Strings

When characters like letters, digits, and symbols are arranged in a sequence, it is known as a string. The usual string object is an immutable (unchangeable) sequence of characters accessed by offset (position).

Strings are immutable, but you can update the value of a string by assigning a new string to the same string variable.

In [1]:
```
s="welocme LJU"
print(id(s))
print(s)
#you can update the value of a string by assigning a new string to the same string variable.
s=s+"2022"
print(id(s))
print(s)
```

```
2243025576304
welocme LJU
2243025537584
welocme LJU2022
```

We will cover the following topics in this chapter:

Creating a string

Accessing values in string

Changing or deleting a string

String operations

Build-in string methods

strings are a sequence of Unicode Characters

Unicode string

All text is unicode text, including the text encoded with one character per byte (8 bits) in the ASCII scheme. Python supports richer character sets and encoding schemes with unicode, that is, strings that may use multiple bytes to represent characters in memory, and which translate text to and from various encodings on files.

Creating a string

We can create strings by enclosing characters within a single quote or double quotes. Python supports triple quotes, but these are generally used to represent multiline strings and docstrings.

In Python strings, single- and double-quote characters are interchangeable, i.e., string literals can be expressed in two single or two double quotes. Here, the two representations work in the same way and return the same type of object.

The reason for supporting the two representations is that it allows programmers to enclose a quote character of the other type within a string without a backslash. It also helps programmers to enclose a single-quote character in a string enclosed in double-quote characters, and vice-versa.

In [2]:
```
s="welcome"
print(s)
```

```
welcome
```

In [4]:
```
s='welcome'
print(s)
```

```
welcome
```

In [5]:
```
#for multiline string
s='''welcome'''
print(s)
```

```
welcome
```

In [6]:
```
s="""hello
welcome"""
print(s)
```

```
hello
welcome
```

In [7]:
```
s='it's blowing outside'
print(s)
```

```
  File "C:\Users\VISHAL\AppData\Local\Temp\ipykernel_13484\1923883316.py", line 1
    s='it's blowing outside'
          ^
SyntaxError: invalid syntax
```

In [9]:
```
s="it's blowing outside"
print(s)
```

```
it's blowing outside
```

In [11]:
```
a=10
print(a,"--",id(a),"--",type(a))
a=str(a)
print(a,"--",id(a),"--",type(a))
```

```
10 -- 2242939873872 -- <class 'int'>
10 -- 2243025646768 -- <class 'str'>
```

Accessing substring from a string

Python does not allow a character type; these are considered as strings of length one, so they are also considered substrings.

We can access individual characters using indexing and with different characters using the process of slicing. The index always starts from zero, but an IndexError may arise while trying to access a character out of the index range. The index is always an integer; we cannot use float or other data types; this will lead to TypeError.

Python allows negative indexing for its sequences. For example, the index of -1 indicates the last item, -2 the second last item, and so on. Python also allows you to access a range of items in a string using the slicing operator (colon).

Slice expressions support the optional third index, which is used as a step (sometimes called stride). Steps are added to the index of each item extracted. Slices are now X[I:J:K], which means that all items in X are removed from offsets I through J*1, by K. K is generally set to +1, which is why all items in a slice are generally extracted from left to right. However, the third limit can be used to skip items or reverse their order if you specify an explicit value. X[1:10:2] will fetch every other item in X from offsets 1–9. It will collect items at offsets 1, 3, 5, 7, and 9. Since the first and second limits are set to 0 and the length of the sequence, respectively, X[::2] receives all items from the beginning to the end of the sequence.

In [12]:
```
# posative Indexing (left to right) (0 to n-1 here n is number of chracter in string)
s="hello world"
print(s[0])
print(s[1])
print(s[40])
```

```
h
e

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\3584315270.py in <module>
      3 print(s[0])
      4 print(s[1])
----> 5 print(s[40])

IndexError: string index out of range
```

In [13]:
```
# negative Indexing (right to left) (start from -1 to -n)
s="hello world"
print(s[-1])
print(s[-2])
print(s[-40])
```

```
d
l

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\606380217.py in <module>
      3 print(s[-1])
      4 print(s[-2])
----> 5 print(s[-40])

IndexError: string index out of range
```

In [19]:
```
#slicing
s="hello world"
print("1-",s[0:6])
print("2-",s[2:3])
print("3-",s[2:])
print("4-",s[:-1])
print("5-",s[:])
print("6-",s[0:7])
print("7-",s[2:-1])
print("8-",s[:-2])
print("9-",s[1:-1])
print("10-",s[-1:-5])
```

```
1- hello
2- l
3- llo world
4- hello worl
5- hello world
6- hello w
7- llo worl
8- hello wor
9- ello worl
10-
```

In [22]:
```
#slicing with step
s="hello world"
print("1-",s[0:6:2])
print("2-",s[2:4:2])
print("3-",s[2::-1])
print("4-",s[:-1:3])
print("5-",s[::2])
print("6-",s[0:7:3])
print("7-",s[2:-1:2])
print("8-",s[:-2:2])
print("9-",s[1:-1:3])
print("10-",s[-1:-5:-2])#negative index starting number grater than ending
print("11-",s[-5::])
```

```
1- hlo
2- l
3- leh
4- hlwl
5- hlowrd
6- hlw
7- lowr
8- hlowr
9- eoo
10- dr
11- world
```

In [23]:
```
#slicing with reverse
s="hello world"
print(s[::-1])
```

```
dlrow olleh
```

Editing and deleting in string

Strings are immutable, which implies that elements of a string once assigned cannot be changed. Python allows you to set the same name to different strings.

```
In [24]: s="hello world"
         s[0]="H"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\4116805414.py in <module>
      1 s="hello world"
----> 2 s[0]="H"

TypeError: 'str' object does not support item assignment
```

```
In [25]: s="hello world"
         print(id(s))
         del s
         print (id(s))
```

```
2243025887088

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\4085130183.py in <module>
      2 print(id(s))
      3 del s
----> 4 print (id(s))

NameError: name 's' is not defined
```

```
In [27]: s="hello world"
         del s[:]
         print(s)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\496973270.py in <module>
      1 s="hello world"
----> 2 del s[:]
      3 print(s)

TypeError: 'str' object does not support item deletion
```

