Assignment Guidelines:

- This assignment covers material in Module 3.
- Submission details:
  - Solutions to these questions must be placed in files a03q1.py, a03q2.py, a03q3.py, and a03q4.py, respectively, and must be completed using Python 3.
  - Download the interface file from the course website to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page.
  - Natural numbers in this course begin at 0.
  - Required functions need all design recipe elements. Functions you define (eg. helper functions) need all design recipe elements except for examples and tests.
- Download the testing module from the course website. Include import check in each solution file.
  - When a function produces a floating point value, you must use check.within for your testing. Unless told otherwise, you may use a tolerance of 0.00001 in your tests.
  - Test data for all questions will always meet the stated assumptions for consumed values.
- Restrictions:
  - Do not import any modules other than math and check.
  - You are always allowed to define your own helper/wrapper functions, as long as they meet the assignment restrictions. Do not use Python constructs from later modules (e.g. dictionaries, loops (for or while or others), zip, sorted, anything with sets or enumerators, lists or any abstract list functions (e.g. map or filter) or string methods that consume or return a list). Use only the functions and methods as follows:
    * abs, len, max and min
    * Any method or constant in the math module
    * Type casting including int(), str(), float(), bool()
    * if statements and recursion
    * Any basic arithmetic operation (including +, -, *, /, //, %, **)  
    * Any basic logical operators (not, and, or)
    * String slicing and indexing as well as string operations using the operators above
    * String methods: capitalize, count, endswith, find, index, isalnum, isalpha, isdecimal, islower, isnumeric, isspace, istitle, isupper, lower, lstrip, replace, rfind, rindex, rstrip, startswith, strip, swapcase, title, upper, zfill
    * The relationship in for strings.
    * Functions are case sensitive by default unless otherwise stated
  - You may use global constants in your solutions, do not use global variables for anything other than testing.
  - Read each question carefully for additional restrictions.
  - The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
1 Third Character
Write a function

```python
new_word(s)
```
that consumes a string `s` and returns a string where all occurrences of the third character of `s` have been replaced by `#`, except for the third character itself. If `s` has less than three characters, the function returns the original string.

For example,
```
new_word("bubble") -> "#ub#le"
new_word("me") -> "me"
```

2 Palindrome
A string is called a palindrome if it reads the same backwards as forward. For this particular question, we are interested in an “alphabetic palindrome”, which reads the same backwards as forward, omitting any non-alphabetic characters and ignoring case sensitivity. An empty string is considered to be an alphabetic palindrome. Write a function

```python
alphabetic_palindrome(phrase)
```
that consumes a string `phrase` and returns `True` if it is an alphabetic palindrome and `False` otherwise.

For example,
```
alphabetic_palindrome("moM") => True
alphabetic_palindrome("computers") => False
alphabetic_palindrome("rAce .caR 12") => True
```

3 Hangman
Hangman is a classic word game where a player attempts to guess a secret word by guessing one letter at a time. For this question, you will be implementing a small part of this game. In particular, write a function

```python
update_dashes(secret, dashes, guess)
```
that consumes three strings:
- `secret`, which is the word the player is attempting to guess
- `dashes`, which is the representation of `secret` using a string of hyphens and already guessed letters, if any
- `guess`, which is the player’s newest guess

The function does not return a value, instead it prints the updated version of `dashes` if `guess` is in `secret`. If `guess` is not in `secret`, the original `dashes` is printed.

In addition, you may assume the following:
- `secret` and `dashes` are of the same length
- `secret` and `dashes` will always form a valid game. For example,
  - if `secret` is “cs”, then `dashes` could be any of the following: "--", "c-", or "-s"
  - and if `secret` is “bubble” then `dashes` could not be "--bb--" because if the user guessed “b” earlier, `dashes` would have been updated to "b-bb--"
- `guess` is always exactly one character
- use hyphens, or minus sign, to represent the unguessed letters in `dashes`
- `secret`, `dashes` and `guess` are non-empty strings
- `secret`, `dashes` and `guess` always contain all lowercase letters
Below are examples of calling update_dashes(secret, dashes, guess).

For example,
```
update_dashes("bubble", "------", "b") => None
```
and the following is printed to the screen:
```
b-bb--
```

Another example,
```
update_dashes("bubble", "b-bb--", "e") => None
```
and the following is printed to the screen:
```
b-bb-e
```

And another example,
```
update_dashes("python", "p------", "x") => None
```
and the following is printed to the screen:
```
p-----
```

4 New Password

At some point or another, you probably have had to set a new password for an account. You might have just wanted to use “password” as your password to make life a whole lot easier, but the application did not allow it. It is now your turn to make that algorithm! Write the function
```
new_password()
```
that consumes no values, but prompts the user with the phrase “Enter password: ” to enter a new password as shown below in the sample interactions. The function determines how strong your password is based on the following criteria:

- It has at least 10 characters
- It has at least 1 uppercase character
- It has at least 1 lowercase character
- It has at least 3 non-alphanumeric characters (not necessarily distinct)
- It has at least 2 digits (not necessarily distinct)
- It contains no spaces

The function returns None and prints the strength of your password. If the password contains all of the criteria listed above, then the function prints Good. If the password contains 4 or 5 out of the 6 criteria, then the function prints Fair. Otherwise, the function prints Weak. Please see the samples below, and match the prompts and the outputs exactly.

A sample interaction after calling new_password() => None
Enter password: goodPass12!@@
Good

Another sample interaction after calling new_password() => None
Enter password: fairPass !!!
Fair
Another sample interaction after calling `new_password()` => None

Enter password: weak12!
Weak

NOTE: the text in blue represents the user’s input. The text in black represents the user prompts and output, which has to be matched exactly.