CSE101: IT Workshop
Semester Examination ***Marks: 40%***
Department of CSE, IIT-BHU ***Time:*** *3 hours*

Your Roll No:	

***Instructions to the CSE101 students*:**

1. Answer all 20 questions. There are 7 pages in this document.
2. Students may bring permitted support material to the examination. The support material should not, in any way, interfere or discomfort any other student in the examination room. No support material can be shared or exchanged between the students during the examination.
3. No electronic item, except a standard watch or/and prescribed medical support, may be carried to the examination.
4. The students may bring their handwritten notebooks (of the size and style in common use at IIT, BHU) to the examination. However, loose papers and sheets are not permitted. Limited amount of printed/photocopied material may be glued to the notebook pages. Every glued item should be of size smaller than the notebook page**.**
5. Each notebook should have student's name written in ink together with a unique identifier. Each student must declare the support material they have taken to the examination by listing each item on page 1 of their exam answer book.
6. Items not complying with the conditions for the permitted support material in the examination, including any undeclared notebook, will be removed from the student's desk and may be reported to the institute authorities for disciplinary actions.
7. A good answer to a question in this examination only requires about half a page of writing. **Start writing answer to each question on a new page.** Correctly label each answer with its question number.
8. **Do not write answers for more than one question on a page.**
9. In this question paper, *Prompt$* at the start of a line denotes the command prompt displayed by a Bash shell. Likewise, >>>> at the start of a line denotes a prompt from a Python interpreter.
10. Each question will be graded as: A, B, C, D or F. General guidelines for grading are:
    i. *A (2 mark): The answer includes all the major points related to the question and the answer is free from incorrect and unrealted information. Where the answer includes a program code, the code should be correct, concise, easy to understand and efficient.*
    ii. *B (1.5 marks): The answer may have missed some issue related to the question. Or, it includes unnecessary/unrelated information or a part of the answer is incorrect. Typically, an incorrect part in the answer will be marked by demerit symbol "X". Smaller errors are marked by demerit symbol "?". Answers with two "?" will attract same grade penalty as an "X".*
    iii. *C (1 mark): Answers receiving two "X" or equivalent demerit will be graded as C.*
    iv. *D (0.5 marks): A poor answer to a quesion but with some worth will receive a grade of D.*
    v. *F (0 marks): All poorly answered questions and unanswered questions will be graded F.*
11. Do not include any statement or code in your answer unless it has been asked. You may lose marks for incorrect but unnecessary statements in answers without a prospect of any benefit.

Unix/Linux Shell Programming

Question 1: Consider the bash script *script* listed below:

```
!#/bin/bash
if ( "$1" = "Uttar Pradesh" )
then
      echo 'IIT BHU is the best institute in the $1'
else
      echo "IIT BHU is not located in $1"
fi
```

What output will be printed when for the bash commands:

```
prompt$./script 'Uttar Pradesh'
prompt$./script "Uttar" "Pradesh"
```

Question 2:
Suppose your login name is `namo` and your current directory is your home directory. You want to copy all `.py` files in directory `/home/namo/varanasi/` into directory `/home/namo/kashi/htbin`. What single command, using the relative pathnames, will copy the files without a need to change the current working directory.

Question 3:
Write a bash script that takes one command line argument: this argument will be the proposed login name for a user being given a new account on the system. Your script must check the proposed name against the names of the current users listed on `/etc/passwd` file and report if the proposed name clashes with an existing name.

Question 4:
Suppose a file named `assignment.odt` is located in your home directory. The file has permission setting 664. Your system administrator has set permission bits on your home directory as 742. Who can delete file called `assignment.odt`?