Operations on Strings

Arithmetic Operations

Relational Operations

Logical Operations

Loops on Strings

Membership Operations

```
In [28]: print("gandhinagar"+" "+"delhi")
```

```
gandhinagar delhi
```

```
In [46]: print("*"*50)
```

```
**************************************************
```

```
In [47]: str="hello"
         print(id(str))
         str+="student"
         print(str)
         print(id(str))
```

```
2243021959920
hellostudent
2243025831920
```

Relational Operations

In [31]:
```
"delhi"=="gandhinagar"
```
Out[31]:
```
False
```

In [32]:
```
"delhi"=="delhi"
```
Out[32]:
```
True
```

In [33]:
```
'delhi' != 'delhi'
```
Out[33]:
```
False
```

In [35]:
```
'mumbai' > 'pune'
# lexiographically
```
Out[35]:
```
False
```

In [36]:
```
'Pune' > 'pune'
```
Out[36]:
```
False
```

Logical Operations

In [37]:
```
'hello' and 'world'
```
Out[37]:
```
'world'
```

In [38]:
```
'hello' or 'world'
```
Out[38]:
```
'hello'
```

In [39]:
```
'' and 'world'
```
Out[39]:
```
''
```

In [40]:
```
'' or 'world'
```
Out[40]:
```
'world'
```

In [41]:
```
not 'hello'
```
Out[41]:
```
False
```

In [43]:
```
not ""
```
Out[43]:
```
True
```

Loops on Strings

In [48]:
```
str="hello "
index=0
for i in str:
    print('str [',index,"] = ",i)
    index+=1
```
```
str [ 0 ] =  h
str [ 1 ] =  e
str [ 2 ] =  l
str [ 3 ] =  l
str [ 4 ] =  o
str [ 5 ] =  
```

In [49]:
```
str="hello "
for i in str:
    print("python")
```
```
python
python
python
python
python
python
```

In [55]:
```
str="hello "
for index,j in enumerate(str):
    print(index,"--",j)
```

```
0 -- h
1 -- e
2 -- l
3 -- l
4 -- o
5 --
```

In [56]:
```
str="hello "
for i,j in enumerate(str):
    print(i,"--",j)
```

```
0 -- h
1 -- e
2 -- l
3 -- l
4 -- o
5 --
```

Membership Operations

In [57]:
```
'D' in 'delhi'
```

Out[57]:
```
False
```

In [59]:
```
"l" in "mango"
```

Out[59]:
```
False
```

In [60]:
```
"l" not in "mango"
```

Out[60]:
```
True
```

Common Functions

len

max

min

sorted

String len() method

This len() method returns the length of the string.

Syntax:

len(str)

Parameters

NA

return

integer value

String max() method

The max() method returns the maximized alphabetical character from the string

Syntax:

max(str)

Parameters

str - From this string, the maximum alphabetical character should be returned

return

give maximized alphabatical character

String min() method

The min() method returns the minimum alphabetical character in the string str.

Syntax:

min(str)

Parameters

str - The string from which the minimum alphabetical character needs to be returned.

Return type

This method returns the minimum alphabetical character from the str string.

sorted()

The sorted() function returns a sorted list of the specified iterable object.

You can specify ascending or descending order. Strings are sorted alphabetically, and numbers are sorted numerically.

Note: You cannot sort a list that contains BOTH string values AND numeric values.

In [63]:
```
str="hello word"
print(len(str))#give total character of string
print(max(str))#give biggest character according to Ascii value
print(min(str))#give smallest character according to Ascii value here space has min ascii value
str="helloword"
print(min(str))
```

```
10
w

d
```

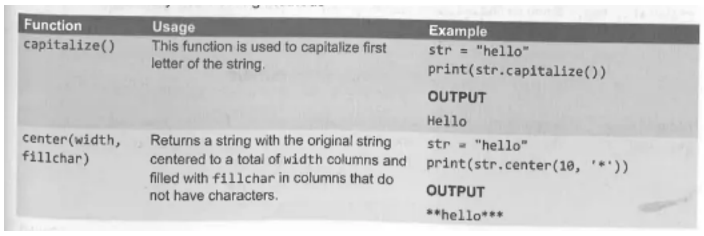
In [64]:
```
str="hello word"
print(sorted(str))
```

```
[' ', 'd', 'e', 'h', 'l', 'l', 'o', 'o', 'r', 'w']
```

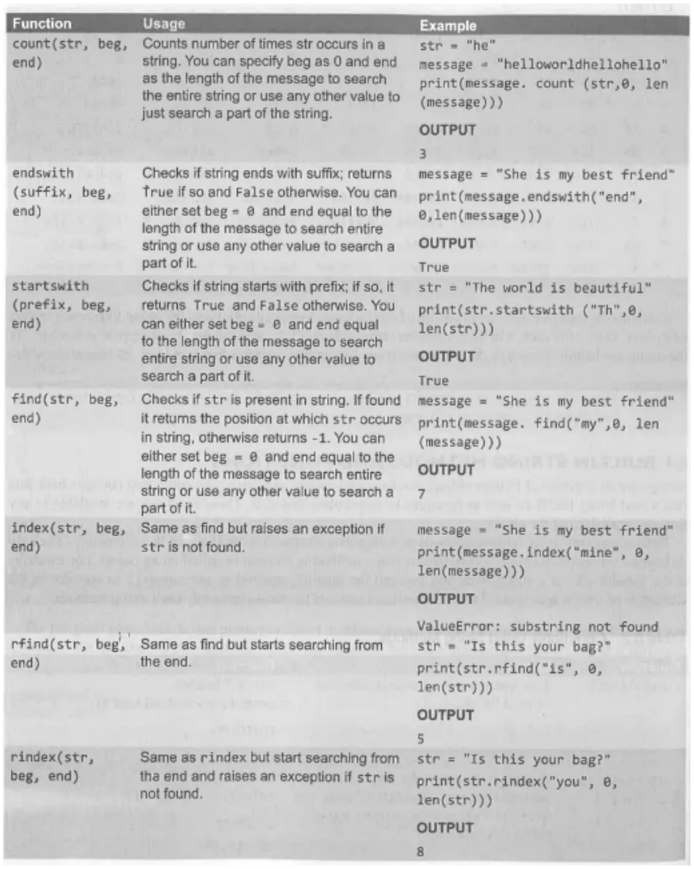
In [65]:
```
str="hello word"
print(sorted(str,reverse=True))
```

```
['w', 'r', 'o', 'o', 'l', 'l', 'h', 'e', 'd', ' ']
```

common function



Function	Usage	Example
`capitalize()`	This function is used to capitalize first letter of the string.	`str = "hello"` `print(str.capitalize())` **OUTPUT** `Hello`
`center(width, fillchar)`	Returns a string with the original string centered to a total of `width` columns and filled with `fillchar` in columns that do not have characters.	`str = "hello"` `print(str.center(10, '*'))` **OUTPUT** `**hello***`



