,
        # Name: Chan Tai Man
        # Student ID: 12345678d
    c. There is a submission link for each question under COMP1001 Exam in https://submit.comp.polyu.edu.hk. Submit your .py file for each question to the corresponding submission box or no mark will be given.
    d. After completing a question, you should submit it right away.
8. Before you leave, please MAKE SURE that you have submitted all your .py files. You will bear the sole responsibility if you fail to do so.
9. If you choose to leave before the official end time, please sign out on an attendance list.

1. (Printing a symmetric pattern) Write a program that will ask user for a symbol and a positive width in terms of the number of characters. Your program will print out a symmetric pattern like the ones below.

```
>>> main()
Please enter a symbol: *
Please enter the width, a positive integer: 30
*                             *
**                           **
***                         ***
****                       ****
*****                     *****
******                   ******
*******                 *******
********               ********
*********             *********
**********           **********
***********         ***********
************       ************
*************     *************
**************   **************
******************************
```

```
>>> main()
Please enter a symbol: $
Please enter the width, a positive integer: 31
$                             $
$$                           $$
$$$                         $$$
$$$$                       $$$$
$$$$$                     $$$$$
$$$$$$                   $$$$$$
$$$$$$$                 $$$$$$$
$$$$$$$$               $$$$$$$$
$$$$$$$$$             $$$$$$$$$
$$$$$$$$$$           $$$$$$$$$$
$$$$$$$$$$$         $$$$$$$$$$$
$$$$$$$$$$$$       $$$$$$$$$$$$
$$$$$$$$$$$$$     $$$$$$$$$$$$$
$$$$$$$$$$$$$$   $$$$$$$$$$$$$$
$$$$$$$$$$$$$$$ $$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
>>>
```

2. (Reversing a string) Write a Python program that will ask user for a string and output the reverse of the words in the string. You can keep the punctuations in the string except for the full stop at the end of the sentence. The full stop, if exist, will stay at the end of the string after the string reversal. Sample outputs below.

```
>>>

Please input a string: Here I am, send me.
The correct output is: me send am, I Here.

>>> main()

Please input a string: It's not that I'm so smart, it's just that I stay with problems longer.
The correct output is: longer problems with stay I that just it's smart, so I'm that not It's.

>>>
```

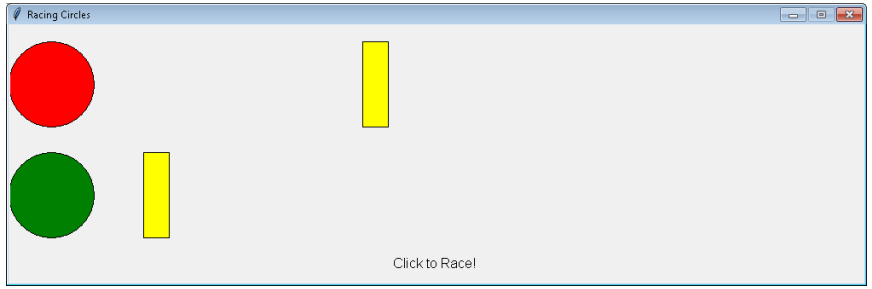
3. (Factoring a number) Write a function that accepts a positive integer and returns a list of factors of the integer. Also write a `main()` to ask use for an integer and print out the list of factors. Sample outputs below:

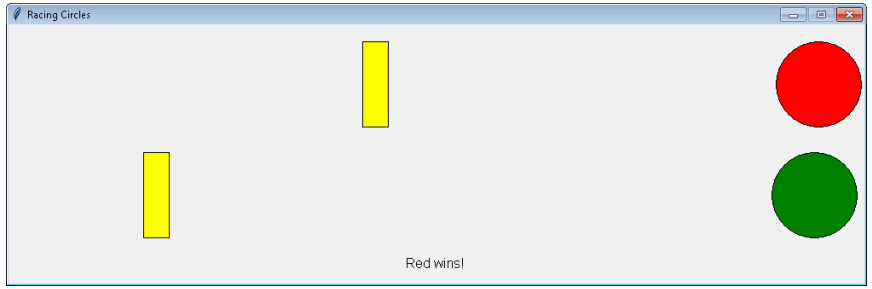
```
Please enter a positive number: 1
The factors for 1 are [1].

>>> main()

Please enter a positive number: 5
The factors for 5 are [1, 5].

>>> main()

Please enter a positive number: 24
The factors for 24 are [1, 2, 3, 4, 6, 8, 12, 24].
```

4. [A Python game] Write a Python GUI using graphics.py. There are two circles, red and green. When a user clicks the window, the two circles will move to the right, until the border of one of the circles touches the right boundary of the window. For each move (or cycle), each circle will move towards right by 0 – 4 pixels randomly. There is also one yellow hurdle along the path of each circle. When the circle touches the hurdle, it will halt its movement for 10000 cycles.

