Chapter 3

Strings

Chapter 3

Strings

In This Chapter:

1. sWAP cASE
2. String Split and Join
3. What's Your Name?
4. Mutations
5. Find a string
6. String Validators
7. Text Alignment
8. Text Wrap
9. Designer Door Mat
10. String Formatting
11. Alphabet Rangoli
12. Capitalize!
13. The Minion Game
14. Merge the Tools!

3.1. sWAP cASE

You are given a string S. Your task is to *swap cases*. I.e, convert all lowercase letters to uppercase letters and vice versa.

For Example:

Www.HackerRank.com → wWW.hACKERrANK.COM
Pythonist 2 → pYTHONIST 2

Input Format

A single line containing a string S.

Constraints

$0 < len(S) \leq 1000$

Output Format

Print the modified string S.

Sample Input

HackerRank.com presents "Pythonist 2".

Sample Output

```
hACKERrANK.COM PRESENTS "pYTHONIST 2".
```

Code

```
def swap_letter(letter):
    if letter.isupper():
        return letter.lower()
    if letter.islower():
        return letter.upper()
    return letter

def swap_case(s):
    return ''.join(map(swap_letter, s))
```

```
if __name__ == '__main__':
    s = raw_input()
    result = swap_case(s)
    print result
```

3.2. String Split and Join

In Python, a string can be split on a delimiter.

Example:

```
>>> a = "this is a string"
>>> a = a.split(" ") # a is converted to a list of strings.
>>> print a
['this', 'is', 'a', 'string']
```

Joining a string is simple:

```
>>> a = "-".join(a)
>>> print a
this-is-a-string
```

You are given a string. Split the string on a " " (space) delimiter and join using a - hyphen.

Input Format

The first line contains a string consisting of space separated words.

Output Format

Print the formatted string as explained above.

Sample Input

```
this is a string
```

Sample Output

```
this-is-a-string
```

Code

```
def split_and_join(line):
    return '-'.join(line.split())

if __name__ == '__main__':
    line = raw_input()
    result = split_and_join(line)
    print result
```

3.3. What's Your Name?

You are given the firstname and lastname of a person on two different lines. Your task is to read them and print the following:

```
Hello firstname lastname! You just delved into python.
```

Input Format

The first line contains the first name, and the second line contains the last name.

Constraints

The length of the first and last name ≤ 10.

Output Format

Print the output as mentioned above.

Sample Input

```
Guido
Rossum
```

Sample Output

```
Hello Guido Rossum! You just delved into python.
```

Code

```
# Enter your code here. Read input from STDIN. Print output

a=raw_input()
b=raw_input()
print "Hello "+a+" "+b+"! You just delved into python."
```

3.4. Mutations

We have seen that lists are mutable (they can be changed), and tuples are immutable (they cannot be changed).

You are given an immutable string, and you want to make changes to it.

Example

```
>>> string = "abracadabra"
```

You can access an index by:

```
>>> print string[5]
a
```

What if you would like to assign a value?

```
>>> string[5] = 'k'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

How would you approach this?

- One solution is to convert the string to a list and then change the value.

Example

```
>>> string = "abracadabra"
>>> l = list(string)
>>> l[5] = 'k'
>>> string = ''.join(l)
>>> print string
abrackdabra
```

- Another approach is to slice the string and join it back.

Example

```
>>> string = string[:5] + "k" + string[6:]
>>> print string
abrackdabra
```

Read a given string, change the character at a given index and then print the modified string.

Input Format

The first line contains a string, S.
The next line contains an integer i, denoting the index location and a character c separated by a space.

Output Format

Using any of the methods explained above, replace the character at index i with character c.

Sample Input