Function	Usage	Example

`count(str, beg, end)`	Counts number of times str occurs in a string. You can specify beg as 0 and end as the length of the message to search the entire string or use any other value to just search a part of the string.	`str = "he"`
		`message = "helloworldhellohello"`
		`print(message. count (str,0, len (message)))`
		OUTPUT
		`3`

`endswith (suffix, beg, end)`	Checks if string ends with suffix; returns True if so and False otherwise. You can either set beg = 0 and end equal to the length of the message to search entire string or use any other value to search a part of it.	`message = "She is my best friend"`
		`print(message.endswith("end", 0,len(message)))`
		OUTPUT
		`True`

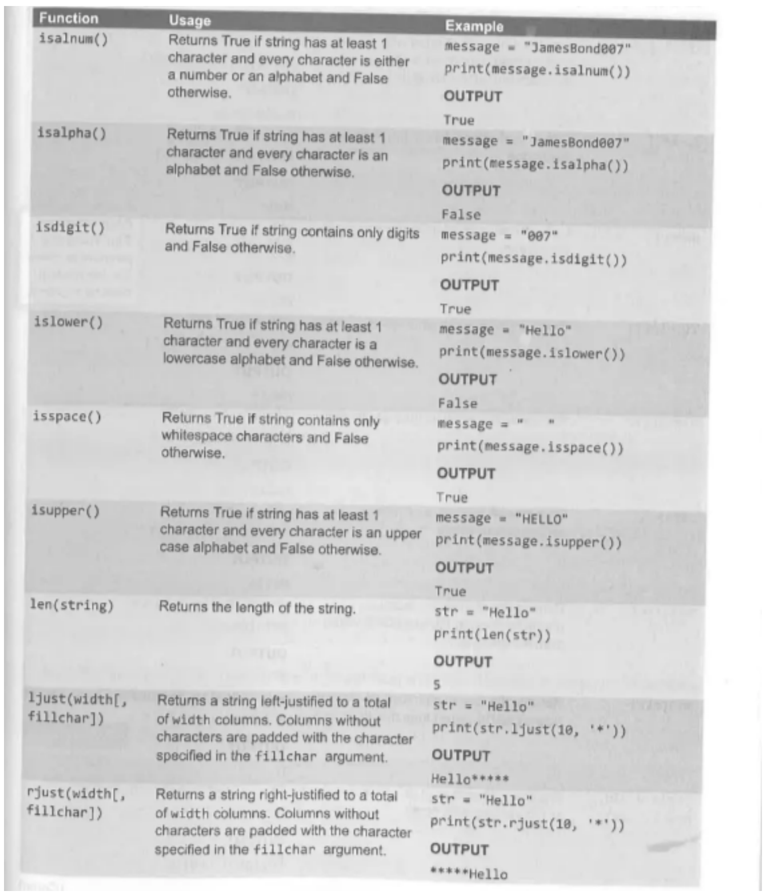
`startswith (prefix, beg, end)`	Checks if string starts with prefix; if so, it returns True and False otherwise. You can either set beg = 0 and end equal to the length of the message to search entire string or use any other value to search a part of it.	`str = "The world is beautiful"`
		`print(str.startswith ("Th",0, len(str)))`
		OUTPUT
		`True`

`find(str, beg, end)`	Checks if str is present in string. If found it returns the position at which str occurs in string, otherwise returns -1. You can either set beg = 0 and end equal to the length of the message to search entire string or use any other value to search a part of it.	`message = "She is my best friend"`
		`print(message. find("my",0, len (message)))`
		OUTPUT
		`7`

`index(str, beg, end)`	Same as find but raises an exception if str is not found.	`message = "She is my best friend"`
		`print(message.index("mine", 0, len(message)))`
		OUTPUT
		`ValueError: substring not found`

`rfind(str, beg, end)`	Same as find but starts searching from the end.	`str = "Is this your bag?"`
		`print(str.rfind("is", 0, len(str)))`
		OUTPUT
		`5`

`rindex(str, beg, end)`	Same as rindex but start searching from tha end and raises an exception if str is not found.	`str = "Is this your bag?"`
		`print(str.rindex("you", 0, len(str)))`
		OUTPUT
		`8`



Function	Usage	Example

`isalnum()`	Returns True if string has at least 1 character and every character is either a number or an alphabet and False otherwise.	`message = "JamesBond007"`
		`print(message.isalnum())`
		OUTPUT
		`True`

`isalpha()`	Returns True if string has at least 1 character and every character is an alphabet and False otherwise.	`message = "JamesBond007"`
		`print(message.isalpha())`
		OUTPUT
		`False`

`isdigit()`	Returns True if string contains only digits and False otherwise.	`message = "007"`
		`print(message.isdigit())`
		OUTPUT
		`True`

`islower()`	Returns True if string has at least 1 character and every character is a lowercase alphabet and False otherwise.	`message = "Hello"`
		`print(message.islower())`
		OUTPUT
		`False`