Question 5:
Write an egrep regular expression to match the following patterns of the date-and-time printed by command `date` in the two examples below: (You may use the following two set variables:

```
weekDays="Sun|Mon|Tue|Wed|Thu|Fri|Sat"
months="Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec" )
```

Examples of data-and-time as printed by command `date`:

```
Fri Jan 14 23:28:56 IST 2016
Thu Feb  4 10:13:56 IST 2016
```

(Your answer should reject strings `Fri Jan 14 33:28:56 IST 2016` and `Fri Jan 44 23:28:56 IST 2016`. Reasons for rejection: the valid first digits in a time (hour) value are space, 1, or 2. Similarly, a two-digit date value cannot have 4 as its first digit.)

Question 6:
Write a script that takes a directory path as its command line argument. It then prints three numbers giving number of files with extensions `.c`, `.py` and `.bash` in the directory. An example output may be as follows:

```
*.bash : 0
*.c : 5
*.py : 12
```

Question 7:
Write a `find` command (with condition/option `exec`) to delete all executable non-directory files under your home directory that have not been accessed for 7 days or more. (You may wish to consider command options: `-atime`, `-type`, `-name`, `-newer`, `-perm`, `-executable`)

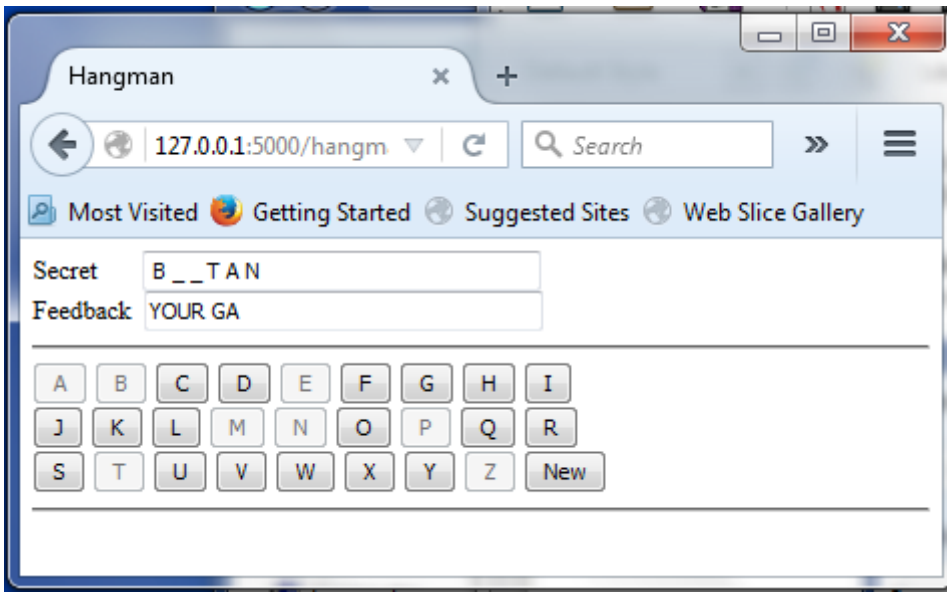
Python, Python libraries and Flask framework

Context for Questions 08 to 15: Game of hangman

A game called Hangman is described. The game is accessed over the web through a browser. Questions in this section relate to a proposed implementation of the game in Python using Flask framework.

A player wanting to play the game accesses URL `localhost:5000/hangman`. In response, the Python application implemented on a Flask based server, picks a word randomly from a list called *wordList*. The player has a maximum limit of 17 penalty points to guess all letters in the chosen word, from now on we will refer to this word as *secretWord*. Each incorrectly guessed letter in secretWord, reduces the remaining penalty quota of the player. Most incorrect guesses have one point penalty. However, the letters in list *doublePenaltyLetters* have 2 point penalty. Twelve letters with double point penalty are: ABEIMNOPRSTU.

To build a convenient interface for the players, the web display is based on an html *form* organised in three parts as in the picture below (The secret word is BHUTAN).



The first part is a text box in which the discovered parts of *secretWord* is visible. Initially, the player has not guessed any letter of the word. This is indicated, in the readonly text box labelled *Secret*, by all letter positions marked by characters "_ " (underscore characters separated by a space). As the player guesses the letters, the correctly guessed letter positions are replaced by the correct letters. Wrongly guessed letters reduce the remaining penalty limit of the player as described above.

