CS118 Program
Hangman, Part A

Let's write the Hangman game! We are going to break this apart so that it won't be so hard to write.

In this first version of the program (`Hangman_A`) prepare a function named `pick_word()` that will contain a hardcoded a list of at least 10 words. Have the function randomly pick a word from the list using the random `choice()` method. Copy the choice to the variable `word` then convert `word` to upper case using the `upper()` method. We're converting to uppercase so that it doesn't matter what case the user enters when s/he guesses a letter. There are no arguments to this function, but it will return the choice of word to the calling program.

The main program will not produce any output on its own. If you first prepare a printing function, you can use that function to help your main program development. Prepare a function `print_list()` that takes as arguments two values: the first is a list of letters (such as the `matched_letters` or `used_letters` lists described below) and the second is a title string to display before showing the letters in the list (this title could be the empty string). The function should provide pretty output – don't just dump the list variable to the `print()` function! To make this look nice, use a neat Python trick:

> You can make a string from a list by using the string's `join()` method. The `join()` method works on a string that will act as a separator, and then concatenating the list elements together with each one separated from the next by that string. For example:
>
> ```
> x = "."
> y = x.join(["1", "2", "3", "4"])
> ```
>
> `y` ends up holding the value: "1.2.3.4"

You can use this trick to make a nice output string that looks like this: `B R O _ N` by using a blank space for the string used for `x` and using `x.join()` on the `matched_letters` list.

Now that you have a couple of useful function, have your main program call the `pick_word()` function, collect the return value, and then make a list (not a string) called "`matched_letters`" that is the same length as that word - but make `matched_letters` contain underscores ("_") in place of the letters. As letters are matched, you will be replacing these underscores with the matched letters. To make this list, we are going to use the "multiplication" of strings, and take advantage of two string methods: `rstrip()` and `split()`:

First, make a new string called `matched_letters` that is just as long as the randomly-selected word. Make this string by "multiplying" a string `"_,"` by the length of `word` and saving the result in the variable `matched_letters`. (We are using this particular `_,` string because it will help us break up the long string into a list.) When you have this command working, you should have a string variable that looks something like this:

```
>>> matched_letters
'_,_,_,_,_,_,_,_,'
```

We want the commas to act as separators between the underscores – but we don't want the right-most comma. So use the `rstrip()` method on this string to remove it. Remember that `rstrip()` normally is used to eliminate whitespace, but can be used to eliminate any trailing character. Just give `rstrip()` a string argument which consists of the characters you want to eliminate --- in this case, a comma. Remember to save this return value in the variable `matched_letters`. When you are done, `matched_letters` should look something like this (note the last comma is gone):

```
>>> matched_letters
'_,_,_,_,_,_,_,_'
```

Finally, remove the commas by applying the `split()` method to the string. The `split()` method breaks a string into smaller strings by splitting where a certain character is found. Once again, provide the `split()` method a string that consists of a comma (remember to save this return value in the variable `matched_letters`) and you should get something that looks like this:

```
>>> matched_letters
['_', '_', '_', '_', '_', '_', '_', '_']
```

Now that you have a list of letters that have been matched, make an integer variable called `matched_count` with the value of 0. This will keep track of how many letters have been matched so far. We can't use the length of the `matched_letters` list because the underscores would be included in that number and we want only a count of how many letters actually have been matched.

Finally, make an empty list called `used_letters`. We will use this to keep track of guessed letters (correct or incorrect).

Submit the file once you have everything working.