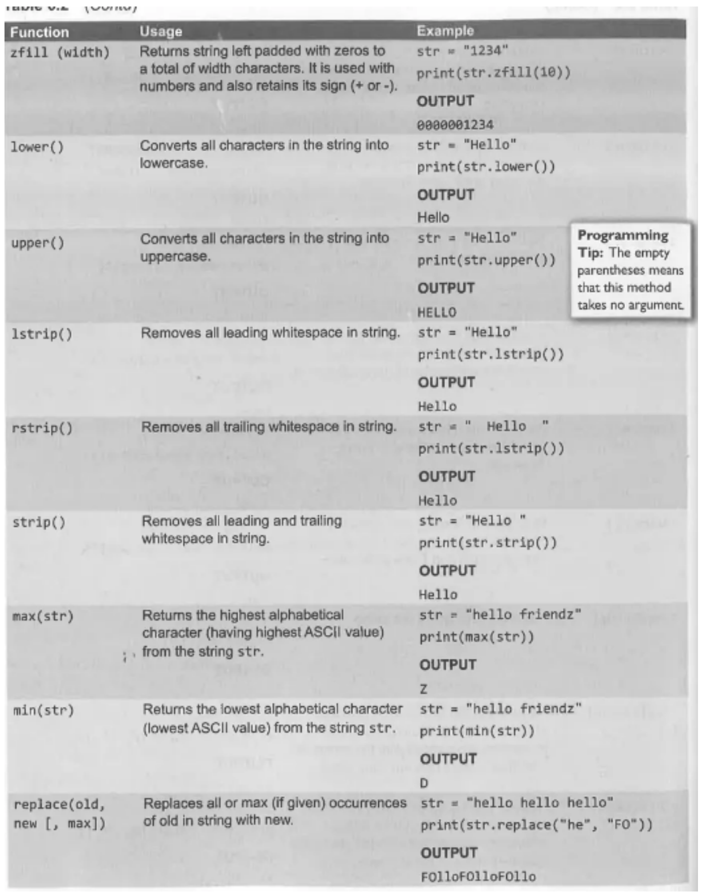
`isspace()`	Returns True if string contains only whitespace characters and False otherwise.	`message = "    "`
		`print(message.isspace())`
		OUTPUT
		`True`

`isupper()`	Returns True if string has at least 1 character and every character is an upper case alphabet and False otherwise.	`message = "HELLO"`
		`print(message.isupper())`
		OUTPUT
		`True`

`len(string)`	Returns the length of the string.	`str = "Hello"`
		`print(len(str))`
		OUTPUT
		`5`

`ljust(width[, fillchar])`	Returns a string left-justified to a total of width columns. Columns without characters are padded with the character specified in the fillchar argument.	`str = "Hello"`
		`print(str.ljust(10, '*'))`
		OUTPUT
		`Hello*****`

`rjust(width[, fillchar])`	Returns a string right-justified to a total of width columns. Columns without characters are padded with the character specified in the fillchar argument.	`str = "Hello"`
		`print(str.rjust(10, '*'))`
		OUTPUT
		`*****Hello`



Function	Usage	Example

`zfill (width)`	Returns string left padded with zeros to a total of width characters. It is used with numbers and also retains its sign (+ or -).	`str = "1234"`
		`print(str.zfill(10))`
		OUTPUT
		`0000001234`

`lower()`	Converts all characters in the string into lowercase.	`str = "Hello"`
		`print(str.lower())`
		OUTPUT
		Hello

`upper()`	Converts all characters in the string into uppercase.	`str = "Hello"`
		`print(str.upper())`
		OUTPUT
		`HELLO`

`lstrip()`	Removes all leading whitespace in string.	`str = "Hello"`
		`print(str.lstrip())`
		OUTPUT
		`Hello`

`rstrip()`	Removes all trailing whitespace in string.	`str = "  Hello  "`
		`print(str.lstrip())`
		OUTPUT
		`Hello`

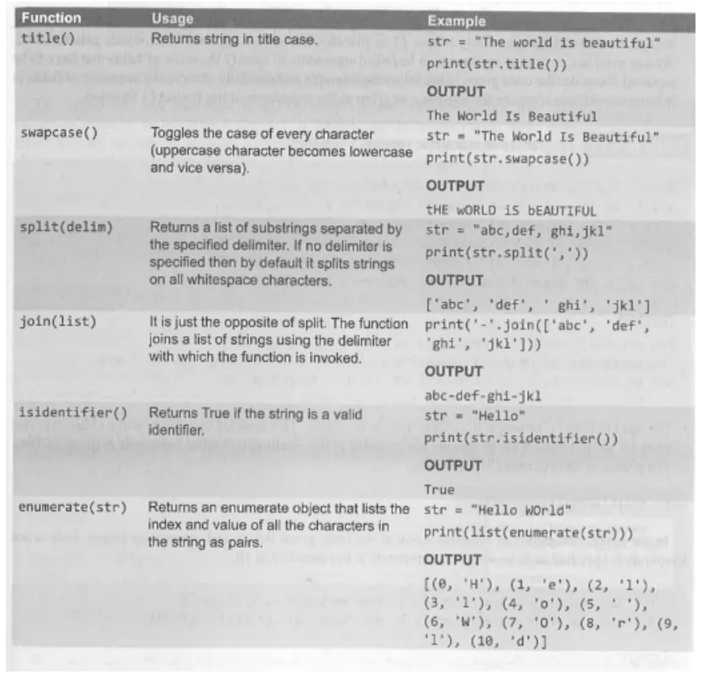
`strip()`	Removes all leading and trailing whitespace in string.	`str = "Hello "`
		`print(str.strip())`
		OUTPUT
		`Hello`

`max(str)`	Returns the highest alphabetical character (having highest ASCII value) from the string `str`.	`str = "hello friendz"`
		`print(max(str))`
		OUTPUT
		`z`

`min(str)`	Returns the lowest alphabetical character (lowest ASCII value) from the string `str`.	`str = "hello friendz"`
		`print(min(str))`
		OUTPUT
		`D`

`replace(old, new [, max])`	Replaces all or max (if given) occurrences of old in string with new.	`str = "hello hello hello"`
		`print(str.replace("he", "FO"))`
		OUTPUT
		`FOlloFOlloFOllo`

Programming Tip: The empty parentheses means that this method takes no argument.



