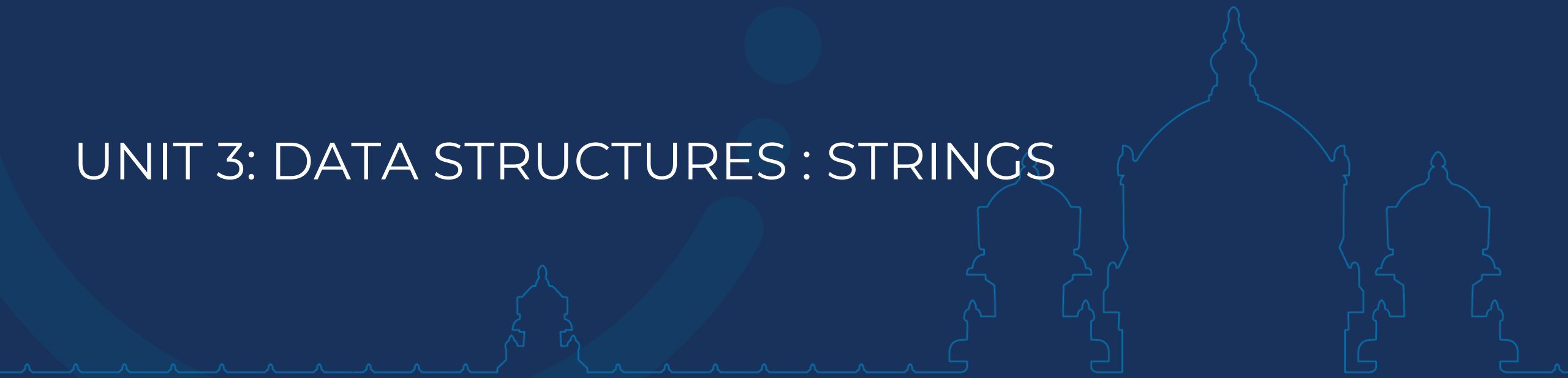
Python Programming

KUMARAGURU college of technology
character is life

KCED
Learning Augmented

UNIT 3: DATA STRUCTURES : STRINGS

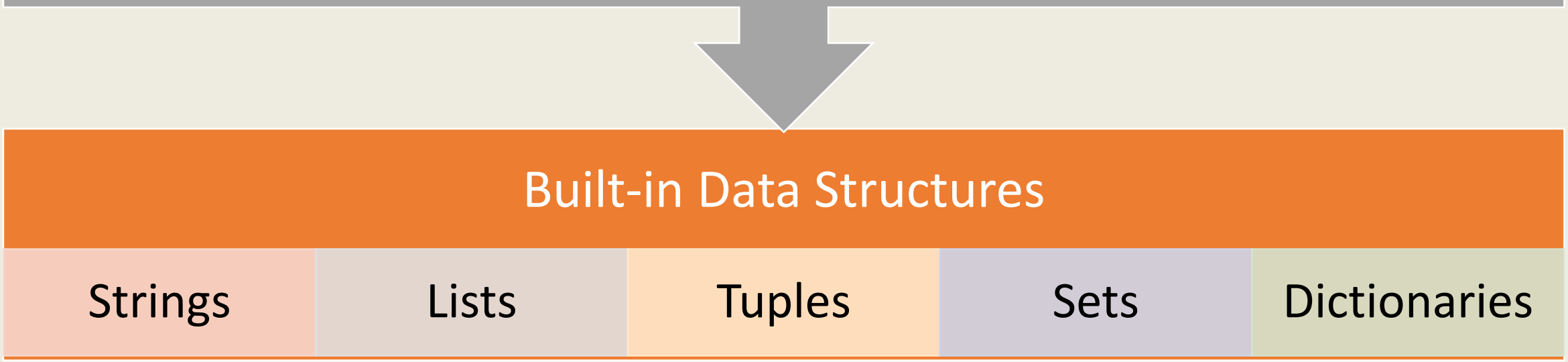


Overview of this lecture

- Introduction
- Defining a String
- Accessing a String
- String Operators and Operations
- Python Built-in Functions for String
- Python Built-in Methods for String
- Applications of String

