Recap some of the string tricks

```
s = 'message'          # Assign variable s a string value 'message'
dir(s)                 # See all string attributes and methods
s = 'longer' + s + 'sounds better' # Concatenate strings
len(s)                 # Check string length
s[0]                   # pick up an alphabet at index 0 from the string
s[3:6]                 # pick up alphabets from indices 3 to 5
str(45)                # convert integer 45 to string '45'
int('27')              # convert string '27' to integer 27
```

Some handy methods

```
s.split(' ')           # split strings into a list of multiple shorter
                       # strings by character ' '
s.isnumeric()          # check if the string contains only numeric
                       # characters
s.upper()              # give an upper case version of the string
s.lower()              # give a lower case verion of the string
```

Note

1. String is immutable

We cannot do something like `s[4] = 'A'`

2. String is iterable

We can check for `'@'` in string `s = 'myemail@peacefulhome'` by

(a) We can do the classic style.

```
found = False
for i in range(len(s)):
    if s[i] == '@':
        found = True
```

(b) Or, because string is *iterable*, we can just iterate through its items

```
found = False
for c in s:
      if c == '@':
            found = True
```

Remark, in this case, we can just use `found = '@' in s`.

Don't worry if you don't get all the tricks, just have fun playing with it.

==========

P1. Write a function named `format_name` to take 2 arguments: `Firstname` and `Lastname`. The function concatenates them together with format `Lastname, Firstname`, and return it as a single string.

Example, when the function is invoked as:

```
==========================================================================
r = format_name("Ajahn", "Brahm")
print(r)
==========================================================================
```

we will see:

```
==========================================================================
Brahm, Ajahn
==========================================================================
```

P2. C major scale has 7 musical notes in an octave: 'A', 'B', 'C', 'D', 'E', 'F', and 'G'. Write a function, named `int2note`, to take an integer number from 1 to 7 and return the corresponding note.

Hint: string will make things easier.

Example, when the function is invoked as:

```
==========================================================================
n = int2note(1)
```

```
print(n)

n = int2note(3)
print(n)

n = int2note(5)
print(n)

n = int2note(7)
print(n)
```

==========

we will see

==========

```
A
C
E
G
```

==========

P3. Write a function named `firstofeach` to take 3 names, then concatenate the first letter from each name together with format $L_1L_2L_3$, and return the result as a string.

Example, when the function is invoked as:

==========

```
r = firstofeach('Jayasaro', 'Dejkunchon', 'Carroll')
print(r)
```

==========

we will see:

```
======================================================================
JDC
======================================================================
```

P4. Write a function named `concat3` to take 3 names: they can be user's idols or ones respectable. Then, the function concatenates the first 3 letters from each name together IN ORDER with format

$$L_{11}L_{12}L_{13}L_{21}L_{22}L_{23}L_{31}L_{32}L_{33}$$

and returns the result as a string.

Example, when the function is invoked as:

```
======================================================================
r = concat3("Jayasaro", "Zinsser", "Carroll")
print(r)
======================================================================
```

we will see:

```
======================================================================
JayZinCar
======================================================================
```

P5. Write a function named `vowel_count` to take a text and count the number of vowel alphabets (a, e, i, o, u) in the text, and return the number as an integer. For example, "*Alibaba*" has 4 vowel alphabets.

Hint: See searching example at the beginning. Also, `str.lower()` *or* `str.upper()` *may be handy.*

Example, when the function is invoked as:

```
======================================================================
```

```
r = vowel_count('Kruntep Mahanakorn Amornrattanakosin')
print(r)
```

```
=======================================================================
```

we will see:

```
=======================================================================
13
=======================================================================
```

P6. Write a function named `find_key` to take (1) a user's motto and (2) a keyword. Then, the function finds and returns a position of the keyword in the user's motto. If there is no keyword in the motto, returns `-1`. If there are multiple keywords found, report the first one.