Function	Usage	Example

`title()`	Returns string in title case.	`str = "The world is beautiful"`
		`print(str.title())`
		OUTPUT
		`The World Is Beautiful`

`swapcase()`	Toggles the case of every character (uppercase character becomes lowercase and vice versa).	`str = "The World Is Beautiful"`
		`print(str.swapcase())`
		OUTPUT
		`tHE wORLD iS bEAUTIFUL`

`split(delim)`	Returns a list of substrings separated by the specified delimiter. If no delimiter is specified then by default it splits strings on all whitespace characters.	`str = "abc,def, ghi,jkl"`
		`print(str.split(','))`
		OUTPUT
		`['abc', 'def', ' ghi', 'jkl']`

`join(list)`	It is just the opposite of split. The function joins a list of strings using the delimiter with which the function is invoked.	`print('-'.join(['abc', 'def', 'ghi', 'jkl']))`
		OUTPUT
		`abc-def-ghi-jkl`

`isidentifier()`	Returns True if the string is a valid identifier.	`str = "Hello"`
		`print(str.isidentifier())`
		OUTPUT
		`True`

`enumerate(str)`	Returns an enumerate object that lists the index and value of all the characters in the string as pairs.	`str = "Hello WOrld"`
		`print(list(enumerate(str)))`
		OUTPUT
		`[(0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'), (5, ' '), (6, 'W'), (7, 'O'), (8, 'r'), (9, 'l'), (10, 'd')]`

to see contents of string module

In [67]:
```python
import string
dir(string)
```

Out[67]:
```
['Formatter',
 'Template',
 '_ChainMap',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_re',
 '_sentinel_dict',
 '_string',
 'ascii_letters',
 'ascii_lowercase',
 'ascii_uppercase',
 'capwords',
 'digits',
 'hexdigits',
 'octdigits',
 'printable',
 'punctuation',
 'whitespace']
```

In [68]:
```python
str="hello"
print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute
__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__seta
ttr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expand
tabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islow
er', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

String capitalize() method

This method provides us a copy of the string with the first letter capitalized and the rest lowercase as the output.

Syntax:

str.capitalize()

Parameters

NA

Return value

string

In [70]:
```python
s="a branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.capitalize()
```

Out[70]:
```
'A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction'
```

String title() method

The title() method returns a copy of the string in which the starting characters of all the words are capitalized.

Syntax:

str.title()

Parameters

NA

Return value

string

In [71]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.title()
```
Out[71]:
```
'A Branch Is An Instruction In A Computer Program That Can Cause A Computer To Begin Executing A Different Instruction'
```

String lower() method

The lower() method returns a copy of the string if all case-based characters have been lowercased.

Syntax:

str.lower()

Parameters

NA

Return value

string

In [72]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.lower()
```
Out[72]:
```
'a branch is an instruction in a computer program that can cause a computer to begin executing a different instruction'
```

String upper() method

The upper() method returns a copy of the string in which all case-based characters have been uppercased.

Syntax:

str.upper()

Parameters

NA

Return type

Returns the string with all characters converted to uppercase

In [73]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.upper()
```
Out[73]:
```
'A BRANCH IS AN INSTRUCTION IN A COMPUTER PROGRAM THAT CAN CAUSE A COMPUTER TO BEGIN EXECUTING A DIFFERENT INSTRUCTION'
```

String swapcase() method

Swapping the case of the characters in a string is accomplished by the swapcase() method.

Syntax:

str.swapcase()

Parameters

NA

Return type

If all the case-based characters are swapped, then this method returns a copy of the string

In [74]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.swapcase()
```
Out[74]:
```
'a BRANCH IS AN INSTRUCTION IN A COMPUTER PROGRAM THAT CAN CAUSE A COMPUTER TO BEGIN EXECUTING A DIFFERENT INSTRUCTION'
```

String count() method

The count() method gives the number of occurrences of the substring sub within the range [start, end] as the output. Arguments start and end are optional, and they are interpreted the same as in slice notation.

Syntax: str.count(sub, start= 0,end=len(string))

Parameters

sub - This is the substring needed for searching

start - Search always starts from this index, and the first character usually starts from the index, so search starts from the index by default.

end - Search ends at this index; the first character always starts from index, and search ends at the last index in the default case

Return value Centered in a string with length width

```
In [77]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.count("i")

Out[77]: 9
```

```
In [78]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.count("i",15,50)

Out[78]: 3
```

String find() method

The find() method determines whether the given string str occurs in the string or in a substring of the string if the starting index beg and ending index end are given.

Syntax: str.find(str, beg=0 end=len(string))

Parameters

str - It is the string needed to be found

beg - It is the starting index of the string; by default, it is zero

end - It indicates the ending index; by default, it is equal to the length of the string.

Return type

Index if found and -1 otherwise

```
In [79]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.find("in")

Out[79]: 15
```

```
In [80]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.find("in",16,len(s))

Out[80]: 27
```

```
In [82]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.find("in",110,len(s))

Out[82]: -1
```

String rfind() method

The rfind() method returns the last index in which the str substring is found or -1 if no such index exists; optionally, the search is restricted to String

Syntax: str.rfind(str, beg=0 end=len(string))

Parameters

str – It specifies the string to be searched

beg - It is the starting index; by default, it is 0

end - By default, the ending index is equal to the length of the string

Return type

Index if found and -1 otherwise

```
In [83]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.rfind("in")

Out[83]: 106
```

```
In [87]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.rfind("in",5,100)

Out[87]: 90
```

```
In [88]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.rfind("in",0,7)

Out[88]: -1
```

String index() method

The index() method determines whether the string str occurs in string or in a substring of the string, given the starting index beg and ending index end. This method is similar to but it raises an exception if sub cannot be found.

Syntax: str.index(str, beg=0 end=len(string))

Parameters

Str: It specifies the string to be searched It is the starting index; by default, it is 0 The ending index, by default, is equal to the string length

Return value Index if and raises an exception if str is not found

```
In [89]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.index("in")

Out[89]: 15
```

```
In [90]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.index("in",16,len(s))

Out[90]: 27
```

```
In [93]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.index("in",110,len(s))

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\3918687989.py in <module>
      1 s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instructio
n"
----> 2 s.index("in",110,len(s))

ValueError: substring not found
```

String rindex() method

When rindex() is called, it returns the last index where the str substring has been found or raises an exception if no index exists; optionally the search is restricted to

Syntax: str.rindex(str, beg=0 end=len(string))

Parameters

str - It specifies the string to be searched

beg – The starting index; by default, it is 0

len – The ending index; by default, it is equal to the length of the string

return index if or not

```
In [94]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.rindex("in")

Out[94]: 106
```

```
In [95]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.rindex("in",5,100)

Out[95]: 90
```

```
In [96]: s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
         s.rindex("in",0,7)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13484\3313547346.py in <module>
      1 s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instructio
n"
----> 2 s.rindex("in",0,7)

ValueError: substring not found
```

String startswith() method

The startswith() method checks whether the string starts with str, optionally restricting the matching with the given indices start and end.

Syntax: str.startswith(str, beg=0,end=len(string))

Parameters

str - String to be checked

beg - This is the optional parameter to set start index to matching boundaries

end - This is the optional parameter to set end index of the matching boundary

Return type This method returns true if found matching with the string; otherwise, it returns false.