   The width of the window is 1000 pixels. The hurdle has a width of 30 pixels and a length which is the same as the diameter of the circle. The position of the hurdle is also random, lying after 50 pixels from the circle and 200 pixels before the right boundary of the window.

   When the game finishes, the program should output a message at the bottom, indicating which circle wins the race. For example,





   Note that a game winner is the circle that touches the boundary first. If both circles touch the boundary at the same time, the program should display “Draw.”

5. (Analysis of sentences) This question analyzes the length of sentences in a text file. There are two parts to this question. The first part is to retrieve sentences from a text file, and the second is to analyze the length of the sentences. You will use treehouse.txt provided in this exam to test your program.
   a. Write a function called `getSentences()` which accepts a file handler as parameter and returns a list of sentences. You may consider that a sentence is ended by either a full stop or a question mark. The title of a paragraph, such as "Readability" in treehouse.txt, considered as part of the next sentence.
   b. Write a function called `analyzeSentences()` which accepts a list of sentences as parameter and prints out the statistics of sentence lengths as shown in the sample outputs below. Each column is 25-character long and they are printed in the center. **You must use dictionary to process the length of the sentences.**

   Implement the two functions below. If you cannot make part (a) work, you may come up a proper list of strings to test part (b).

```
def main():

     def getSentences(infile):
          """ This function accepts a file handler of a text file and returns the sentences in the file as a list."""

          # To be completed by you

     def analyzeSentences(aList):
          """ This function accepts a list of sentences and prints out the statistics of the sentence length. """

          # To be completed by you


     filename = input("Please input a text file: ")
     infile = open(filename)
     sList = getSentences(infile)
     for i in sList:
          print(i)
     analyzeSentences(sList)
```

```
main()
```

Below are the first few lists of strings in `sList` and the output statistics of treehouse.txt.

```
>>>
Please input a text file: treehouse.txt
['At', 'Treehouse', 'we', 'are', 'gearing', 'up', 'to', 'launch', 'a', 'new', 'course', 'called', '‘Python', 'Basic
s’.']
['It', 'will', 'be', 'released', 'later', 'in', 'July', 'but', 'for', 'now', 'teacher', 'Kenneth', 'Love', 'talks',
'about', 'why', 'you', 'should', 'consider', 'learning', 'the', 'programming', 'language', 'named', 'after', 'Monty
', 'Python.']
['When', 'I', 'need', 'to', 'build', 'a', 'web', 'app,', 'I', 'reach', 'for', 'Python.']
['When', 'I', 'need', 'to', 'automate', 'some', 'small', 'task', 'on', 'my', 'system,', 'I', 'reach', 'for', 'Pytho
n.']
['When', 'I', 'want', 'to', 'find', 'the', 'most', 'common', 'colors', 'in', 'an', 'image,', 'I', 'reach', 'for', '
Python.']
['When', 'I…OK,', 'I', 'think', 'you', 'get', 'the', 'picture.']
['Basically,', 'when', 'I', 'need', 'to', 'code', 'something', 'and', 'the', 'language', 'doesn’t', 'matter,', 'I',
'use', 'Python.']
```

Length of sentence	Frequency in counts	Frequency in %
4	1	3.57
6	1	3.57
8	1	3.57
12	2	7.14
14	1	3.57
15	5	17.86
16	3	10.71
19	1	3.57
20	1	3.57
21	1	3.57
23	2	7.14
27	1	3.57
29	1	3.57
33	1	3.57
39	1	3.57
40	1	3.57
41	1	3.57
42	1	3.57
45	1	3.57
50	1	3.57

6. (Exceptions) We have discussed the problem of moving *n* piles of rocks placed at different positions of a line, and the solution is the median of the positions' numbers. Below is a function `getNumber()` which receives from user a positive integer for the total number of piles. To make it simple, the function raises an exception from class `Illegal` for every possible exceptions. The `main()` uses `getNumber()` to continuously prompt user for a valid input. Try different illegal inputs to understand how it works.

