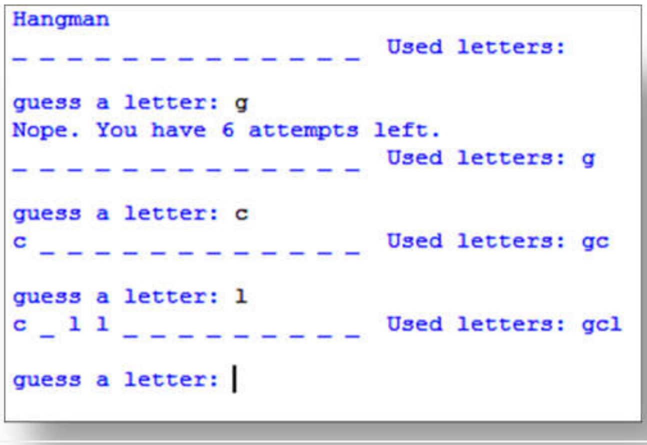
MY FIRST GAME–HAMGMAN

Use your knowledge about decisions, repetitions and strings to design and write a Hangman game. Pick a word - save it as a secret word in the program but don't show it on the screen. Instead, print as many dashes as the letters of your word. Ask the user to enter a letter and add it to the list of used letters. If the letter matches any spaces in your word, exchange the dashes with the letter. If the letter does not match, the number of attempts decreases with one. You have 7 attempts to guess the word. The screen should look like the screenshot below:

For guidance study and follow the pseudo code and check the marking scheme below. Submit to Moodle.



```
Hangman
_ _ _ _ _ _ _ _ _ _ _ _ _   Used letters:

guess a letter: g
Nope. You have 6 attempts left.
                            Used letters: g
_ _ _ _ _ _ _ _ _ _ _ _ _

guess a letter: c
c _ _ _ _ _ _ _ _ _ _ _ _   Used letters: gc

guess a letter: l
c _ l l _ _ _ _ _ _ _ _ _   Used letters: gcl

guess a letter: |
```

	MARKING SCHEME	**NAME:_________________________________**				
COM	• Header contains name, date, description **½** mark for each • Descriptive variable names are used • Program is formatted to improve readability • Comments are added in the main sections (on the side)	**0**	**0.5**	**1**	**1.5**	**2**
APP	• Prints an intro message starting the game and show the main rules ❶ • Prints dashes for the missing letters **<u>at the beginning</u>** of the game and after each turn ❶ • Prompt the user to input a letter after the dashes are printed after each turn ❶ • Prints the guessed letters in their correct places (as shown) ❷ • Prints the **used letters <u>on the side</u>** (as shown) ❶ • Prints a message after each unsuccessful attempt showing the **number of trials** remaining (as shown) ❶ • Print the correct message at the end – when you win and when you lose ❶	**0 2**	**4**	**5**	**6**	**7 8**
TH	• Student is able to troubleshoot their code without teacher's or classmates' support ❷ • Program is not case sensitive & user-friendly (good prompts) ❶ • Program is error-proofed – only letter inputs, only one at a time ❷ • Only the first guess counts when guessing the same letter twice ❶ • A visual representation of a hangman is included in addition to the regular printing. Print multiple lines string with all the dashes and symbols to represent a hangman in different stages (keep the printings small) ❶ • Secret word is randomly chosen from a selection with specific topic printed at the beginning, like "FOOD" or "ANIMAL" or inputted by the user ❶	**0 2**	**4**	**5**	**6**	**7 8**

Hangman Pseudo code

The Pseudo code starts by writing the Input and Output sections first:

Input to the problem:

- **secret word (preset)**
- **my_let (**input letters one by one by the user)

Output of the problem:

- secret word printed (letter by letter)
- **used letters**
- prompt for guessing
- feedback about each guess - correct or incorrect
- game outcome use **missing letters**to check when it is over

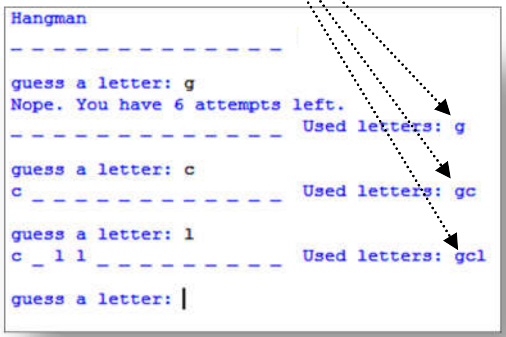
Next write the Process section showing all the steps:

Process (order of instructions):

```
Store the secretWord into a variable
Set usedLetters to an empty string
Set allowed attempts to 7
Print "Hangman" at the top and on the next line the secret word with underscores (" _ ")→use len()
Start a loop while allowed attempts are more than zero
    Set a variable missingLetters (as the length of your secret word) to keep track of how many letters from the secret word are still not guessed.
    Prompt user for an input → a new letter and store it in a variable newLetter

    Error-proof the input - check for 1 letter input at a time →use len() and
                          - check for a letter input (use String method)  and
    for full marks only
    skip till the rest of it works
                          - check if the letter is not already guessed (inside the usedLetters string)

    If the new letter is not in the secret word,
        Reduce allowed attempts by 1
        Print the number of the remaining attempts
    Add the new letter to the string of usedLetters
    Start a for loop going though each letter from the secretWord
        Check if each letter has been guessed→is it inside the usedLetters string
        If yes - print the current letter (keep printing on the same line→use comma)
        Decrease the value of missingLetters by one for each printed letter
        If not – print underscore " _ " (keep printing on the same line)
    Print the usedLetters on the side of the word. Put a label "Used letters: "before it.
    If the secret word is guessed (no letters are missing),
        Print "You won!" message and exit (break) the loop.
If no more attempts are allowed,
    Print the secretWord
```



```
Hangman
_ _ _ _ _ _ _ _ _ _ _ _ _ _

guess a letter: g
Nope. You have 6 attempts left.
                             Used letters: g
_ _ _ _ _ _ _ _ _ _ _ _ _ _

guess a letter: c
c _ _ _ _ _ _ _ _ _ _ _ _ _  Used letters: gc

guess a letter: l
c _ l l _ _ _ _ _ _ _ _ _ _  Used letters: gcl

guess a letter: |
```

Hint: Use in and not in operators to check whether a letter is present/not present in a word

operator	example	result
in	‘e’in‘team’	**True**
not in	‘s’not in‘spirit’	**False**