Data Structure

Data Structure is a way to store and organize data so that it can be used efficiently



Introduction

A string in Python is sequence of alphanumeric characters which are grouped together to form a meaningful string

Character is a basic unit and building block for creating a string

String is an object of the *str* class

Creating a string

Method 1

Using constructor of the string class

#**Creating a string**

S1=str() #creates empty string

S2=str(“Welcome”) # creates string object for welcome

Method 2:

S1=“ “

S2=“welcome”

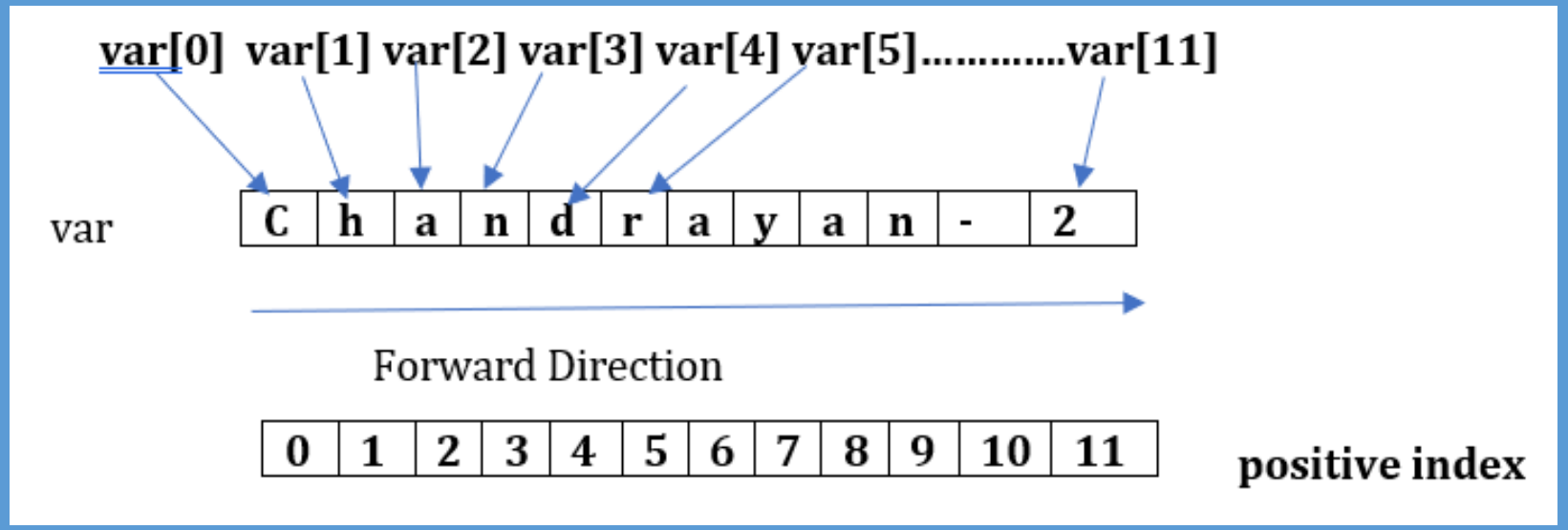
Example

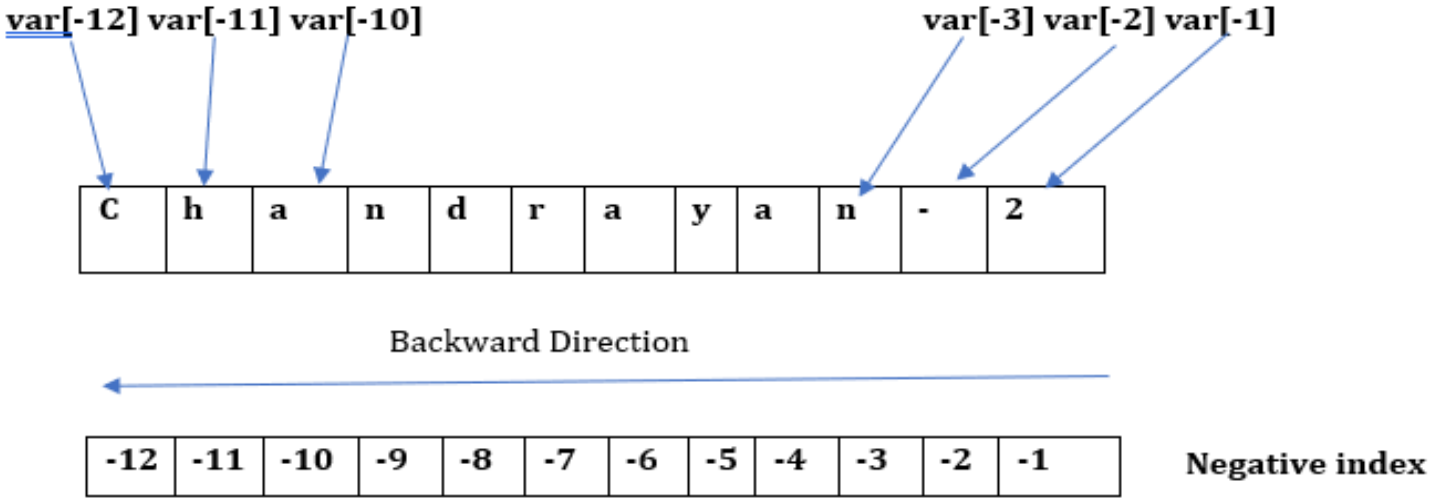
```
# all of the following are equivalent
my_string = 'Hello‘
my_string = "Hello“
my_string = "'Hello“‘

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
             the world of Python"""“
```

Representation of string

- A string can be considered as array of characters and it can be represented as sequential arrangement of characters in an array.
- Elements/characters of the string is accessed using its index value.
- The index value is always an integer, we can't use float or any other datatypes as index value.
- Python supports both positive and negative index

Accessing array elements in Forward direction using Positive index





Accessing array elements in backward direction using negative index

Strings are Immutable

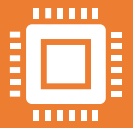
Strings are immutable means the contents of the string cannot be changed after it is defined or created

Assume the following string:
>>> String="Amplify hope"
>>> print(String)
Amplify hope

Now, let's to change the 3rd character of the original string.
>>> String[2]='P' # Assign the 'P' to 3rd index value.