   You are asked to implement a function `getPosition(postList,i)` which asks user for the position of i^{th} rock pile. Each position can be used only for one pile, and the `postList` contains a list of all the used positions. Therefore, an exception will be raised if the inputted position is in `postList`. You also need to implement the missing code in `main()` to use `getPosition(postList,i)` to get all the positions.

```
class Illegal(Exception): # A custom exception
    pass                  # nothing executed

def getNumber():
    """This function asks user for a positive integer. It will raise Illegal for any illegal input."""

    try:
        x = eval(input("Please enter the number (>0) of piles of rocks >> "))
    except:
        raise Illegal

    if (type(x) != int) or (x <= 0):
        raise Illegal

    return x

def getPosition(posList,i):
    """This function asks user for an unselected position. It will raise Illegal for any illegal input."""

    # To be completed by you




def main():
    # get the number of rock piles
    n = None
    while (True):
        try:
            n = getNumber()
        except Illegal:
            print("getNumber(): Your input is illegal.")
        else:
            break

    if n != None:
        # Get the positions of the n piles and enter them into a list.

        # To be completed by you
```

```
        # Find a median of the inputted numbers.
        if len(rockList) == n:
            rockList = sorted(rockList)  # in case the inputted numbers are not sorted
            print("The position where all the rock piles should be moved to is ", rockList[int(len(rockList)/2)],".", sep="")

main()
```

7. (Another rabbit population problem) Let's reconsider the rabbit population problem whose solution is given by the Fibonacci numbers. Now we modify the problem by restricting that each pair of male-female rabbit can reproduce only one more male-female pair. In other words, the modified problem is
   - Start out with a male-female pair of rabbit, i.e., one pair in the first month.
   - It takes two months for each pair to be reproductive, i.e., two pairs in the third month.
   - (Modified) Each pair reproduces only one more male-female pair as soon as they become reproductive and then stop reproducing in the subsequent months.
   - How many pairs of male-female rabbits after *n* months if no rabbits die during this period?

   Write down the base cases and the inductive step in a recursive solution to this modified problem in a docstring. No code is required.

8. (The binary approach to the X→Y problem) Recall the three examples of using a binary approach to solve the X→Y problem:

   Example 1: 12 → 50
   a. 1100 → 110010
   b. The first 4 leftmost digits are the same.
   c. After a ×2 operation, 1100 becomes 11000.
   d. After a +1 operation, 11000 becomes 11001.
   e. After a ×2 operation, 11001 becomes 110010.
   f. Therefore ((12×2)+1)×2 = 50 (3 operations)

Example 2: 8 → 42

a. 1000 → 101010
b. The first 4 leftmost digits are **not** the same.
c. After a +1 operation, 1000 becomes 1001.
d. After a +1 operation, 1001 becomes 1010.
e. Now the first 4 leftmost digits are the same.
f. After a ×2 operation, 1010 becomes 10100.
g. After a +1 operation, 10100 becomes 10101.
h. After a ×2 operation, 10101 becomes 101010.
i. Therefore ((8+1+1)×2+1)×2 = 42 (5 operations)

Example 3: 10 → 35

a. 1010 → 100011
b. The first 4 leftmost digits are **not** the same.
c. After a +1 operation, 1010 becomes 1011.
d. After a +1 operation, 1011 becomes 1100.
e. After a +1 operation, 1100 becomes 1101.
f. After a +1 operation, 1101 becomes 1110.
g. After a +1 operation, 1110 becomes 1111.
h. After a +1 operation, 1111 becomes 10000.
i. After a +1 operation, 1111 becomes 10001.
j. Now the first **5** leftmost digits are the same.
k. After a ×2 operation, 10001 becomes 100010.
l. After a +1 operation, 100010 becomes 100011.
m. Therefore (10+1+1+1+1+1+1+1)×2+1 = 35 (9 operations).

Design an algorithm based on the binary approach to obtain the minimum number of operations in a docstring.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

END