```
abracadabra
5 k
```

Sample Output

```
abrackdabra
```

Code

```
def mutate_string(string, position, character):
    return string[:position] + character + string[position+1:]

if __name__ == '__main__':
    s = raw_input()
    i, c = raw_input().split()
    s_new = mutate_string(s, int(i), c)
    print s_new
```

3.5. Find a string

The user enters a string and a substring. You have to print the number of times that the substring occurs in the given string. String traversal will take place from left to right, not from right to left.

NOTE: String letters are case-sensitive.

Input Format

The first line of input contains the original string. The next line contains the substring.

Constraints

$1 \leq len(string) \leq 200$

Each character in the string is an *ascii* character.

Output Format

Output the integer number indicating the total number of occurrences of the substring in the original string.

Sample Input

```
ABCDCDC
CDC
```

Sample Output

```
2
```

Concept

In Python, the length of a string s is found by the function len(s).
To traverse through the length of a string, use a *for* loop:

```
for i in range(0, len(s)):
    print (s[i])
```

A range function is used to loop over some length:

```
range (0, 5)
```

Here, the range loops over 0 to 4. 5 is excluded.

For more examples, go to
http://bit.ly/2mqlqcg
Which redirect you to
http://www.thelearningpoint.net/computer-science/learning-python-programming-and-data-structures/learning-python-programming-and-data-structures--tutorial-12--string-manipulation

Code

```
def count_substring(string, sub_string):
    #no. of comparisons = len_string - len_sub_string + 1
    len_string = len(string)
    len_sub_string = len(sub_string)
    count = 0
    len_ = len_string- len_sub_string + 1

    for x in range(0, len_):
        if string[x:x+len_sub_string] == sub_string:
            count = count+1
    return count
```

```
if __name__ == '__main__':
    string = raw_input().strip()
    sub_string = raw_input().strip()

    count = count_substring(string, sub_string)
    print count
```

3.6. String Validators

Python has built-in string validation methods for basic data. It can check if a string is composed of alphabetical characters, alphanumeric characters, digits, etc.

str.isalnum()

This method checks if all the characters of a string are alphanumeric *(a-z, A-Z and 0-9)*.

```
>>> print 'ab123'.isalnum()
True
>>> print 'ab123#'.isalnum()
False
```

str.isalpha()

This method checks if all the characters of a string are alphabetical *(a-z and A-Z)*.

```
>>> print 'abcD'.isalpha()
True
>>> print 'abcd1'.isalpha()
False
```

---

str.isdigit()

This method checks if all the characters of a string are digits *(0-9)*.

```
>>> print '1234'.isdigit()
True
>>> print '123edsd'.isdigit()
False
```

---

str.islower()

This method checks if all the characters of a string are lowercase characters *(a-z)*.

```
>>> print 'abcd123#'.islower()
True
>>> print 'Abcd123#'.islower()
False
```

---

str.isupper()

This method checks if all the characters of a string are uppercase characters *(A-Z)*.

```
>>> print 'ABCD123#'.isupper()
True
>>> print 'Abcd123#'.isupper()
False
```

You are given a string .
Your task is to find out if the string contains: *alphanumeric characters, alphabetical characters, digits, lowercase and uppercase characters*.

Input Format

A single line containing a string s.

Constraints

Output Format

In the first line, print `True` if s has any *alphanumeric characters*. Otherwise, print `False`.
In the second line, print `True` if s has any *alphabetical characters*. Otherwise, print `False`.
In the third line, print `True` if s has any *digits*. Otherwise, print `False`.
In the fourth line, print `True` if s has any *lowercase characters*. Otherwise, print `False`.
In the fifth line, print `True` if s has any *uppercase characters*. Otherwise, print `False`.

Sample Input

```
qA2
```

Sample Output

```
True
True
True
True
True
```

You can refer to python documentation on string methods at https://docs.python.org/2/library/stdtypes.html#string-methods

Code

```
if __name__ == '__main__':
    s = raw_input()
    if any(alphabet.isalnum() for alphabet in s):
        print 'True'
    else:
        print 'False'
    if any(alphabet.isalpha() for alphabet in s):
        print 'True'
    else:
        print 'False'
    if any(alphabet.isdigit() for alphabet in s):
        print 'True'
    else:
        print 'False'
    if any(alphabet.islower() for alphabet in s):
        print 'True'
    else:
        print 'False'
    if any(alphabet.isupper() for alphabet in s):
        print 'True'
    else:
        print 'False'
```

3.7. Text Alignment

In Python, a string of text can be aligned *left*, *right* and *center*.

.ljust(width)

This method returns a left aligned string of length *width*.

