MAR 2024	S	M	T	W	T	F	S
	31					1	2
	3	4	5	6	7	8	9
	10	11	12	13	14	15	16
	17	18	19	20	21	22	23
	24	25	26	27	28	29	30

Strings

FEBRUARY '24 | 07th Week 043-323 | 12 MONDAY

String Module

TO Provide flexibility for programmer Python introduced String module → Inbuilt module.

08

ascii_letters = list of all lower and upper case character.

09 ascii_lowercase = list of all lower case character.

ascii_uppercase = list of all upper case character.

10 digits = list of all the digits.

hex digit = list of all digit & char in hexadecimal.

11 Whitespace = list of all the white spaces.

Punctuation = list of all special character in keyboards.

12

Reverse a string.

01 (a)
```
S= input()
rev = ""
for i in S:
    rev= i +rev
print (rev)
```
← यें 'i' beginning में जोड़ता है

(b)
```
print(S[::-1])
```

a=97, A=65 } unicode value

given String is rotation of another String (or) not.

04
```
def fn(S1,S2):
    if len(S1) != len(S2)
        return False.
    temp= S1+S1
    return temp.find(S2) != -1
```
O(n)

S1= 'ABCD'
S2 = 'CDAB'
S1+S2 = ABCD CDAB ×
S1+S2 = ABCDABCD

07

Palindrome

```
return S==S[::-1]
```

Remove duplicate

```
def fn(s):
    l=[]
    for ch in s:
        if ch not in l:
            l.append(ch)
    return ''.join(l)
```

Sort the String

```
def fn(S):
    l= list(S)
    l.sort()
    return ''.join(l)
```
list to string

FEBRUARY 13 '24 TUESDAY
07th Week 044-322

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

ord("a") = integer representation of character
chr(65) = character for ASCII value

String is subsequence of other (or) not

A subsequence of a string is obtained by removing zero (or) more character and whatever the character we pick, character that we don't remove, we need to put those characters in the same order as they appear in the original string (don't disturb relation position)

eg: 'ABC' = '', 'A', 'B', 'C', 'AB', 'AC', 'BC', 'ABC' → 2^n.

(a)
```
def subseq(S1, S2):
    i, j = 0, 0
    while i < len(S1) and j < len(S2):
        if S1[i] == S2[j]:
            j = j + 1
        i = i + 1
    if j == len(S2):
        return True
    else:
        return False
```
TC = O(n)

(b)
```
def subseq(S1, S2, m, n)
    if n == 0:
        return True
    if m == 0:
        return False
    if S1[m-1] == S2[n-1]:
        return subseq(S1, S2, m-1, n-1)
    else:
        return subseq(S1, S2, m-1, n)
```
TC = O(n)

Substring (or) not

Characters has to be in consecutive and there should same characters as they appear in given string.

eg: 'ABCD' → '', 'A', 'B', 'C', 'D', 'AB', 'AC', 'AD', 'BC', 'BD', 'CD', 'ABC', 'BCD'

```
S1 = 'geeks for geeks'
S2 = 'geeks'
print(S2 in S1) # True
print(S2 not in S1) # False
```

digit = '[0-9]'
spaces = '[ ]'
special char = '[^a-z A-Z 0-9]'

Count No of vowels

(a)
```
import re
def fun(s):
    return len(re.findall('[aeiou]', s))
s = input()
Print(s)
Print(fun(s))
```

(b)
```
count = 0
for i in s:
    if i in 'aeiou':
        count += 1
return count
```

Count Consonants

```
import re
def fun(s):
    return len(re.findall('[^aeiou]', s))
s = input()
```

```
count = 0
for i in s:
    if i not in('aeiou'):
        count += 1
return count
```

Count alphabet

```
import re
def fun(s):
    return len(re.findall('[a-zA-Z]', s))
s = input()
```

[^aeiou] = matches string that starts with a vowel.
[^aeiou] = matches any character that is not vowel.

Count spaces

(a)
```
def fun(s)
    return s.count(' ')
```

(b) re.findall('[ ]')

Count No. of uppercase letters.

```
count = 0
for ch in s:
    if ch in string.ascii_uppercase:
        count += 1
Print(count)
```

re.findall('[A-Z]')

Count No. of digit in string

```
count = 0
for ch in s:
    if ch in string.digits:
        count += 1
print(count)
```

FEBRUARY
15 '24
THURSDAY
07th Week
046-320

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

Convert even index chas into uppercase & odd into lower case.

```
08  def fun(s):
        res = ''
09      for i in range(len(s)):
            if i%2 != 0:
10              res = res + s[i].upper()
            else:
11              res = res + s[i].
        return res.
12  print(fun(s))
```

reverse even index words

```
def fun(s):
02      l=[]
        index = 0
03      for i in s.split(' '):
            if index%2==0:                    → may be odd
04              l.append(i[::-1])             → may be lower case / upper case
            else:
05              l.append(i)
            index+=1
06      return ' '.join(l)
```

Middle character of string

```
def fun(s)
    if len(s)%2 !=0:
        return s[len(s)//2]
    else:
        return s[len(s)//2-1 : len(s)//2+1]
```

→ No. of times of occurrence also be same

Anagram (both string contain same characters)

M-1
```
def anag(S1, S2)
    if len(S1) != len(S2)
        return False
    S1 = sorted(S1)
    S2 = sorted(S2)
    return (S1 == S2)
```
O(nlogn)

Left most repeating character.