Below this is *feedback* box carrying the initial part (prefix) of the string: `YOUR GAME IS LOST`. The number of letters (including spaces) visible in this text box, equal the penalty points lost by the player. When the full message is displayed, the player has lost the game. If the player guesses all letters of *secretWord*, the displayed message is: `YOU WON THE GAME`.

The final visible part at the bottom has three rows of 9 buttons each – one button for each letter in the English alphabet together with a button labelled *New*. The *New* button start a new game. All buttons are initially *enabled* (default). All enabled buttons are *submit* buttons. A click on a letter (alphabet) button marks the player's guess of the letter in *secretWord*. The clicked buttons are shown, in the later displays, as *disabled*. These buttons do not submit forms any more. At the end of the game, only *New* button is not disabled.

In answering the questions, make sure that identifiers (variables as well as functions) are used in a consistent fashion over the question set.

Question 08:
Identify all variables that would be needed to properly render the *html/jinja2 template* in preparation for its display in a browser.

Write Python code to assign initial values to these variables. This code is run when the game is first accessed by a player using URL: `localhost:5000/hangman`. It is also run when button *new* is clicked by the player to start a new game.

Question 09:
Consider a html/jinja2 template that would be needed to render the game status on the browser. This is a large document, therefore the questions here are related to only the specific parts in this template.

Write a complete tag line `<form>`, with all necessary attributes (and values) for an implementation of this game.

Question 10:
Write a `render_template` call showing all keyword parameters that would create the response html document ready for return to the requesting browser. Also write the related import statements needed in the module.

Question 11:
Provide html/jinja2 template code that would (a) correctly display the text boxes labelled as *Secret* and *Feedback*, and (b) embed the hidden values necessary to extract the status of the game when form is submitted back to the server application (that is, to a function(s) implementing the game of hangman).

You need only write the part of the template code that directly relates to these form elements. Do not write any part of the template enclosing these lines.

Question 12:
Write html/jinja2 template code to display all nine buttons in the middle row of the buttons.

Your code should correctly display the buttons in their correct modes: the letters that have already been selected/guessed by the player are rendered as disabled buttons. The buttons for not-yet-selected letters are the submit buttons.

Question 13:
Write Python/Flask code to extract the values from the submitted form and assign these extracted values to the correct variables in Python program in preparation for servicing the received request from the player's browser. (A form is received, at the hangman server/application, when the player clicks a submit button in the browser while playing the game.)

Question 14:
Write Python code using the variables set in your answer to Question 13, to determine the classification of the letter selected by the player. The three classifications of interest are: (a) the selected letter is a letter in secretWord, (b) the selected letter causes a single-point penalty, or (c) the selected letter causes a two-point penalty.

Question 15:
Write Python code, using the variables used in the answers to Questions 13 and 14, to compute the new values for variables necessary to call `render_template` method (For example, refer to Question 10).

Python Programming

Context for Questions 16 to 20: Game of Tic-Tac-Toe

The game of Tic-Tac-Toe (also commonly known as game of Naughts and Crosses) has a 3*3 grid (also called, board) of 9 places. A (human) player takes turns with a computer to play the game – the human player fills a place with letter O, then the computer fills a place with letter X. They continue taking turns till one of the following event occurs.

1. The player wins the game
2. The computer wins the game, or
3. All places are filled. In this case the game is a draw.

You will define Python functions to implement this game that will begin a series of 100 games to be played between a player and a computer.

- The player will get to make the first move in the first game of the series.
- The winner of a game gets to play the game immediately following the win.
- If a game is drawn, the participant who played second in the drawn game, gets to make the first move in the new game.