```
>>> width = 20
>>> print 'HackerRank'.ljust(width,'-')
HackerRank----------
```

.center(width)

This method returns a centered string of length *width*.

```
>>> width = 20
>>> print 'HackerRank'.center(width,'-')
-----HackerRank-----
```

.rjust(width)

This method returns a right aligned string of length *width*.

```
>>> width = 20
>>> print 'HackerRank'.rjust(width,'-')
----------HackerRank
```

You are given a partial code for generating the *HackerRank Logo* of variable *thickness* (must be an *odd* number).

Input Format

A single line containing the *thickness* value for the logo.

Output Format

Output the desired logo.

Sample Input

```
5
```

Sample Output

```
    H
   HHH
  HHHHH
 HHHHHHH
HHHHHHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHHHHHHHHHHHHHHHHHHHHHH
  HHHHHHHHHHHHHHHHHHHHHHHHH
  HHHHHHHHHHHHHHHHHHHHHHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
  HHHHH                HHHHH
                    HHHHHHHHH
                     HHHHHHH
                       HHHHH
                         HHH
                          H
```

```
#Code
hickness = int(raw_input()) #This must be an odd number
c = 'H'
# Top Cone
for i in range(thickness):
        print((c * i).rjust(thickness - 1) + c + (c *
        i).ljust(thickness - 1))
# Top Pillars
for i in range(thickness + 1):
        print((c * thickness).center(thickness * 2) + (c *
        thickness).center(thickness * 6))
# Middle Belt
for i in range((thickness + 1) // 2):
        print((c * thickness * 5).center(thickness * 6))
# Bottom Pillars
for i in range(thickness + 1):
        print((c * thickness).center(thickness * 2) + (c *
        thickness).center(thickness * 6))
# Bottom Cone
for i in range(thickness):
        print(((c * (thickness - i - 1)).rjust(thickness) + c + (c *
        (thickness - i - 1)).ljust(thickness)).rjust(thickness * 6))
```

3.8. Text Wrap

The *textwrap* module provides two convenient functions: *wrap()* and *fill()*.
https://docs.python.org/2/library/textwrap.html

textwrap.wrap()

The *wrap()* function wraps a single paragraph in text (a string) so that every line is *width* characters long at most.
It returns a list of output lines.

```
>>> import textwrap
>>> string = "This is a very very very very very long string."
>>> print textwrap.wrap(string,8)
['This is', 'a very', 'very', 'very', 'very', 'very', 'long', 'string.']
```

textwrap.fill()

The *fill()* function wraps a single paragraph in text and returns a single string containing the wrapped paragraph.

```
>>> import textwrap
>>> string = "This is a very very very very very long string."
>>> print textwrap.fill(string,8)
This is
a very
very
very
very
very
long
string.
```

You are given a string S and width .

Your task is to wrap the string into a paragraph of width w.

Input Format

The first line contains a string, S.

The second line contains the width, w.

Constraints

$0 < len(S) < 1000$

$0 < w < len(S)$

Output Format

Print the text wrapped paragraph.

Sample Input

```
ABCDEFGHIJKLIMNOQRSTUVWXYZ
4
```

Sample Output

```
ABCD
EFGH
IJKL
```

```
IMNO
QRST
UVWX
YZ
```

Code

```
import textwrap

def wrap(string, max_width):
    return "\r\n".join(textwrap.wrap(string, max_width))


if __name__ == '__main__':
    string, max_width = raw_input(), int(raw_input())
    result = wrap(string, max_width)
    print result
```

3.9. Designer Door Mat

Mr. Vincent works in a door mat manufacturing company. One day, he designed a new door mat with the following specifications:

- Mat size must be N × M. (N is an odd natural number, and M is 3 times N.)
- The design should have 'WELCOME' written in the center.
- The design pattern should only use |, . and - characters.

Sample Designs