Example, when the function is invoked as:

```
=======================================================================
r = find_key("For benefits and happiness of us all till the end of time",
                  "happiness")
print(r)

r = find_key("For benefits and happiness of us all till the end of time",
                  "courage")
print(r)

r = find_key("For benefits and happiness of us all till the end of time",
                  "of")
print(r)
=======================================================================
```

we will see:

```
=======================================================================
```

```
17
-1
27
```

==========

Hint: approaches:

Approach 1 (Loop and find):

Example

```
s = 'Cave filled with gemstones'
w = 'gem'
found = False
for i in range(len(s)-len(w)+1):
    if s[i:(i+len(w))] == w:
        found = True
```

Approach 2 (Method `find`)

Try `s.find(w)`

P7. Write a function named `to_key` to take arguments (1) a user's motto and (2) a keyword. Then, the function cuts the text from beginning to the keyword (including the keyword) and returns the cut out. If there is no keyword in the motto, the function returns an empty string `''`. If there are multiple keywords found, the function cuts text to the first keyword.

Example, when the function is invoked as:

==========

```
r = to_key('For benefits and happiness of us all till the end of time',
           'happiness')
print(r)

r = to_key('For benefits and happiness of us all till the end of time',
```

```
              'of')
print(r)

r = to_key('For benefits and happiness of us all till the end of time',
              'wisdom')
print(r)

r = to_key('For benefits and happiness of us all till the end of time',
              'For')
print(r)
```

==========

We will see:

==========

```
For benefits and happiness
For benefits and happiness of

For
```

==========

P8. Write a function named `k2k` to take 3 arguments: a motto and 2 keywords. Then, the function cuts the text between those 2 keywords and returns the cut out. If any or both of the keywords are not in the motto, returns an empty string `''`. If there are multiple keywords found (either for the beginning or the end), take the first one.

Example, when the function is invoked as:

==========

```
r = k2k('Life is not a game to win, but a melange of lessons to learn.',
           'game', 'Life')
print(r)
```

```
r = k2k('Life is not a game to win, but a melange of lessons to learn.',
           'not', 'win')
print(r)

r = k2k('Life is not a game to win, but a melange of lessons to learn.',
           'win', 'not')
print(r)

r = k2k('Life is not a game to win, but a melange of lessons to learn.',
           'live', 'learn')
print(r)

r = k2k('Life is not a game to win, but a melange of lessons to learn.',
           'but', 'lessons')
print(r)

r = k2k('Life is not a game to win, but a melange of lessons to learn.',
           'lessons', 'learn')
print(r)
```

==========

We will see:

==========

```
Life is not a game
not a game to win
not a game to win

but a melange of lessons
lessons to learn
```

==========

P9. (difficulty 3*) DNA is composed 4 basic coding elements: A, G, C, and T. In a DNA strand, there usually are short sequences with repeating DNA codes. The repeating code sequence is called *k-mer*. A frequently-appear k-mer is likely to have important hidden meanings. Therefore, identification of frequently-appear k-mers is the first step of studying DNA hidden meaning.

Write a function named `count_kmer` taking two arguments: `kmer` and `text`, then counting a number of `kmer`s appear in the `text` and returning the count as an integer.

Example, when the function is invoked as:

```
==========================================================================
r = count_kmer('ACTAT', 'ACAACTATGCATACTATCGGGAACTATC')
print(r)

r = count_kmer('AC', 'ACAACTATGCATACTATCGGGAACTATC')
print(r)

r = count_kmer('ATA', 'CGATATATCCATAG')
print(r)
==========================================================================
```

We will see:

```
==========================================================================
3
4
3
==========================================================================
```

Notice that

1. ACAACTATGCATACTATCGGGAACTATC has k-mer "ACTAT" appear 3 times.
2. ACAACTATGCATACTATCGGGAACTATC has k-mer "AC" appear 4 times.
3. CGATATATCCATAG has k-mer "ATA" appear 3 times including the overlapping.

P10. (difficulty 4*) In an old-time programming class, to properly turn in a programming submission, the submission file has to be named `tAA_BBBBBBBBB.tar`, where AA is a topic number and BBBBBBBBB is a student id (10 digits) without hyphen. For example, `t01_6210112341.tar` is a proper submission file name, while the followings are invalid submission file names: `t1_6210112341.tar` (missing one digit of the topic number), `6210112341.tar` (no topic number at all), `t08_621011234.tar` (student id is not complete. A digit is missing), `t04_621011234-1.tar` (there is a hyphen), `t05_6210112341.zip` (wrong file extension), `t056210112341.tar` (no underscore), `t05_62101 12341.tar` (there is a space), `T05_6210112341.tar` (capitalized T), `t15-621011234.tar` (underscore becomes hyphen), `t015_6210112341.tar` (extra digit), etc.