```
In [97]: 'my name is nitish'.startswith('my')

Out[97]: True

In [98]: 'my name is nitish'.startswith('1my')

Out[98]: False

In [99]: 'my name is nitish'.startswith('myn')

Out[99]: False

In [100]: 'my name is nitish'.startswith('my n')

Out[100]: True
```

String endswith() method

The boolean value true is returned when the string ends with the given suffix; otherwise, it returns false. Also, it restricts the matching with the given indices start and end, which are optional.

Syntax: str.endswith(suffix[, start[, end]])

Parameters

suffix – It can be a string or a tuple of suffixes to look for

start – It represents the start of the slice

end – It indicates the end of the slice

```
In [102]: 'my name is nitish'.endswith('sh')

Out[102]: True

In [103]: 'my name is nitish'.endswith('sho')

Out[103]: False
```

String isalnum() method

The isalnum() method checks whether the string contains alphanumeric characters.

Syntax: str.isa1num()

Parameters NA Return type This method returns true if all characters in the string are alphanumeric and there is at least one character; otherwise, it returns false.

```
In [104]: "vishalnot".isalnum()
Out[104]: True
```

```
In [105]: "12345698".isalnum()
Out[105]: True
```

```
In [106]: "vishnot123".isalnum()
Out[106]: True
```

```
In [107]: "vish not 123".isalnum()
Out[107]: False
```

String isalpha() method

The isalpha() method checks whether the string consists of alphabetic characters only.

Syntax: str.isalpha()

Parameters NA

Return type The method returns true if there is at least one character in the string and all characters are alphabetic; otherwise, it returns false.

```
In [108]: "vishalnot".isalpha()
Out[108]: True
```

```
In [109]: "12345698".isalpha()
Out[109]: False
```

```
In [110]: "vishnot123".isalpha()
Out[110]: False
```

String islower() method

You can check whether all case-based characters (letters) in a string are lowercase by calling the islower() method.

Syntax: str.islower()

Parameters NA

Return type

The method returns true if all cased characters in the string are lowercase

```
In [112]: "vishal".islower()
Out[112]: True
```

```
In [113]: "VishaL".islower()
Out[113]: False
```

```
In [115]: "vish123".islower()
Out[115]: True
```

String isnumeric() method and isdigit()

The isnumeric() method checks whether a string contains only numeric characters. It is only available on unicode objects.

Syntax: str.isnumeric()

Parameters NA

Return type

This method returns true if all characters are numeric; otherwise, it returns false.

```
In [117]: "123456".isnumeric()
Out[117]: True
```

```
In [118]: "vish123".isnumeric()
Out[118]: False
```

```
In [130]: "123456".isdigit()
Out[130]: True
```

```
In [131]: "vish123".isdigit()
Out[131]: False
```

String isspace() method

The isspace() method checks whether the string consists of whitespace.

Syntax: str.str.isspace()

Parameters NA

Return type

This method returns true if there is at least one character in the string and there are only whitespace characters; otherwise, it returns false.

```
In [119]: " ".isspace()
Out[119]: True
```

```
In [121]: " vish not".isspace()
Out[121]: False
```

String istitle() method

The istitle() method checks whether all case-based characters in a string following non-case-based letters are uppercase and whether all other case-based characters are lowercase.

Syntax: str.istitle()

Parameters NA

Return type

As an example, uppercase characters can only follow uncased characters, and lowercase characters can only follow cased characters. So, this method returns true if the string is a title cased string and there is at least one character; otherwise, it returns false.

```
In [123]: "Vishal Acharya".istitle()
Out[123]: True
```

```
In [124]: "Vishal acharya".istitle()
Out[124]: False
```

String isupper() method

The isupper() method determines whether all the case-based characters (letters) in the string are uppercase.

Syntax: str.isupper()

Parameters NA

Return type The method returns true only if all characters in the string are uppercase

```
In [126]: "VISHAL".isupper()

Out[126]: True

In [127]: "ViSHAL".isupper()

Out[127]: False
```

string isidentifier() method

```
In [133]: "1nb".isidentifier()

Out[133]: False

In [134]: "vish1".isidentifier()

Out[134]: True

In [135]: "True".isidentifier()

Out[135]: True

In [137]: "sum".isidentifier()

Out[137]: True
```

String join() method

The join() method returns a string containing all the string elements of a sequence separated by an str separator.

Syntax: str.join(sequence)

Parameters sequence - In this case, the elements will be joined in a sequence

Return type

The method returns a string that is the concatenation of the strings in the sequence seq. The string providing this method acts as a separator between elements.

```
In [1]: " ".join(['hi', 'my', 'name', 'is', 'vishal'])

Out[1]: 'hi my name is vishal'

In [3]: "vishal".join("acharya")

Out[3]: 'avishalcvishalhvishalavishalrvishalyvishala'

In [4]: str1="vishal"
        str2="acharya"
        str1.join(str2)

Out[4]: 'avishalcvishalhvishalavishalrvishalyvishala'

In [5]: "<".join("123")

Out[5]: '1<2<3'
```

String split() method

The split() function splits a string into words using str as a separator (breaks on whitespace if not specified), there is an option limiting the number of splits to num.

Syntax: str.split(str="", num=string.count(str))

Parameters str - It is any delimiter; by default, this is space This is the number of lines to be made

Return type This method returns a list of lines

In [6]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.split()
```

Out[6]:
```
['A',
 'branch',
 'is',
 'an',
 'instruction',
 'in',
 'a',
 'computer',
 'program',
 'that',
 'can',
 'cause',
 'a',
 'computer',
 'to',
 'begin',
 'executing',
 'a',
 'different',
 'instruction']
```