The places in the grid are identified by two letters specifying row and column of the place. Three row identifiers are letters T, M, and L respectively corresponding to the top, middle and bottom rows. The columns are identified by letters L for left, M for middle, and R for right. The player can choose to specify row and column letters in either order. That is, TR and RT refer to the same place at the right top of the board.

The computer uses the the following rules in determining its move.

1. If place MM is free, computer inserts X there. Otherwise,
2. The computer tries to find a place where it can fill an X to win the game. If such a place exists, the computer chooses the place. Otherwise,
3. The computer tries to find a place where (human) player can put O to win the game. If such a place exists, the computer chooses the place. Otherwise,
4. The computer seeks a corner place to fill. Otherwise,
5. The computer chooses an unfilled place with an X.

The rules for determining the game winner are:
The participant (a player or a computer) who places their three letters in a row, a column or a diagnol first wins the game. The game ends as soon as a winner is found.

Help generated doc strings for an implementation of the game is provided on the following page. Code for functions `allFill` and `showBoard` are also given.

```
Help on module ticTacToe:

NAME
    ticTacToe

FUNCTIONS
    allFill(board)
        Returns True if no more move is possible.

    compMove(board)
        Determines and returns the next move for computer.

        The move is returned as a tuple (column, row).
        Does not change the board

    getMove(board)
        Reads players choice of move.

        Continues to prompt player till valid input is provided.
        The move is returned as a tuple (column, row).
        Does not change the board.

    playGame(board, turn)
        Plays a game; plays a single move before a recursive call.

        It first determines if the game is not over yet.
        One move is made for the participant using symbol turn.
        The move is entered into the board. A recursive call is
        made to function playGame to continue the game.

        If the game is over, the winner's symbol is returned.

    showBoard(board)
        Prints the board for the player.

    startSeries()
        As described in the context statement.

        Symbol "-" in a place denotes the place is unoccupied.

    symbolWins(board, symbol)
        Returns True if player using symbol has won in the board.
```

```
def showBoard(board):
    print board[0]
    print board[1]
    print board[2]
```

```
def allFill(board):
    for r in range(0,3):
        for c in range(0,3):
            if board[r][c]=="-":
                return False
    return True
```

Question 16:
Define function `symbolWins` that takes as argument the current `board` and a player's `symbol`. If the player using `symbol` has won the game in the `board` it returns `True` otherwise returns `False`.

Question 17:
Function `getMove` defined below does not correctly read the player's moves typed on the keyboard. Read and understand the code to find the mistakes in the code. Then, write your corrected version.

```
def getMove(board):

    showBoard(board)
    mv = raw_input("Your move ([LMR][TMB])|([TMB][LMR]):")

    if len(mv) == 2 or not (mv[0] in "LMRTB" and mv[1] in "LMTRB"):
        print "Bad choice"
        return getMove(board)

    if mv in ["LL" "LR", "RL", "RR", "TT", "TB", "BT", "BB"]:
        print "Bad choice"
        return getMove(board)

    if "T" in mv:
        r = 0
    if "M" in mv:
        r = 1
    if "B" in mv:
        r = 2

    if "L" in mv:
        c = 0
    if "M" in mv:
        c = 1
    if "R" in mv:
        c = 2

    return (c, r)
```

Question 18:
Define function `startSeries` that calls function `playGame` repeatedly till the series ends. The function `playGame` uses a string argument symbol with value "`X`" or "`O`" to determine who will play the next move in the game. Your definition of function `startSeries` must correctly provide these values for call to function `playGame`. The function also determines the winner of the series. Return value from function `playGame` provides the result of each completed game.

Question 19:
Define recursive function `playGame` that first determines, based on the argument board, if the game continue to the next move. If the game has ended it displays the result and returns the winning symbol or "`-`" (for a drawn game) to the calling function.

Question 20:
Define function `compMove` to correctly implement the rules that determine a move by the computer. The function returns a tuple (column, row) giving the place selected by the computer as its next move. However, the place is not filled by the function to match the behaviour of function `getMove`.