Write a function named `is_valid` to take in a submission file name and return boolean `True` when the name is a valid submission file name and boolean `False` when it is invalid.

Hint: function `len`, string method `islower`, and string method `isnumeric` can be handy. Also, `'-' in 'space-x'` gives `True`, while `'-' in 'Pacific'` gives `False`.

Example, when the function is invoked as:

```
=====================================================================
r = is_valid('t05_6210112341.tar')
print(r)
print(type(r))


r = is_valid('t5_621011234-1.zip')
print(r)
```

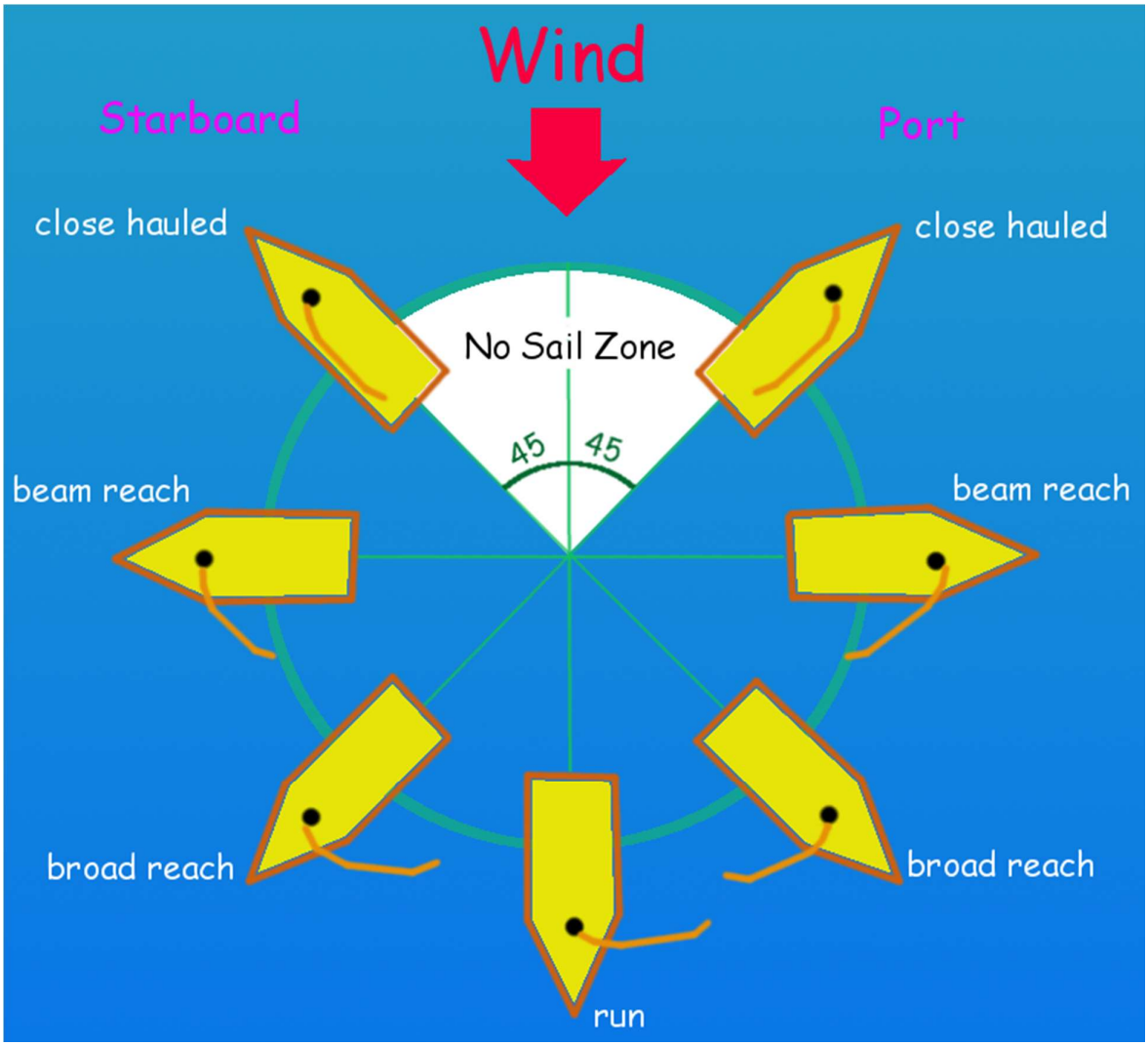
```
r = is_valid('T5_621011234-1.zip')
print(r)
```

```
==========================================================================
```

