Introduction

String represents group of characters.

Most of the real time data is in the form of strings.

Example: The names of the person, their Addresses, vehicle number, credit card number etc.

In python **str** datatype represnts string

Note: There is no separate datatype to represent individual characters. Python handles strings and characters in the same manner

Creating a String

It can be created by assigning a group of characters to a variable.

The group of characters should be enclosed inside single quotes or double quotes.

```
s1 = 'Welcome to Core Python Learning'
s2 = " Welcome to Core Python Learning "
```

There is no difference between single quotes and double quotes while creating the strings.

Tripple single quotes or **Tripple double quotes** can be used to represent strings. They are useful to represent multiline strings

```
data = ''' Welcome to core Python
The basic concepts of Python can be easily learned.
We will start with data types and operators '''
```

The above string can also be written using double quotes

```
data = """ Welcome to core Python
The basic concepts of Python can be easily learned.
We will start with data types and operators """
```

Marking a Substring

Quotation marks can be used to mark a substring in a string

In that case use one type of quotes for Outer string and another type of quotes for inner string as given in following examples

```
s1 = 'Welcome to "Core Python" Learning'
print(s1)

s2 = "Welcome to 'Core Python' Learning"
print(s2)
```

```
Welcome to "Core Python" Learning
Welcome to 'Core Python' Learning
```

Escape Characters

Different escape characters can be used to format the strings.

The list of Escape characters are as follows:

- \b : Backspace
- \n : New Line
- \t : Horizontal Tab Space
- \v : Vertical Tab Space
- \r : Enter button
- \x : Character x
- \ : Displays single ' \ '

Example to demonstrate use of \n and \t

```
s1 = "Welcome to \t Core Python \nLearning"
print(s1)
```

```
Welcome to       Core Python
Learning
```

To nullify the effect of escape characters, 'raw' string can be created by adding 'r' before the string.

Raw strings take escape characters as ordinary characters in a string and display them as it is.

```
s1 = r"Welcome to \t Core Python \nLearning"
print(s1)
```

```
Welcome to \t Core Python \nLearning
```

To create a string with Unicode Characters, Add 'u' at the begining of the string.

Each unicode character contains 4 digits preceded by a '\u'

```
name = u'\u0915\u094b\u0930 \u092a\u0948\u0925\u0964\u0928'
print(name)
```

```
कोर पैथ।न
```

Length of a String

It represents number of characters in string.

The function can be used is len()

It returns number of characters in the string including space

```
s1 = 'Core Python'
n = len(s1)
print("Length of String:",n)
```

```
Length of String: 11
```

Indexing of Strings

Index represents the position number. It is written using square braces [ ].

By specifying the position number through an index, individual elements can be referred.

str[i] : Refers ith element of the string.

Hence 'i' is called the string index as it is specifying the position number of the elements in the string

Whenever index number is negative number, it refers to elements in reverse order

Index	0	1	2	3	4	5	6	7	8	9	10	11
Character	H	a	p	p	y		C	o	d	i	n	g
Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Program 1: Program to access each element of string in forward and reverse orders using while loop

```
# indexing on string
```

```
st1 = "Core Python"

# Access each character using while loop
n = len(st1)
i = 0
while i<n :
  print(st1[i], end= ' ')
  i += 1

# Put cursor on next line
print()

# Access in reverse order
i = -1
while i >= -n:               # Here i will take values -1, -2, -3, ...
  print(st1[i], end = ' ')
  i -= 1

# Put cursor on next line
print()

# Access in reverse order using negative indices
i = 1
n = len(st1)
while i<=n:                  # Here i will take values 1, 2, 3, ...
  print(st1[-i], end = ' ')
  i += 1
```

```
C o r e   P y t h o n
n o h t y P   e r o C
n o h t y P   e r o C
```

For Loop can be used to access each element of String.

The following for loop simply takes each element into variable 'i' and displays it

```
for i in st1:
   print(i)
```

To display string in reverse order, slicing operation on string can be used.

The format of slicing is

stringname[start : stop : stepsize]

If not specified then default values are:

- start: 0
- stop: n-1
- stepsize: 1

Following for loop will display all elements of string:

```
for i in st1[ : : ] :
   print(i)
```

To get all elements in reverse order use stepsize negative as -1

This will display the elements from last to first elements in steps of 1 in reverse order

```
for i in st1 [ : : -1] :
   print(i)
```

Program 2: To access the characters of a string using for loop

```
# Accessing elements of string using for loop
st1 = "Core Python"

# Access each character using for loop
for i in st1:
  print(i, end=' ')

# Put cursor on new line
print()

# Access in reverse order
for i in st1[ : : -1]:
  print(i, end=' ')
```

```
C o r e   P y t h o n
n o h t y P   e r o C
```

Slicing the Strings

A slice represents a part or piece of a string.