It shows the following error.

```
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    String[2]='P'
TypeError: 'str' object does not support item assignment.
```

Accessing a string



Indexing -Accessing individual characters using index operator [].

Slicing – Range of characters or substring of a string can be accessed using slicing.

Traversing using loop. (for /while loop)

Accessing a String: Indexing

```
>>> var="Chandrayan-2"
>>> var
'Chandrayan-2'
>>> var[0] #To access the first character
'C'
>>> var[5] # To access the sixth character
'r'
```

Following example demonstrates the **index out of range error** when you try to the 13th character which does not cover in the index range. The index range for the variable var is 0 to 11.

```
>>> var[12]
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    var[12]
IndexError: string index out of range
```

Remember:

- Index values are zero-based.
- The index value for first character in string is always 0.
- This rule is common for slicing and indexed accessing.

Accessing a String: Indexing-Negative index

- Individual characters can be accessed using negative index.
- The index range for backward direction is from -1 to -12, where -1 represents the last character and -12 represents the first character of the string.

Consider var="Chandrayan-2"

```
>>> var[-1] # Access the last element using negative index
'2'
>>> var[-2] # Access last but one character using negative index
'-'
>>> var[-12]# Access the first character
'C'
```

Accessing a String: Slicing

- The slicing refers to access the range of characters or extract the substring from the string.
- The colon operator "**:** **"** is used with the index operator [] along with positive or negative indexing.

