Assignment Guidelines
• This assignment covers material in Module 03.
• Submission details:
  - Solutions to these questions must be placed in files a03q1.py, a03q2.py, a03q3.py, and a03q4.py.
  - You must be using Python 3 or higher. Do NOT use Python 2.
  - Download the interface file from the course Web page 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.
• Download the testing module from the course web page. Include import check in each solution file.
  - When a function returns 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.
• Restrictions:
  - Do not import any modules other than math and check.
  - Do not use Python constructs from later modules (e.g. loops and lists). Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
  - While 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.
The snake and the camel.

1. Even though they do not affect the result, conventions are important in computer programming. They help improve the readability of source code, and make software maintenance easier. For example, in Python, underscores ("_") are used to delimit words in variable names: `earned_grade`, `power_of`, etc. This convention is often called snake_case. In some other languages, like Java, there are no delimiters. Instead, the first letter of each new word is capitalized: `earnedGrade`, `powerOf`, etc. Note that the very first letter is still lower case. This convention is called camelCase.

Write the function `to_snake_case` which consumes a string that follows the camelCase convention, and converts it to snake_case. As part of your contract, assume the input string complies with the following rules:
   a. It only contains alphanumerical characters ("a" to "z", "A" to "Z", and "1" to "9"),
   b. It never starts with an uppercase character,
   c. It never starts with a number.

Here are a few examples as run through the Python interpreter:

```python
>>> earned_grade_str = to_snake_case("earnedGrade")
>>> earned_grade_str
'earned_grade'
>>> to_snake_case("powerOf")
'power_of'
>>> to_snake_case("cs116Assignment3")
'cs116_assignment3'
>>> to_snake_case("w3cMeetings")
'w3c_meetings'
>>> to_snake_case("question5c")
'question5c'
>>> to_snake_case("isXNegative")
'is_x_negative'
>>> to_snake_case(""")
''
```

Note that `to_snake_case` attaches numbers to the preceding word, so 'waterloo4Ever' becomes 'waterloo4_ever'.
A password generation technique that should certainly not be used to protect your bank account.

2. Write the function `generate_password` which consumes nothing and will prompt the user for three pieces of information (first name, birth year and favourite colour) and returns a password according to the description below.

The password will be made up of the reversed form of the favourite colour (call this REVC) surrounded by the two first digit of the birth year (call these D₁ and D₂). All of this followed by an underscore and then the two last digits of the birth year increased by 1 where 9 becomes 0 (call these D₁₃ and D₁₄) surrounded by the uppercase version of the first and last letter of the first name (call these N₁ and N₂). The format of the password is thus:

\[ D₁REVCD₂_ N₁D₁₃D₁₄N₂ \]

You may assume that each individual user input will be nonempty and that the user will enter a 4-digit year.

The prompts to be used should be (in the following order):

- Enter your favourite colour: 
- Enter your first name: 
- Enter your birth year: 

Note: " " are used to mark white space characters in the example above.

A few examples when run through the Python interpreter:

```
>>> my_password = generate_password()
Enter your favorite colour: pUrPLe
Enter your first name: Lucy
Enter your birth year: 1990
>>> my_password
'1eLPrUp9_L01Y'

>>> generate_password()
Enter your favorite colour: blue
Enter your first name: v
Enter your birth year: 2008
'2eulb0_V19V'

>>> generate_password()
Enter your favorite colour: RED
Enter your first name: ARTHUR
Enter your birth year: 0899
'0DER8_A00R'
```
Enumerating characters.

3. Write the function `print_chars` which consumes a string `text`, returns `None`, and prints on consecutive lines the number of occurrences of each unique character in `text`, in order of first appearance, and without repetition.

For example:

```python
>>> print_chars('anne')
'a' occurs 1 time
'n' occurs 2 times
'e' occurs 1 time
```
```python
>>> print_chars('I am Salsa!')
'I' occurs 1 time
' ' occurs 2 times
'a' occurs 3 times
'm' occurs 1 time
'S' occurs 1 time
'l' occurs 1 time
's' occurs 1 time
'!' occurs 1 time
```

Note the format of each print statement is “'X' occurs N times” for the character X when N is greater than 1, and “'X' occurs N time” when character X occurs once. Also note that X is surrounded by quotes, and that the count is case-sensitive.
Good for romans, not so good for the computer age.

4. In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. The method is named after Julius Caesar, who used it in his private correspondence. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet, possibly wrapping around the alphabet. For example, with a right shift of 3, ‘a’ would be replaced by ‘d’, ‘B’ would become ‘E’ (case is preserved), ‘Y’ would become ‘B’, and so on. With a shift of 26, ‘a’ would go all around the alphabet and come back to ‘a’ (because there are exactly 26 letters in the alphabet). With a shift of 27, it would become ‘b’.

Write the function `caesar_encrypt` that consumes a string of alphabetical letters and a natural number. It returns a string after applying a Caesar encryption.

For example:

```python
>>> nice_encrypted = caesar_encrypt("aBcd", 1)
>>> nice_encrypted
'bCde'
>>> caesar_encrypt("aBcd", 0)
'aBcd'
>>> caesar_encrypt("", 999)
''
>>> caesar_encrypt("wXyz", 28)
'yZab'
```

You are not allowed to use `ord()` or `chr()`.