The format of slicing is

```
stringname [ start : stop : stepsize ]
```

If not specified then default values are:

- start: 0
- stop: n - 1

- stepsize: 1

[ ] ↳ *4 cells hidden*

Repeating the Strings

The repeatition operator is denoted by '*' symbol.

It is useful to repeat the string for several times.

str * n repeats the string for n times

[ ] ↳ *1 cell hidden*

Concatenation of Strings

The operator '+' is called addition operator when used on numbers.

When '+' operator is used on strings, it is called concatenation operator.

It joins or concatenates the strings.

[ ] ↳ *1 cell hidden*

Checking Membership

A string or character is a member of another string or not using **'in'** or **'not in'** operator

The 'in' operator returns True if the string or character is found in the main string. Otherwise it returns False.

The 'not in' operator returns False if a string or character is found in the main string, otherwise it returns True.

The operators 'in' and 'not in' make case sensitive comparisons.

Program 3: Program to know whether a sub string exists in main string or not

```
# To know whether sub string is in main string or not

st1 = input("Enter the Main String:")
sub = input("Enter the sub string:")

if sub in st1:
  print(sub,"is found in the main string")
```

```
  print(sub, "is found in the main string")
else:
  print(sub,"is not found in the main string")
```

```
Enter the Main String:This is python class
Enter the sub string:is
is is found in the main string
```

Comparing Strings

Relational operators like >, >=, <, <=, ==, != can be used to compare two strings.

It returns Boolean value 'True' or 'False' depending on the strings being compared.

While comparing the strings, Python interpreter compares them by taking them in English Dictionary Order.

The string which comes first in the dictionary order will have low value than the string which comes later.

[ ] ↳ *1 cell hidden*

Removing Spaces from a String

A space is also considered as a charcter inside the string.

The unnecessary spaces in a string will lead to wrong results.

```
if 'Mukesh  ' == 'Mukesh':
    print('Welcome')
else:
    print('Name not found')

# The output of above code will be 'Name not found'
# Here first string has extra two spaces which results into unequal strings
```

Extra spaces can be removed before they compared.

It is possible with following three methods:

1. rstrip( ): It removes the spaces which are at the right side of the string.
2. lstrip( ): It removes the spaces which are at the left side of the string.
3. strip ( ): It removes spaces from both the sides of the string.

Note: These methods do not remove spaces which are in the middle of the string.

```
name = "  Happy   Coding   "
print("Use of lstrip:", name.lstrip(),"Right spaces remains as it is")
print("Use of rstrip:", name.rstrip(),"Right spaces are removed")
print("Use of strip:", name.strip(),"Both side spaces are removed")
```

```
Use of lstrip: Happy   Coding    Right spaces remains as it is
Use of rstrip:   Happy   Coding Right spaces are removed
Use of strip: Happy   Coding Both side spaces are removed
```

Finding Substrings

The methods to locate substrings in a strings are:

1. find( ) and index( ) : Search the first occurrence of sub string from the begining
2. rfind( ) and rindex( ) : Search the first occurrence of substring from right to left direction

The find ( ) method returns -1 if the substring is not found in the main string

The index ( ) method returns 'ValueError' exception if the substring is not found.

The format of the method is:

```
mainstring.find(substring, begining, ending)
mainstring.index(substring, begining, ending)
```

Program 4: Program to find first occurrence of sub string in a given main string

```
# to find first occurrence of sub string in a given main string
st1 = input("Enter main string:")
sub = input("Enter sub string:")

# Find position of sub in st1
# Search from 0th to last character in str
n = st1.find(sub, 0, len(st1))

# find() returns -1 if sub string is not found
if n == -1:
  print("Sub string not found")
else:
  print("Sub string found at position:", n+1)
```

```
Enter main string:This is python class
Enter sub string:is
Sub string found at position: 3
```

Program 5: Program to find first occurrence of sub string in a given main string using index( )

```
# to find first occurrence of sub string in a given main string
st1 = input("Enter main string:")
sub = input("Enter sub string:")

# Find position of sub in st1
# Search from 0th to last character in str
try:
  n = st1.index(sub, 0, len(st1))
except ValueError:
  print('Substring not found')
else:
  print("Sub string found at position:", n+1)
```

```
Enter main string:This is python class
Enter sub string:is
Sub string found at position: 3
```

Methods find( ) and index ( ) returns index of **only first occurrence** of substring.