Syntax:
String_obj[start_index :end_index: step_size]

- **start_index -** specifies starting index position of substring. It is optional parameter, if not specified, it takes the value 0 by default.
- **end_index** - specifies end index position of substring. Result excludes character at end index. It is optional parameter, if it is not specified, it takes end of string by default.
- **step_size -** an integer value which specifies increment step count for index position. It is optional parameter. If it is not specified, it increments the index value by 1.

Accessing a String: Slicing -Example

var="Chandrayan-2"

Access the string in forward direction using positive index.

```
>>> var[0:3] #To access characters from index value 0 to 2,excluding the character at 3rd postion
'Cha'
>>> var[4:10] # Access the characters from 4th to 9th character,excluding 10th character
'drayan'
```

Access the string in backward direction using negative index.

```
>>> var[-9:-3]
'ndraya'
>>> var[-12:-5]
'Chandra'
```

Example – slicing

V	I	K	R	A	M
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Consider Name= ‘ VIKRAM’

S.NO	Slicing notation	Output	Explanation
1	Name [0:2]	‘VI’	Extracts from 0^{th} index position to 1^{st} index position. The character at index position 2 is excluded.
2	Name [2:5]	‘KRA’	Extracts characters from 2^{nd} position to 4^{th} position
3	Name [:4]	‘VIKR’	Extracts the character from the beginning to 3^{rd} position
4	Name [:]	‘VIKRAM’	If the start and end position is not specified, it retrieves the entire string.
5	Name [1: -1]	‘IKRA’	This notation excludes the first and last character and print the substring.
6	Name [2:]	‘KRAM’	Extract from 2^{nd} position to end of the string
7	Name [0:5:2]	‘VKA’	From index 0 to index 5 with step size 2(increment by 2).
8	Name [: : 3]	‘VR’	From index 0, extract characters with step size 3,until the end of the string.

Accessing a String: Traversing using Loops

- For loop and while loops can be used to traverse/ access all characters in a string.

```
s="Python Programming"
for character in s:
    print(character, end="")
```

Output: Python Programming

```
s=" Python Programming "
for index in range(0,len(s)):
    print(s[index],end="")
```

Output: Python Programming

```
s=" Python Programming "
index=0
while(index <len(s)):
    print(s[index],end="")
    index=index+1
```

Output: Python Programming

String Operators and Operations

S.No	Operator	Operator name	Description
1	+	Concatenation	Concatenates (joins) two strings
2	*	Repetition	Repetition operator(*) is used to concatenate the same string multiple times
3	[]	Index operator	Retrieve the specific character at the index value
4	[:]	Slicing operator	Extract the range of characters
5	in	Membership operator	Returns true if character exists in the given string
6	not in	Membership operator	Returns true if character does exists in the given string.
7	>,<,<=,>=,==, !=	Comparison operators	These operators are used to compare the strings.

Concatenation (+ operator)

Used to join two strings

```
>>> s1="welcome "
>>> s2="to kct"
>>> s1+s2
'welcome to kct'
```

Repetition * operator

- Used to concatenate the same string multiple times
- Repetition operator

s1=“welcome“

>>> s1*3

'welcomewelcomewelcome'

Membership (in and not in operator)

Used to check whether a string is present in another string

```
>>> s1="Kumaraguru college of technology"
```

```
>>> "college" in s1
```

```
True
```

```
>>> "engineering" in s1
```

```
False
```

```
>>> "engineering" not in s1
```

```
True
```