In [11]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.split("in")
```

Out[11]:
```
['A branch is an ',
 'struction ',
 ' a computer program that can cause a computer to beg',
 ' execut',
 'g a different ',
 'struction']
```

In [10]:
```
s="A branch, is an instruction in a computer, program that can cause, a computer to begin executing, a different instruction"
s.split(",")
```

Out[10]:
```
['A branch',
 ' is an instruction in a computer',
 ' program that can cause',
 ' a computer to begin executing',
 ' a different instruction']
```

Syntax:

str.replace(old, new[, max])

Parameters

old – This is the old substring to be replaced

new - This is a new substring that replaces the old substrings

max - When the optional argument max is given, only the first count occurrences are replaced

Return type This method returns a copy of the string with all instances of substring old replaced by the new one. By specifying the optional argument max, only the first count occurrences are replaced.

In [12]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.replace("in","of")
```

Out[12]:
```
'A branch is an ofstruction of a computer program that can cause a computer to begof executofg a different ofstruction'
```

In [16]:
```
s="A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction"
s.replace("in","of",2)
```

Out[16]:
```
'A branch is an ofstruction of a computer program that can cause a computer to begin executing a different instruction'
```

String lstrip() method

The lstrip() method returns a copy of the string that has been stripped of all characters at the beginning (default whitespace characters).

Syntax: str.lstrip([chars])

Parameters chars - You can specify which characters need to be trimmed.s

In [19]:
```
"abcvishal acharya".lstrip("abc")
```
Out[19]:
```
'vishal acharya'
```

In [39]:
```
"abcvishal acharya".rstrip("a")
```
Out[39]:
```
'abcvishal achary'
```

In [20]:
```
"  abcvishal".lstrip()
```
Out[20]:
```
'abcvishal'
```

In [ ]:
```
String rstrip() method

Whitespace characters are stripped from the string using rstrip()
(default).

Syntax:
str.rstrip([chars])

Parameters
chars - You can give what chars have to be trimmed
```

In [21]:
```
"abcvishal acharya".rstrip("ya")
```
Out[21]:
```
'abcvishal achar'
```

In [24]:
```
"abcvishal acharya  ".rstrip(" ")
```
Out[24]:
```
'abcvishal acharya'
```

In [25]:
```
"abcvishal acharya  ".rstrip("a")
```
Out[25]:
```
'abcvishal acharya  '
```

String strip() method

The Strip() method returns a copy of a string in which all whitespace characters have been stripped (default).

Syntax: str.strip([chars])

Parameters chars - Characters will be trimmed at the start or the end with the selected characters.

In [26]:
```
"abcvishal acharya".strip("a")
```
Out[26]:
```
'bcvishal achary'
```

In [27]:
```
" abcvishal acharya ".strip(" ")
```
Out[27]:
```
'abcvishal acharya'
```

In [28]:
```
" abcvishal acharya ".strip("a")
```
Out[28]:
```
' abcvishal acharya '
```

String maketrans() method

Using each character in the intabstring is mapped into the character in the outtabstring at the same position. Translate() then takes this table as input.

Syntax: str.maketrans(intab, outtab]);

Parameters intab - The actual characters are in this string outtab - There are corresponding mapping characters in this string

Return type The method returns a translate table to be used with the translate() function

In [34]:
```
intab="aeiou"
outtab = "12345"
my = "this is string translate function…!!"
trantab =my.maketrans(intab, outtab)
print (my.translate(trantab))
```
```
th3s 3s str3ng tr1nsl1t2 f5nct34n…!!
```

String translate() method

A copy of the string is returned by the translate() method, in which the characters have been translated using table (created by maketrans() in the string module), optionally removing any characters found in the string

Syntax: str.translate(table[, deletechars]);

Parameters table can use the maketrans() function to create translate table deletechars - Several characters in the source string need to be removed

Return type This method returns a copy of the translated string

In [5]:
```
import string

a_string = '!hi. wh?at is the weat[h]er lik?e.'
new_string = a_string.translate(str.maketrans('','',string.punctuation))


print(new_string)
```

```
hi what is the weather like
```

In [6]:
```
# Find the length of a given string without using the len() function

s = input('enter the string')

counter = 0

for i in s:
  counter += 1

print('length of string is',counter)
```

```
enter the stringvishal
length of string is 6
```

In [7]:
```
# Extract username from a given email.
# Eg if the email is nitish24singh@gmail.com
# then the username should be nitish24singh

s = input('enter the email')

pos = s.index('@')
print(s[0:pos])
```

```
enter the emailvishal76@gmail.com
vishal76
```

In [8]:
```
# Count the frequency of a particular character in a provided string.
# Eg 'hello how are you' is the string, the frequency of h in this string is 2.

s = input('enter the email')
term = input('what would like to search for')

counter = 0
for i in s:
  if i == term:
    counter += 1

print('frequency',counter)
```

```
enter the emailacharyavishal87@gmail.com
what would like to search fora
frequency 5
```

In [9]:
```
# Write a program which can remove a particular character from a string.
s = input('enter the string')
term = input('what would like to remove')

result = ''

for i in s:
  if i != term:
    result = result + i

print(result)
```

```
enter the stringvishalacharya
what would like to removeh
visalacarya
```

In [11]:
```
# Write a program that can check whether a given string is palindrome or not.
# abba
# malayalam

s = input('enter the string')
flag = True
for i in range(0,len(s)//2):
  if s[i] != s[len(s) - i -1]:
    flag = False
    print('Not a Palindrome')
    break

if flag:
  print('Palindrome')
```

```
enter the stringabcba
Palindrome
```

In [12]:
```
# Write a program to count the number of words in a string without split()

s = input('enter the string')
L = []
temp = ''
for i in s:

  if i != ' ':
    temp = temp + i
  else:
    L.append(temp)
    temp = ''

L.append(temp)
print(L)
```

```
enter the stringhow are you
['how', 'are', 'you']
```