If we want all the occurrences of sub string then we need to develop additional logic.

Whenever first occurrence of substring is identified, the method will return the index.

Further searching should be carried out from next position of returned index onwards.

Program 6: Program to display all positions of sub string in a given main string

```
# To find all occurrences of sub string in a main string
st1 = input("Enter main string:")
sub = input("Enter sub string:")

i = 0
flag = False                          # Becomes True if substring is found
n = len(st1)

while i < n :
  pos = st1.find(sub, i, n)
  if pos != -1:
    print("Found at position:",pos+1)
    i = pos + 1                        # Search from pos+1 position onwards
    flag = True
  else:
    i = i + 1                          # Search from next position onwards

if flag == False:
  print("Substring not found")
```

```
Enter main string:This is python class
```

```
Enter sub string:is
Found at position: 3
Found at position: 6
```

Program 7: Program to display all positions of sub string in a given main string - Version 2

```
# To find all occurrences of sub string in a main string
st1 = input("Enter main string:")
sub = input("Enter sub string:")

pos = -1                                # Initialization
flag = False                            # Becomes True if substring is found
n = len(st1)

while True :                           # The loop will continue forever
  pos = st1.find(sub, pos+1, n)        # Find the index of sub string and start searching from
  if pos == -1:                        # Breaks the while loop when there is no substring
    break;
  print("Found at position:", pos+1)
  flag = True

if flag == False:
  print("Substring not found")
```

```
Enter main string:This is python class
Enter sub string:is
Found at position: 3
Found at position: 6
```

Counting Substring in a String

The method count( ) can be used to count number of occurrences of sub string in a given string.

There are two forms to use the count method

```
# It returns integer value that represents how many times sub string has appeared in main string
stringname.count(substring)
```

```
# It limits the search for given indices
stringnam.count(substring, begin, end)
```

Demonstration of count( ) method

```
st = "New Delhi"
```

```
n = st.count('Delhi')
print("Count for 'Delhi':",n)

n = st.count('e',0,3)
print("Count for 'e':",n)

n = st.count('e',0,len(st))
print("Count for 'e':",n)
```

```
Count for 'Delhi': 1
Count for 'e': 1
Count for 'e': 2
```

Strings are Immutable

Immutable Object: The object whose content cannot be changed.

Example: numbers, string, tuples

Mutable Object: The object whose content can be changed.

Example: Lists, set, dictionaries

The reasons behind the strings are **immutable** in Python.

1. Performance:

For immutable objects there is fixed size in memory. Hence allocation time and access time is less.

It improves performance of software

2. Security:

As the strings are immutable, any attempt to modify the existing string object will create new object in a memory.

The **Changed Identity Number** of the object will get the understanding that someone modified original string.

```
st = 'abcd'
print(st[0])

st[0] = 'x'
```

The above Error displays that strings are immutable

Suppose we have two strings s1 and s2.

```
s1 = 'one'
s2 = 'two'
```

Here s1 is pointing to **'one'** object and s2 is pointing to **'two'** object.

If we make an assignment statement as

```
s2 = s1
```

Now both s1 and s2 are pointing to same object **'one'** The object **two** is not modified at all.

Hence whenever we print the value of s2, it will show as 'one' but string objects are not modified.

Verification of above example is as follows:

```
s1 = 'one'
s2 = 'two'
print('Before Assignment')
print("id of s1",id(s1))
print("id of s2",id(s2))

s2 = s1
print('After Assignment')
print("id of s1",id(s1))
print("id of s2",id(s2))
```

```
Before Assignment
id of s1 140634638591472
id of s2 140634638591536
After Assignment
id of s1 140634638591472
id of s2 140634638591472
```

Replacing a string with another String

replace( ): The method replaces a sub string in a string with another string

The format of method is

```
stringname.replace(old,new)
```

Original string is not modified in above method

```
st = 'That is a beautiful City'
```

```
s1 = 'City'
s2 = 'Flower'
st1 = st.replace(s1,s2)

print(st)
print(st1)
```

```
That is a beautiful City
That is a beautiful Flower
```

Splitting and Joining String

split( ): The method is used to break a string into pieces. These pieces are returned as a list.

