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.
  - Helper functions need design recipe elements but not 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 functions, as long as they meet the assignment restrictions. Do **not** use Python constructs from later modules (e.g. loops, map, filter or lists), string methods that consume or return a list or the command `zip`. 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()`
    - Any basic arithmetic operation (including `+`, `-`, `*`, `/`, `//`, `%`, `**`)
    - 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`, `rsplit`, `rstrip`, `startswith`, `strip`, `swapcase`, `title`, `upper`, `zfill`
    - The relationship `in` for strings.
    - Functions are **case sensitive** by default unless otherwise stated
    - `input` and `print` as well as the formatting parameter `end` and method `format`. Note that all prompts must match exactly in order to obtain marks so ensure that you do not alter these prompts.
  - 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.**

1 Passwords

Write the function

`valid_password(s)`

that consumes a string `s` and returns `True` if and only if the string is a valid password and `False` otherwise. A password is considered valid if and only if:

- It contains no spaces
- It is at least 8 characters long
- It contains at least 1 lowercase alphabetic letter
- It contains at least 1 uppercase alphabetic letter
- It contains at least 1 number
- It contains at least one character of the following collection of special characters:

`!@#$%^&*(),.<>?{}[]+-_`

which are any number key when shift is pressed, any pair of 4 different kinds of parentheses and the symbols: period, comma, plus, minus, underscore, question mark.

Sample:

```
valid_password("aB1!filler") -> True
valid_password("f4!L") -> False
valid_password("Tru$tNo1") -> True
```

2 Gain a Mars

Two strings are said to be anagrams of each other if and only if they can both be formed using the exact same characters (including any spaces) with the same multiplicity. Write a function:

`is_anagram(s, t)`

which consumes two strings `s` and `t`, and returns `True` if and only if the two words are anagrams of each other and `False` otherwise.

Sample

```
is_anagram("meat", "team") => True
is_anagram("stay", "says") => False
is_anagram("tt", "TT") => False
```

3 Plane Walking

Write a function

`distance_from_origin()`

that consumes no values but prompts the user first for a number `n` corresponding to the number of directions it will receive. It then prompts the user for a direction from `N`, `E`, `W`, or `S` (corresponding to North, East, West or South) a total of `n` times and then it computes how far away a user would be from the origin if they walked one unit in each of the directions passed on the Cartesian plane. It prints the distance from the origin rounded to 3 decimal places and returns this value.

A sample interaction after calling `d = distance_from_origin()` is on the next page

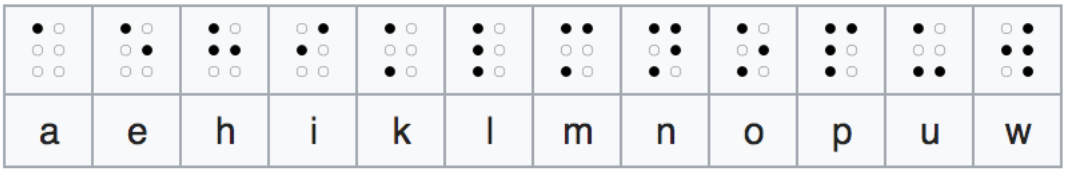
```
Enter a number of directions to be entered: 5
Enter a direction (N, E, W, S): N
Enter a direction (N, E, W, S): N
Enter a direction (N, E, W, S): S
Enter a direction (N, E, W, S): E
Enter a direction (N, E, W, S): E
The distance from the origin is 2.236 units.
```

and the value of `d` is 2.23606797749979. Use the prompt provided in the text to help with the proper display. A note that `"{0:0.3f}".format(2.23606797749979)` will help to round your decimal properly to 3 decimal places as it takes parameter 0 and formats it to match the `0.3f` standard, which is three decimal places. You may assume that the given input is valid.

4 Hawaiian Braille

For this problem, we will look at the Hawaiian language. The Hawaiian alphabet consists of just 12 letters: the vowels **a, e, i, o, u** and the consonants **h, k, l, m, n, p, w** as well as the 'okina[1] glottal stop symbol denoted by `'` (this symbol will largely be ignored for this task but might appear in words).

Braille is a system devised by Louis Braille to help blind people read by touch. It consists of a sequence of raised dots and blank spaces. The braille letters corresponding to those in the Hawaiian alphabet are:



Write a function:

`hawaiian_braille(s)`

that consumes a string `s` corresponding to a Hawaiian word (that is, only using the above 12 letters), returns `None` and prints out the braille equivalent of the string. Letters should be separated by a single space and a single newline character (no extra spaces) should be printed at the end of each of the three lines. Use the letter `'o'` to represent a raised character and use the letter `'b'` to represent a blank. You may assume that `s` is non-empty. Ignore any glottal stop symbols you might encounter.

Sample

```
hawaiian_braille("hawaiian") => None
```

and the following is printed to the screen:

```
ob ob bo ob bo bo ob oo
oo bb oo bb ob ob bb bo
bb bb bo bb bb bb bb ob
```

[1] Not a typo - the apostrophe is meant to be there before 'okina