In [15]:
```
# Write a program to count the number of words in a string without split()

s = input('enter the string')
L = []
temp = ''
for i in s:

  if i != ' ':
    temp = temp + i
  else:
    L.append(temp)
    temp = ''

L.append(temp)
print(L)
print(len(L))
```

```
enter the stringhow are you
['how', 'are', 'you']
3
```

In [16]:
```
# Write a python program to convert a string to title case without using the title()
s = input('enter the string')

L = []
for i in s.split():
  L.append(i[0].upper() + i[1:].lower())

print(" ".join(L))
```

```
enter the stringhow are you
How Are You
```

In [17]:
```
# Write a program that can convert an integer to string.

number = int(input('enter the number'))

digits = '0123456789'
result = ''
while number != 0:
  result = digits[number % 10] + result
  number = number//10

print(result)
print(type(result))
```

```
enter the number12345
12345
<class 'str'>
```

In [18]:
```
#Write a program to create a string made of first,middle and last character
s=input("enter string: ")
new_string=" "
new_string=s[0]+s[len(s)//2]+s[-1]
print(new_string)
```

```
enter string: vishalacharya
vaa
```

In [19]:
```
#Write a program to find all occuences of a sub string in a given string by ignoring the case.
s=input("enter string: ")
a=input("sub string: ")
print(s.upper().count((a.upper())))
```

```
enter string: lj university
sub string: I
2
```

In [20]:
```
#Write a program to calculate the sum and average of the digits present in a string.
s=(input("enter string: "))
sum=0
for i in s:
    if i.isdigit()==True:
        sum+=int(i)
print(sum)
```

```
enter string: vish123 lki25 457
29
```

In [21]:
```
k=(input("enter string: "))
sum=0
s=k.split(" ")
for i in s:
    if i.isdigit()==True:
        sum+=int(i)
print(sum)
```

```
enter string: vish123 lki25 457
457
```

In [23]:
```
#Write a Python program to print even length words in a string.
s=input("enter string: ")
n=s.split(" ")
for i in n:
    if len(i)%2==0:
        print(i,end=" ")
```

```
enter string: computer fsd python hgy kju lkio lkio
computer python lkio lkio
```

In [24]:
```
#Write a Python program to Uppercase Half String from the given string.
s=input("enter string: ")
new_string1=s[0:len(s)//2:1].upper()
new_string2=s[len(s)//2:len(s)+1:1]
print(new_string1+new_string2)
```

```
enter string: computer fsd python hgy kju lkio lkio
COMPUTER FSD PYTHOn hgy kju lkio lkio
```

In [25]:
```
#Write a Python program to capitalize the first and last character of each word in a string
s=input("enter a string")
s = s.title()
result =  ""
for word in s.split():
    result += word[:-1] + word[-1].upper() + " "
print(result)
```

```
enter a stringlkibngvv ki mnjh
LkibngvV KI MnjH
```

In [26]:
```
#Write a program to Create a string made of the middle three characters
s=input("enter string: ")
new_string=s[(len(s)//2)-1:(len(s)//2)+2]
print(new_string)
```

```
enter string: jhuykjj jhuykjj jhuykjj jhuykjj
j j
```

In [27]:
```
"""Write a program to check if two strings are balanced. For example,
strings s1 and s2 are balanced if all the characters in the s1 are present in s2.
The character’s position doesn’t matter."""
s1=input("enter string: ")
s2= input("enter string: ")
flag=0
for i in s1:
    if i in s2:
        continue
    else:
        flag=1
if(flag ==1):
    print("not balanced")
else:
    print("balanced")
```

```
enter string: lkju
enter string: lk
not balanced
```

In [28]:
```
"""Write a program to check if two strings are balanced. For example,
strings s1 and s2 are balanced if all the characters in the s1 are present in s2.
The character’s position doesn’t matter."""
s1=input("enter string: ")
s2= input("enter string: ")
flag=0
for i in s1:
    if i in s2:
        continue
    else:
        flag=1
if(flag ==1):
    print("not balanced")
else:
    print("balanced")
```

```
enter string: lk
enter string: lkju
balanced
```

In [29]:
```
#Write a program to Split a string on hyphens
s1=input("enter string: ")
s2=s1.split("-")
print(s2)
```

```
enter string: hy]oi-uy-iu
['hy]oi', 'uy', 'iu']
```

In [30]:
```
#Replace each special symbol with # in the following string
import string
s=input("enter string: ")
for i in string.punctuation:
    s=s.replace(i,"#")
print(s)
```

```
enter string: kjk* &^
kjk# ##
```

In [31]:
```
import string

test_str = 'vid, is best: for ! gjhks ;'

test_str = test_str.translate(str.maketrans('', '', string.punctuation))
print(test_str)
```

```
vid is best for  gjhks
```

In [32]:
```
"""Write program that takes user’s name and pan card number
as input validate information using is x function and print the detail¶"""
def pan():
    pan_name=input("enter name")
    if pan_name.isalpha()==False:
        print("sorry, wrong  name")

    pan_number= input("enter pan number: ")
    if pan_number.isalnum()==False:
        print("sorry,wrong number")

    print("correct")

pan()
```

```
enter namejhjj
enter pan number: 1454
correct
```

In [33]:
```
#Wpp program that encrypts a message by adding a key value to every character.
s="abcdefz "
index=0
while index<len(s):
    l=s[index]
    print(chr(ord(l)+3),end="")
    index+=1
```

```
defghi}#
```

In [34]:
```
#Wpp a program to generate abecedarian series
s="ABCDEFGH"
s1="ate"
for i in s:

    print(i+s1)
```

```
Aate
Bate
Cate
Date
Eate
Fate
Gate
Hate
```

In [35]:
```
#Wpp that accept a string from user and redisplay the same string after removing vowels from it
s=input("enter string: ")
new=""
for i in s:
    if i in "aeiouAEIOU":
        continue
    else:
        new=new+i
print(new)
```

```
enter string: ljiet ljuniversity
ljt ljnvrsty
```

In [36]:
```
"""Wpp that find whether a given character is present in a string or not.
In case it is present it prints the index at which it is
present do not use built in find function to search the character"""
s=input("enter string: ")
a=input("enter character:")
index=0
for i in s:
    if a==i:
        print(index,"of s",a)
    index+=1
```

```
enter string: jhujhu
enter character:j
0 of s j
3 of s j
```