One can use any character as a separator which can be mentioned inside the parenthesis

```
st.split(',') # Here separator is comma
```

Example for splitting of string

```
st = 'One, Two, Three, Four'
st1 = st.split(',')
print(st1)
```

```
['One', ' Two', ' Three', ' Four']
```

Program 8: Program to accept and display group of numbers

```
# To accept group of numbers and display them
st = input("Enter the numbers separated by spaces")

# Cut the string where space is found
lst = st.split(' ')

# Display the numbers from the list
for i in lst:
  print(i, end='\t')
```

```
Enter the numbers separated by spaces11 22 33 44 55
11      22      33      44      55
```

join( ): To join group of strings and make a single string

The format of the method is

```
separator.join(st)
# Separator: The character to be used between the strings in the output
# st: tuple or list of strings
```

To demonstrate use of join() method

```
st = ['one','two','three']
st1 = "--".join(st)
print(st1)

st2 = '.'.join(st)
print(st2)

st3 = ''.join(st)
print(st3)

st4 = ['apple','guava','grapes','mango']
st5 = ':'.join(st4)
print(st5)
```

```
one--two--three
one.two.three
onetwothree
apple:guava:grapes:mango
```

Changing case of String

There are four methods to chaange case of a string:

1. upper( ): To convert all the characters of a string to uppercase

2. lower( ): To convert all the characters of a string to lowercase

3. swapcase( ): To convert capital letters into small letter and vice versa

4. title( ): To convert the string such that each word in the string will start with capital letter and remaining will be the small letter

```
st = "Python is the future"

st1 = st.upper()
print("Uppercase:",st1)

st2 = st.lower()
print("Lowercase:",st2)
```

```
st3 = st.swapcase()
print("Swapcase:",st3)

st4 = st.title()
print("Title:",st4)
```

```
Uppercase: PYTHON IS THE FUTURE
Lowercase: python is the future
Swapcase: pYTHON IS THE FUTURE
Title: Python Is The Future
```

Checking Starting and Ending of a String

startswith( ): The method is useful to check whether a string is starting with substring or not.

The format of the method is

```
stringname.startswith(substring)
```

When substring is found in the main string, it returns True, otherwise False

endswith( ): The method is useful to check whether a string is ending with substring or not.

The format of the method is

```
stringname.endswith(substring)
```

When substring is found in the main string, it returns True, otherwise False

```
st = "This is Python"
print(st.startswith('This'))
print(st.endswith('Python'))
```

```
True
True
```

String Testing Method

There are several methods to test the nature of characters in string.

These methods returns either True or False.

These methods can be applied to individual characters as well

1. isalnum( ): This method returns True if there is atleast one character and all the characters in a string are alphanumeric (A to Z, a to z, 0 to 9). Otherwise it returns False.

2. isalpha( ): This method returns True if there is atleast one character and all the characters in a string are alphabetic (A to Z, a to z). Otherwise it returns False.

3. isdigit( ): This method returns True if string contains only numeric digits (0 to 9). Otherwise it returns False.

4. islower( ): This method returns True if string contains atleat one character and all the characters are in lowercase. Otherwise it returns False.

5. isupper( ): This method returns True if string contains atleat one character and all the characters are in uppercase. Otherwise it returns False.

6. istitle(): This method returns True if each word of the string starts with capital letter and there is atleast one character in the string. Otherwise it returns False

```
st = "Delhi 999"
print("Has string all alphabates?",st.isalpha())

st1 = "Delhi"
print("Has string all alphabates?",st1.isalpha())
```

```
Has string all alphabates? False
Has string all alphabates? True
```

Working with Characters

Characters are individual elements of string.

Characters can be retrieved from string using indexing and slicing

Example to demonstrate how charaters can be retrieved using indexing and slicing

```
st = "Hello"

# obtaining character using indexing
ch1 = st[0]
print(ch1)

ch2 = st[1]
print(ch2)

# Obtaining character using slicing
ch3 = st[0:1]
```

```
print(ch3)

ch4 = st[1:2]
print(ch4)
```

```
H
e
H
e
```

We can apply string testing method on individual characters as well.