```
Size: 7 x 21
---------.|.---------
------.|..|..|.------
---.|..|..|..|..|.---
```

```
-------WELCOME-------
---.|..|..|..|..|.---
------.|..|..|.------
---------.|.---------

Size: 11 x 33
---------------.|.---------------
------------.|..|..|.------------
---------.|..|..|..|..|.---------
------.|..|..|..|..|..|..|.------
---.|..|..|..|..|..|..|..|..|.---
-------------WELCOME-------------
---.|..|..|..|..|..|..|..|..|.---
------.|..|..|..|..|..|..|.------
---------.|..|..|..|..|.---------
------------.|..|..|.------------
---------------.|.---------------
```

Input Format

A single line containing the space separated values of N and M.

Constraints

$5 < N < 101$

$15 < M < 303$

Output Format

Output the design pattern.

Sample Input

```
9 27
```

Sample Output

```
------------.|.------------
---------.|..|..|.---------
------.|..|..|..|..|.------
---.|..|..|..|..|..|..|.---
----------WELCOME----------
---.|..|..|..|..|..|..|.---
------.|..|..|..|..|..|.------
---------.|..|..|..|..|.---------
------------.|..|..|.------------
```

Note: More than 6 lines of code will result in a score of 0. Comment lines are counted. Blank lines are not counted.

Code

```
N, M = map(int,raw_input().split()) # More
for i in xrange(1, N, 2):
    print str('.|.' * i).center(M, '-')
print 'WELCOME'.center(M, '-')
for i in xrange(N-2, -1, -2):
    print str('.|.' * i).center(M, '-')
```

3.10.String Formatting

Given an integer, n, print the following values for each integer i from 1 to n:

1. Decimal
2. Octal
3. Hexadecimal (capitalized)
4. Binary

The four values must be printed on a single line *in the order specified above* for each i from 1 to n. Each value should be space-padded to match the width of the *binary* value of n.

Input Format

A single integer denoting n.

Constraints

$1 \leq n \leq 99$

Output Format

Print n lines where each line i (in the range) contains the respective decimal, octal, capitalized hexadecimal, and binary values of i. Each printed value must be formatted to the width of the binary value of n.

Sample Input

```
15
```

Sample Output

```
 1   1   1    1
 2   2   2   10
 3   3   3   11
 4   4   4  100
 5   5   5  101
 6   6   6  110
 7   7   7  111
 8  10   8 1000
 9  11   9 1001
10  12   A 1010
11  13   B 1011
12  14   C 1100
13  15   D 1101
14  16   E 1110
15  17   F 1111
```

Code

```
def print_formatted(number):
    width = len('{:b}'.format(number))

    for i in range(1, number+1):
        print str.rjust(str(i), width),\
            str.rjust('{:o}'.format(i), width),\
            str.rjust('{:X}'.format(i), width),\
            str.rjust('{:b}'.format(i), width)
```

```
if __name__ == '__main__':
    n = int(raw_input())
    print_formatted(n)
```

3.11.Alphabet Rangoli

You are given an integer, N. Your task is to print an alphabet rangoli of size N. (Rangoli is a form of Indian folk art based on creation of patterns.)

Different sizes of alphabet rangoli are shown below:

```
#size 5

--------e--------
------e-d-e------
----e-d-c-d-e----
--e-d-c-b-c-d-e--
e-d-c-b-a-b-c-d-e
--e-d-c-b-c-d-e--
----e-d-c-d-e----
------e-d-e------
--------e--------