We will see:

```
==========================================================================
True
<class 'bool'>
False
False
==========================================================================
```

P11. (difficulty 5*) A sailboat harnesses the wind as its driving force. A proficient sailor can adjust the sail to go to any direction he or she wants. An illustration below shows heading direction relative to the wind direction. A sailboat cannot sail directly toward the wind or even any direction within approximately 90 degrees upwind (45 degree on either side). However, a sailor can go to an upwind location by tacking or called beating into the wind. That is, a sailor sails zig-zag toward the upwind direction.



Write a function named `sailor_mate`. The function takes wind direction and desired heading direction, determines the sailing strategy, and reports it. The wind and desired directions are represented by clock notation, e.g., `"12:00"` is northward, `"3:00"` o'clock is eastward, `"6:00"` o'clock is southward, `"9:00"` is westward, etc. The sailing strategy is one of the options: `"tacking"`, `"port close hauled"`, `"port beam reach"`, `"port broad reach"`, `"run"`, `"starboard broad reach"`, `"starboard beam reach"`, and `"starboard close hauled"`. Note: in short, when a boat is heading 0-45 degree against the wind in either side, the strategy is `"tacking"`; when a boat is heading between 45-90 degree is `"port close hauled"` if left side (called port side) taking the wind or `"starboard close hauled"` if right side (called starboard side) taking the wind; when a boat is heading perpendicular to the wind (90 degree), it is either `"port beam reach"` or

`"starboard beam reach"`; when a boat is heading between 90 and 0 along the wind, it is either `"port broad reach"` or `"starboard broad reach"`; when it is heading the wind direction, it is `"run"`.

Hint:

(1) it may be easier to work from clock notation to degree first;
(2) notice that 1 hr = 30 degree and 1 min = 0.5 degree;
(3) it may be easier to "normalize" the directions, i.e., wind direction w = 0 degree and boat direction b = 30 degree is the same as w = 10 and b = 40;
(4) since either clock notation or degree has a cycle nature, i.e., 0 o'clock = 12 o'clock and 0 degree = 360 degree. Modulo operator % may be handy;
(5) even the question does not ask, it is a good practice to print out any intermediate computation out for double-check/debug.

Example, when the function is invoked by:

```
=====================================================================
r = sailor_mate("9:00", "12:00")
print("9:00", "12:00", r)


ticks = ["12:00", "1:30", "1:31", "2:59", "3:00", \
            "3:01", "5:59", "6:00", "6:01", "8:59", "9:00", "9:01", \
            "10:29", "10:30"]
for b in ticks:
     r = sailor_mate("6:00", b)
     print("6:00", b, r)
=====================================================================
```

Note: do not worry about code in highlight. It is a list. We will learn about it later. It is just a convenient way so that we can show you a quite comprehensive example.

We will see:

```
=====================================================================
9:00 12:00 starboard beam reach
6:00 12:00 tacking
6:00 1:30 tacking
```

```
6:00 1:31 port close hauled
6:00 2:59 port close hauled
6:00 3:00 port beam reach
6:00 3:01 port broad reach
6:00 5:59 port broad reach
6:00 6:00 run
6:00 6:01 starboard broad reach
6:00 8:59 starboard broad reach
6:00 9:00 starboard beam reach
6:00 9:01 starboard close hauled
6:00 10:29 starboard close hauled
6:00 10:30 tacking
==========================================================================
```

Notice that this example covers a full cycle and emphasizes boundary cases.