It is useful to identify what type of character it is

Program 9: Program to know what type of character entered by user

```
# to know the nature of character
st = input('Enter a character:')
ch = st[0]                          # Consider only 0th character

if ch.isalnum():
  print('It is alphabet or numeric character')
  if ch.isalpha():
    print('It is an alphabet')
    if ch.isupper():
      print("It is capital letter")
    else:
      print("It is lowercase letter")
  else:
    print("It is numeric digit")
elif ch.isspace():
  print("It is space")
else:
  print("It may be a special character")
```

```
Enter a character:@
It may be a special character
```

Sorting Strings

The sort( ) method is used to sort the group of strings into alphabetical order.

Format of method sort ( )

```
st.sort()
```

Here st represents an array that contain group of strings.

Original order of strings will be lost as the changes are made in same string on which the method is called.

To retain the original array even after sorting, we can use sorted( ) function

Format of the function sorted( )

```
st1 = sorted(st)
```

Here st is original array which will not be modified. The sorted array is stored in st1 in above case

Program 10: Program to sort group of strings into alphabetical order

```
# Sorting group of strings
st = [ ] # Take an empty array

# Accept count of strings
n = int(input("How many strings?"))

# Append strings to st array
for i in range(n):
  print("Enter a string", end = ' ')
  st.append(input())
st1=st
# Sort the array
st.sort()
print(st)
st2 = sorted(st1)

print("Sorted List:")
for i in st2:
  print(i)
```

```
How many strings?3
Enter a string apple
Enter a string mango
Enter a string kiwi
['apple', 'kiwi', 'mango']
Sorted List:
apple
kiwi
mango
```

Searching in the String

A target string is searched in group of strings.

The most useful techniques are sequential search or linear search.

It is done by comparing the searching string 's' with every string in the group

The basic logic for searching is:

```
 for i in range(len(st)):  # Repeat from 0th to n-1th string
    if s == st[i]   # if s is matching with st[i] then found
```

Program 11: A program to search for the position of string in a given group of string

```
# Sorting group of strings
st = [ ] # Take an empty array

# Accept count of strings
n = int(input("How many strings?"))

# Append strings to st array
for i in range(n):
  print("Enter a string", end = ' ')
  st.append(input())
# Ask for the string to search
s = input("Enter string to search:")

# Linear search or sequential search
flag = False

for i in range(len(st)):
  if s == st[i]:
    print("Found at ", i+1)
    flag = True

if flag == False:
  print("Not Found")
```

```
    How many strings?3
    Enter a string MP
    Enter a string PY
    Enter a string AOA
    Enter string to search:PY
    Found at  2
```

ab# **Finding Number of Characters and Words**

len ( ): The function returns number of characters in the string

We can find the number of characters in string without len ( ) function using following logic:

```
i = 0
for s in st:
   i += 1
```

Program 12: A program to find the length of string without using len( ) function

```
# To find no of characters in a string
st = input("Enter a string:")

i = 0
for s in st:
  print(st[i], end = '')
  i += 1

print("No of characters", i)
```

```
Enter a string:MADAM
MADAMNo of characters 5
```

Program 13: To find the number of words in a string

Logic: To count the number of spaces and add 1 to it

If there are more than one space then it can be handled using 'flag'

```
# To find number of words in a string
st = input('Enter a string:')

i = 0                 # To keep track of each index of string
c = 0                 # Count Number of spaces

Flag = True           # This becomes False when no space is found
for s in st:
  if Flag == False and st[i] == ' ':     # Count only when there is no space previously
    c = c + 1
  if st[i] == ' ':
    Flag = True
  else:
    Flag = False
  i = i + 1

print('No of Words:', c+1)
```

```
Enter a string:Today is thursday
No of Words: 3
```