#size 10

------------------j------------------
----------------j-i-j----------------
--------------j-i-h-i-j--------------
------------j-i-h-g-h-i-j------------
----------j-i-h-g-f-g-h-i-j----------
--------j-i-h-g-f-e-f-g-h-i-j--------
------j-i-h-g-f-e-d-e-f-g-h-i-j------
----j-i-h-g-f-e-d-c-d-e-f-g-h-i-j----
--j-i-h-g-f-e-d-c-b-c-d-e-f-g-h-i-j--
j-i-h-g-f-e-d-c-b-a-b-c-d-e-f-g-h-i-j
--j-i-h-g-f-e-d-c-b-c-d-e-f-g-h-i-j--
----j-i-h-g-f-e-d-c-d-e-f-g-h-i-j----
------j-i-h-g-f-e-d-e-f-g-h-i-j------
--------j-i-h-g-f-e-f-g-h-i-j--------
----------j-i-h-g-f-g-h-i-j----------
------------j-i-h-g-h-i-j------------
--------------j-i-h-i-j--------------
----------------j-i-j----------------
------------------j------------------
```

The center of the rangoli has the first alphabet letter *a*, and the boundary has the alphabet letter (in alphabetical order).

Input Format

Only one line of input containing N, the size of the rangoli.

Constraints

$0 < N < 27$

Output Format

Print the alphabet rangoli in the format explained above.

Sample Input

```
5
```

Sample Output

```
--------e--------
------e-d-e------
----e-d-c-d-e----
--e-d-c-b-c-d-e--
e-d-c-b-a-b-c-d-e
--e-d-c-b-c-d-e--
----e-d-c-d-e----
------e-d-e------
--------e--------
```

Code

```
import string
def print_rangoli(size):
    for i in range(size - 1, 0, -1):
        row = ['-'] * (2 * size - 1)
        for j in range(size - i):
            row[size - 1 - j] = row[size - 1 + j] = string.ascii_lowercase[j + i]
        print '-'.join(row)

    for i in range(0, size):
        row = ['-'] * (2 * size - 1)
        for j in range(0, size - i):
            row[size - 1 - j] = row[size - 1 + j] = string.ascii_lowercase[j + i]
        print '-'.join(row)

if __name__ == '__main__':
    n = int(raw_input())
    print_rangoli(n)
```

3.12.Capitalize!

You are given a string S. Your task is to capitalize each word of S.

Input Format

A single line of input containing the string, S.

Constraints

$0 < len(S) < 1000$

The string consists of alphanumeric characters and spaces.

Note: in a word only the first character is capitalized.
Example 12abc when capitalized remains 12abc.

Output Format

Print the capitalized string, S.

Sample Input

hello world

Sample Output

Hello World

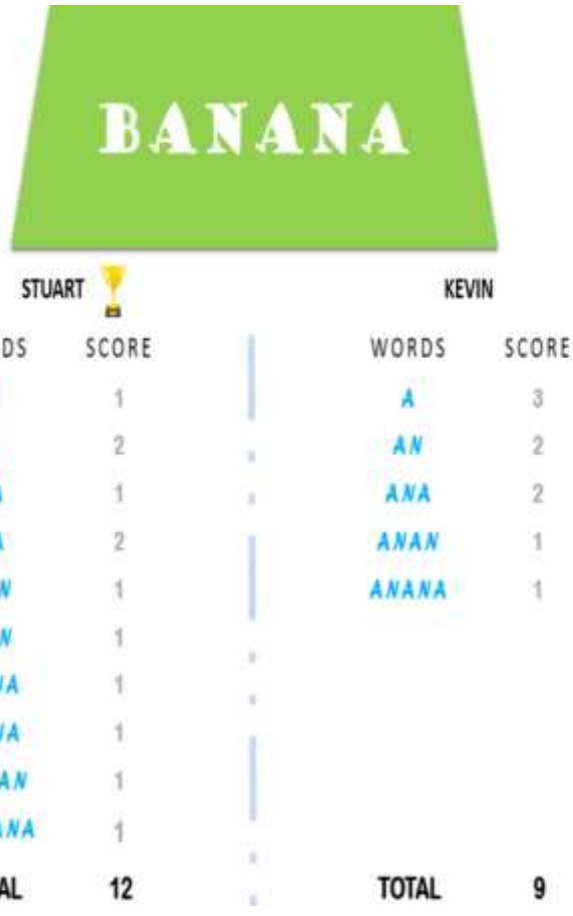
Code