Comparison Operator

String comparison

==, <,>, <=, >=, !=

s1="abcd"

s2="ABCD"

s1>s2

True

s1<s2

False

s1="abc"

s2="abc"

s1==s2

True

s1>=s2

True

s2="abc".upper()

s1>=s2

True

s1>s2

True

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

KUMARAGURU college of technology
character is life

KCED
Learning Augmented

LEARNING AUGMENTED

Python Built-in Functions for String

S.no	Function	Description
1	len(string)	To find the length of the string
2	min(s)	To find the character which has minimum unicode character
3	max(s)	To find the character which has maximum unicode character
4	sorted(str)	The sorted() method sorts the elements of a given iterable in a specific order(list) - Ascending or Descending. sorted(str,reverse=True)-descending order

Example

Consider s1="Angry Bird"

```
>>> s1
'Angry bird'
>>> len(s1) # To find the length of the string s1
10

>>> max(s1)
'y'

>>> min(s1)
' '
```

```
>>> sorted(s1)

[' ', 'A', 'b', 'd', 'g', 'i', 'n', 'r', 'r', 'y']
```

Python Built-in Methods for String

- Python offers 40 + built-in methods for string handling and they can be classified into five types:
  - Conversion methods
  - String test methods
  - Search methods
  - Substitution methods
  - Miscellaneous methods

Conversion methods - Examples

```
>>> s="The Bird is powered by its own Life and by its Motivation"
>>> print(s)
The Bird is powered by its own Life and by its Motivation
>>> s.capitalize() # Captialize the first character of the string
'The bird is powered by its own life and by its motivation'

>>> s.lower() # Converts the entire string into lowercase
'the bird is powered by its own life and by its motivation'

>>> s.upper()
'THE BIRD IS POWERED BY ITS OWN LIFE AND BY ITS MOTIVATION'

>>> s.swapcase() # Changes uppercase character into lowercase and lowercase character into uppercase
'tHE bIRD IS POWERED BY ITS OWN lIFE AND BY ITS mOTIVATION'

>>> s.title() # Title case
'The Bird Is Powered By Its Own Life And By Its Motivation'
```

Testing Strings

```
>>> s='robotics' # Original String
>>> s.islower()  # Checks whether the string object s is in lowercase
True
>>> s='PYTHON'    #Original String
>>> s.isupper()   # Checks  whether the string object s is in uppercase
True
>>>  s='india2020' # String with digits
>>>  s.isdigit()    # Checks for the digits in the string
False
>>>  s='2020'
>>>  s.isdigit()
True
>>> s='Python3.4.8' # Input String
>>> s.isalpha()  #Checks whether the input string has only alphabets.
False
>>> s='india2020' #Input String
>>> s.isalnum()     #Checks whether the input string has alphabets and numbers.
True
```

Searching Methods

1. find(substr)- Find() searches for the substring in the input string.

Syntax- str_obj.find(substr)

Example 1- find() without start and end index value for search

```
>>> s= "Small aim is a crime" # Input String
>>> s.find('a')                # Search the character 'a' and return its index value
2
```

Example 2- if the sustring is not found, it returns the value -1

```
>>> s= "Small aim is a crime" # Input String
>>> s.find('I')               # check for the substring 'I
-1
```

As we know Python is case sensitive,though lower case 'i' is there in the input string, it returns -1.

Searching Methods

2. **index(str)**- This method is also to search for a substring .It is similar to the find() method, only difference is that it raises 'ValueError' exception if 'substr' doesn't exist.

Syntax- str_obj.index(str)

Example 1:

```
>>> s= "Small aim is a crime" # Input String
>>> s.index('im')            #search for the substring 'im
7
```

Example 2:

```
>>> s= "Small aim is a crime" # Input String
>>> s.index('I')             #search for the substring 'I'
Traceback (most recent call last):
  File "<pyshell#118>", line 1, in <module>
    s.index('I')            #search for the substring 'I'
ValueError: substring not found
```

Searching Methods

3. count(str,i ,j):- count() counts the occurrences of a substring in the input string. This method takes three arguments. First argument is the substring to be searched,second argument i-start the search from index position i ,third argument j-refers search ends at j^{th} index. Second and third arguments are optional,if omitted ,it searches in the entire input string.