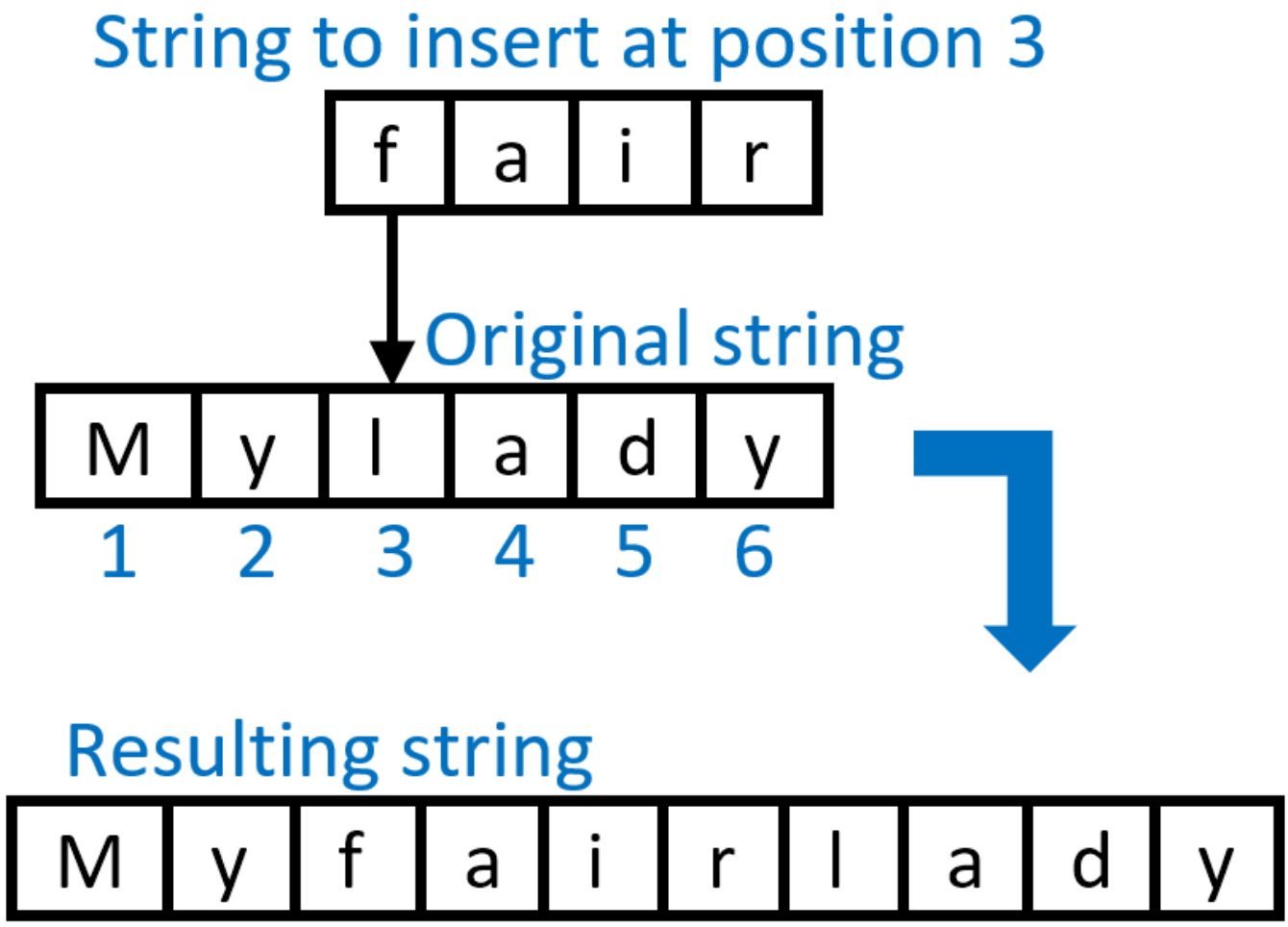
Inserting Sub string into String

Inserting a sub string in between any given input string is not possible directly as String is immutable.

But it can be possible by creating a new string in which updated content can be put.

Steps to make updated string after performing insertion:

1. Create an empty list
2. Insert first 'n-1' characters from given string where 'n' is the location at which sub string to be inserted
3. Append sub string
4. Put remaining characters of given string from 'n' to the 'last character'
5. Convert the list into string to get final output



Program 14: Program to insert sub string in a given string at particular position

```
# To insert a sub string in a string
```

```
st = input("Enter a string:")
sub = input("Enter sub string which is to be inserted:")
n = int(input("Enter position number:"))

# To get the correct position of index for insertion
n = n - 1

# Find the length of string and substrings
len1 = len(st)
len2 = len(sub)

# Take an empty list to form a new string
st1 = [ ]

# Append first n-1 characters from given string to empty list
for i in range(n):
  st1.append(st[i])

# Append sub string to list
for i in range(len2):
  st1.append(sub[i])

# Append the remaining characters from given string to list
for i in range(n,len1):
  st1.append(st[i])

# Convert list into string using join() method
updated = "".join(st1)

# Display updated string
print("String after insertion:",updated)
```

```
Enter a string:This python class
Enter sub string which is to be inserted:is
Enter position number:6
String after insertion: This ispython class
```

✓ 10s completed at 19:58