```
def capitalize(s):
    return ' '.join(map(str.capitalize, s.split(' ')))
```

```
if __name__ == '__main__':
    string = raw_input()
    capitalized_string = capitalize(string)
    print capitalized_string
```

3.13.The Minion Game

Kevin and Stuart want to play the '**The Minion Game**'.
Both players are given the same string, S.
Both players make substrings using the letters of the string S.
Stuart has to make words starting with *consonants*.
Kevin has to make words starting with *vowels (a e i o u)*.
The game ends when both players make all possible substrings.
A player gets +1 point for each occurrence of substring in string S.
For Example: String = *BANANA*
Kevin's vowel beginning word = *ANA*
ANA occurs twice in *BANANA*. Hence, Kevin will get 2 Points.



Your task is to determine the winner of the game and their score.

Input Format

A single line of input is the string S (only uppercase letters).

Constraints

$0 < len(S) \leq 10^6$

Output Format

The name of the winner and his score separated by a space.

If the game is a draw, print *Draw*.

Sample Input

BANANA

Sample Output

Stuart 12

```
def minion_game(S):
    n = len(S)
    stuart = 0 # consonents
    kevin = 0  # vowels

    for i in range(n):
        if S[i] in ('A', 'E', 'I', 'O', 'U'):
            kevin += n - i
        else:
            stuart += n - i

    if kevin > stuart:
        print 'Kevin', kevin
    elif stuart > kevin:
        print 'Stuart', stuart
    else:
        print 'Draw'

if __name__ == '__main__':
    s = raw_input()
    minion_game(s)
```

3.14.Merge the Tools!

Consider the following:

- A string,s , of length n where .
- An integer,k , where k is a factor of n.

We can split s into n/k subsegments where each subsegment,t_i , consists of a contiguous block of k characters in s. Then, use each t_i to create string u_i such that:

- The characters in u_i are a subsequence of the characters in t_i.
- Any repeat occurrence of a character is removed from the string such that each character in u_i occurs exactly once. In other words, if the character at some index j in t_i occurs at a previous index <j in t_i, then do not include the character in string u_i.

Given s and k, print s/k lines where each line i denotes string u_i.

Input Format

The first line contains a single string denoting s.

The second line contains an integer, k, denoting the length of each subsegment.

Constraints

$1 \leq n \leq 10^4$

$1 \leq k \leq n$

It is guaranteed that n is a multiple of k.

Output Format

Print n/k lines where each line i contains string u_i.

Sample Input

```
AABCAAADA
3
```

Sample Output

```
AB
CA
AD
```

Explanation

String s is split into n/k = 3 equal parts of length .

We convert each t_i to u_i by removing any subsequent occurrences non-distinct characters in t_i:

1. $t_0 = \texttt{"AAB"} \rightarrow u_0 = \texttt{"AB"}$
2. $t_1 = \texttt{"CAA"} \rightarrow u_1 = \texttt{"CA"}$
3. $t_2 = \texttt{"ADA"} \rightarrow u_2 = \texttt{"AD"}$

We then print each u_i on a new line.

```
from collections import OrderedDict

def merge_the_tools(string, k):
    n = len(string)
    p = n / k

    for i in range(p):
        start = i * k
        end = start + k
        #print ''.join(ch for ch, _ in itertools.groupby(string
        print "".join(OrderedDict.fromkeys(string[start:end]))
```

```
if __name__ == '__main__':
    string, k = raw_input(), int(raw_input())
    merge_the_tools(string, k)
```