Syntax:- str_obj.count(str)

Example 1:- count() without start and end index value

```
>>> s= "Small aim is a crime" # Input String
>>> s.count('m') # Count the occurences of 'm'
3
```

Example 2:- count() with start and end index value

```
>>> s.count('m',5,10) # Count the occurences of 'm' from the index value 5 to 9
1
```

Searching Methods

endswith(suffix, beg=0, end=len(string))

- Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise

```
>>> s1="Hello world"
>>> s1.endswith("world")
True
```

startswith(str, beg=0,end=len(string))

- Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.

```
>>> s1.startswith("python")
False
```

Substitution methods

1. **replace (old,new)**- This method replaces all the occurences of the substring 'old' with 'new'. count is an optional argument. If count argument is specified, it replaces only the number of occurences specified in the count. if count is omitted,it replaces all the occurences of 'old' substring.

Syntax- str_obj.replace(oldStr,newStr)

Example:

```
>>> s= "Small aim is a crime" # Input String
>>> s.replace('a','A') # Replace all the occurences of 'a' with 'A'.
'SmAll Aim is A crime'
```

Example 2:

```
>>> s
'Small aim is a crime'
>>> s.replace('a','A',2) # Replace only two occurences of 'a' with 'A'.
'SmAll Aim is a crime'
```

Substitution methods

2. **split()** -This method splits the input string into list of words/substring.

Syntax- str_obj.split()

```
>>> s= "Small aim is a crime" # Input String
>>> s.split() #Split into list of substrings
['Small', 'aim', 'is', 'a', 'crime']
```

3. **join()**-This method is combine the strings in a list to single string. You need to specify the seperator to be used between strings.

Syntax- Separator.join(str_obj)

The Separator is a object which indicates the separator that is to used to join the strings. Assume that string **s** is a list which has the following strings:
s=['small','aim','is','a','crime'].

```
>>> sep=' '
>>> sep.join(s)
'small aim is a crime'
```

The above statement includes space as a separator to join the strings in a list.

Other methods

- **lstrip() -** Removes all leading whitespace in string.

```
>>> s1="  hey cool"
>>> s1.lstrip()
'hey cool'
```

- **rstrip() -** Removes all trailing whitespace of string.

```
>>> s2="hi world   "
>>> s2.rstrip()
'hi world‘
```

- **strip() -** Performs both lstrip() and rstrip() on string.

```
>>> s="  hell "
>>> s.strip()  // removes left and right strip
'hell'
```

Formatting Methods

rjust(width,fillchar)

- Returns a space-padded string with the original string right-justified to a total of width columns.

Example-1

```
txt = "banana"
x = txt.rjust(20)
print(x, "is my favourite fruit.")
```

Output:

```
              banana is my favourite fruit.
```

Example-2

```
txt = "banana"
x = txt.rjust(20, "O")
print(x)
```

output

```
OOOOOOOOOOOOOObanana
```

Formatting Methods

ljust(width, fillchar)

- Returns a space-padded string with the original string left-justified to a total of width columns.

Example:

```
txt = "banana"

x = txt.ljust(20)

print(x)
```

Output

banana

center(width, fillchar)

Returns a space-padded string with the original string centered to a total of width columns.

Example:

```
str = "Hello World"
str2 = str.center(20,"$")
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

Output:

Old value: Hello Java
New value: $$$$$Hello Java$$$$$

.format()

Programmers can include %s inside a string and follow it with a list of values for each %

print(“My name is %s I am studying in %s"%(“Alice","BE"))

Output

‘My name is Alice I am studying in BE’

Instead of % we can use {0}, {1} and so on

Print(“My name is {0} I am studying in {1}".format("John","BE"))

Output

‘My name is John I am studying in BE'

Applications of Strings

- Searching for a substring
- String substitution
- Checking for presence of alphabets / number/ decimals/digits in a string.
- Upper case/lower case conversion
- Splitting or joining strings.
- To carry out any kind of character transformation in a string. Ex: Encryption/Decryption operation.

KUMARAGURU college of technology
character is life

KCED
Learning Augmented

Example Programs on Strings

Write a program to prompt your friend name through input statement and store it in a variable myFriend. Write a print statement to greet them with their name.

Example is given below:

Input: What's your friend name?

Ajay

Output :

Hello Ajay !!!

Program:

