**Instructions**

1. Create a new GitHub repository called Week11-Hangman, then clone it to your computer.
2. Inside your local Week11-Hangman folder, make files called game.js, letter.js, main.js and word.js.
3. With the files from Step 2, you'll put together a console-based hangman app that incorporates objects, user-input and constructors in its solution.
   * Both letter.js and word.js should be constructor files:
   * word.js should contain all of the methods which will check the letters guessed versus the random word selected.
   * letter.js should control whether or not a letter appears as a "\_" or as itself on-screen.
   * Your game.js file will randomly select a word for the player.
   * main.js will contain the logic of your app. Running it in Terminal/Bash will start the game.
   * The app should end when a player guesses the correct word or runs out of guesses.

**Tips**

1. Program the game one piece at a time! Break the program down into fragments rather than chipping away little by little at giant, abstract problem.
   1. Focus on one part of the smaller problem and get some code working.
   2. Rinse and repeat (and remember, console.log is your friend).
2. Try your best to deliver a *working/playable* game by the end of the deadline.
   1. Submit your app whether you finish or not; if you don't show us anything then we won't be able to help you with whatever programming issues you encountered this week.
3. **Working game > pretty dud**. We're focusing on game mechanics, not just on the look and feel of your app.

1) You can use letter.js to store a constructor function that takes in a letter. The letter constructor can have a property to check if the letter passed in is supposed to appear and a method that, based on the property for the letter appearing, shows the letter or shows an underscore.

2) The letter.js constructor can be used inside the word.js file.

The word.js constructor can have the following properties:

word: equivalent to the word passed in.

lets: array of letter objects

found: boolean

getLets: method to construct letter objects from the word passed in and push the letter objects to the lets array.

didWeFindTheWord: method that sets this.found in the word object to true or false if all letter objects have a true value in their appear property.

checkIfLetterFound: method that checks to see if the letter that was guessed matches the letter(s) in the word.

wordRender: render the word as string from lets objects.

3) main.js: main file to ru. requires prompt and word.js

game object: main game logic.

game.wordbank: can have word bank of different words.

game.wordsWon

wordsWon : integer that tracks how many words have been guessed correctly.

game.guessesRemaining : integer for guesses per word

game.currentWrd : the word object

game.startGame : method to start game. takes in a word object. Reset guesses remaining using resetGuessesRemaining. Get random word from word bank in game.js.

Populate currentWrd (made from Word constructor function) object with letters.

Continue prompting user with this.keepPromptingUser.

game.resetGuessesRemaining : resets guesses remaining to the default amount

game.keepPromptingUser : uses prompt module and get method to retrieve and store letter user inputs.

Output the letter guessed to terminal. Check if letter was found using checkIfLetterFound() and save the value into a variable.

Use the variable to check If the user guessed incorrectly minus the number of guesses they have left via console.log. If they guessed the letter correctly, then congratulate them via console.log. if they guessed the word correctly, congratulate them on winning and return to exit game.

Render the word using wordRender() and display letters user has guessed already. if guessesRemaining is greater than 0 and word has not been found, use keepPromptingUser(). Otherwise, end the game and log out what the answer was.