(a)
```
def fun(S):
    for i in range(len(S)):
        for J in range(i+1, len(str)):
            if St[i] == St[J]
                return i
    return -1
```
⇒ TC — O(n²)

(b) — abccbd # 1
```
char = 256
def ch(str):
    count = [0] * char
    for i in range(len(st
        count[ord(St[i])]
            += 1
    for i in range(len(st))
        if count[ord(St[i]
            > 1:
            return i
    return -1
Str = 'abccbd' # 1
```

Left most NONe repeating char.

(a)
```
def NONrep(str):
    for i in range(len(str)):
        flag = False
        for J in range(i+1, len(str)):
            if str[i] == str[J]:
                flag = True
                break
        if flag == False
            return i
    return -1.
```

(b)
```
char = 256
def nonrep(str):
    count = [0] * char
    for i in str:
        count[ord(i)] += 1
    for i in range(len(str)):
        if (count[ord(str[i])]
            == 1:
            return i
    return -1.
```

list to string = ''.join( )

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

Word count — No. of spaces count करके add (+1).

Reverse order of words.

i/p – Anubhav Kumar Gupta   o/p – Gupta Kumar Anubhav

```
S= input(' ')
l = s.split()
l1 = l[::-1]
output = " ".join(l1)
print(output)
```

kth smallest element

```
def fun(arr,n,k):
    arr.sort()
    print(arr[k-1])
```

Palindrome or not?

M-2

```
S=input(" ")
low= 0
high = len(str)-1
while low < high:
    if S[low] != S[high]:
        print("no")
        break
    low = low+1
    high= high-1
else:
    print("yes")
```

Check if string is rotated (or) not.

```
def rot(s1, s2, k):
    if len(s1) != len(s2):
        return False

    left rotated = s1[k:]+s1[:k]
    right rotated = s1[-k:] + s1[:-k]

    return s2 == left rotated or s2 == right rotated.
```

Pangram = all 26 letters must be there.

```
def Pan(s):
    return len(set(s)) == 26
```

Pattern Searching

i/p - txt = 'geeksforgeeks'
pat = 'eks'
o/p = 2 10

i/p - txt = 'aaaaa'
pat = 'aaa'
o/p = 0 1 2

→ Pattern searching algorithms are sometimes also refferd to as a string search algorithm and are considered as a part of the string algorithm.

These algorithms are useful in the case of searching a string within another string.

Features :—

(a) Pattern searching algo should recognize familiar pattern quickly & accurately
(b) Recognize and classify unfamiliar patters.
(c) identify pattern even when partly hidden.
(d) Recognize quickly with ease and with automaticity.

→ Naive : $\theta((n-m+1)*m)$ } No preProcessing
→ Naive when all character of pattern are distinct : $O(n)$ }

m → length path
n → txt length
$1 \leq m \leq n$

→ Rabin Karp : $\theta((n-m+1)*m)$ } preProcess pattern
But better than Naive on Average

→ KMP : $O(n)$

Suffix tree : $O(m)$ preprocess text
~~→ Data structure (Trie)~~

Apart from KMP - Boyer Moore, 2 algorithms are also in linear time

FEBRUARY
19
'24
08th Week
050-316
MONDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

↳ Google search, youtube search, Ctrl+F is also a pattern searching

08

i/p = txt = 'geeksforgeeks' o/p — 0 10
09 pat = 'geeks'

10 i/p = txt = "AAAAA" o/p — 0 1 2
pat = 'AAA'

11

⟹ txt = input ()
12 pat = input ()
pos = txt.find (pat)
01 while pos >= 0:
print (pos)
02 pos = txt.find (pat, pos+1)

03 Search the pattern from pos+1,
so that this particular occurrence that I found already is not appeared ag

04

05

06

07

Sliding Window Protocal

Longest Substring w/o Repeating character.

(I) abcdebe adf.

(II) ~~abcabc~~ ababc

l= 1 → a b a b c
len= 2 → ab ba ab bc
len= 3 → aba bab (abc)
len= 4 → abab babc
len= 5 → ababc

Initially window length = 0,
↳ all must be unique on that window
then increase the length of that window by '1'

यदि कोई भी repeate कर रहा तो पहला repeating character को हटाना है।

We have to find longest NON repeating character.

↳ We can solve using two pointers as well. — O(n2)

↳ TC = O(n)

abcdee

जब तक वो repeating character बाहर ना हो जाए, इसलिए इस concept को sliding window बोलते है।

move F

F → first, S → second

abcdecbeadf ⇒ len = 0

abcdecbeadf ⇒ l = 1

abcdecbeadf ⇒ l = ~~2~~ ~~3~~ ~~4~~ 5

abcdecbeadf → decbeadf

remove it

len = 6 ⇐ cbeadf