```
# Take the input from the user
name=input("What's your friend name:    ")
print("Hello," + name + "!!!")
```

Output:

What's your friend name: Ajay

Hello, Ajay!!!

Write a program to ask the programmer to enter the username for Olympiad exam. The user name should be between 3 to 15 characters. If the length is not between 3 to 15 characters, print that it is a invalid username.

Program:

```
# This program is to check whether the username is 3 to 15 characters long
uName=input("Enter the username: ") #input is stored in the variabe uName
length=len(uName)   # len() to find the length of the user name
if (length>3 ) and (length<16): #Verifying length
    print("Valid Username..")
else:
    print("Invalid Username")
```

Output 1:

Enter the username: Tenali

Valid Username..

Output 2:

Enter the username: We

Invalid Username

Write a program to count the number of lower-case letters in a string:

```
# This program is to count the number of lower case letters in a string
strVar=input ("Enter the string :")
lCount= 0    #Defining a variable to count the lowercase letters
for i in strVar:
    if(i.islower()):    #Check each character in the string is in lower case
        lCount=lCount+1
# To Print the number of lower-case letters
print ("The number of lower case letters in string ",lCount)
```

Output:

Enter the string : Your Best Teacher is Your Last Mistake

The number of lower case letters in string 26

Write a Program to count the number of vowels in the given string

Solution 1

```
# This program is to count the vowels in a string
strVar=input("Enter the string : ")
vCount=0  # Counter variable
str1=strVar.lower()
# For loop to iterate over each character of string
for i in str1:
    if (i == 'a' or i == 'e' or i == 'i' or i == 'o' or i == 'u'):
        vCount=vCount+1
print("The number of vowels are:" , vCount)
```

or

Solution 2

```
# This program is to count the vowels in a string
strVar=input("Enter the string : ")
vCount=0  # Counter variable
str1=strVar.lower()
vCount=str1.count('a')+ str1.count('e')+ str1.count('i')+
str1.count('o')+str1.count('u')
print("The number of vowels are:" , vCount)
```

```
Enter the string : Hello
The number of vowels are: 2
```

Write a Program to check whether given string is palindrome or not

```
string=input("Enter the string:")
rev_string=string[::-1]
print("Reversed string: ",rev_string)
if (string==rev_string):
    print("Yes,Given string is a palindrome")
else:
    print("No,Given string is not a palindrome")
```

```
Enter the string:madam
Reversed string:  madam
Yes,Given string is a palindrome
```

Summary

- String is an immutable data object.
- String operations results in creating a new string which is stored in new location.
- String data structure has its own set of operators to handle and manipulate strings.
- The index operator [] is used to access individual characters in a string.
- The for and while loop can also be used for traversing the string.
- The concatenation operator (+), Repetition operator (+),slicing operators(:) are some useful operators.
- Python has some built-in methods and functions to perform various string operations like string substitution, concatenation, substring searching, conversion of characters etc.

Tips

Blue light is basically the rays that are emitted by the digital screen, which may cause severe health problems. Flux software for PCs, Twilight app for Smartphones can act as blue light filters.

Microcosm
The Co-existing Architecture

Kumaraguru Eco Campus

Annually, the **4200 trees** inside the Kumaraguru Campus, on an average, releases **10,92,000 pounds** of oxygen inhaled by **42,000 humans**.

Wellness 360

Find ways to structure and optimize your time for when and where you learn best and keep your **learning on track**

KUMARAGURU college of technology

KCED
Learning Augmented