Boolean string functions

```
#True or False answer
```

11)startswith()

```
s="I am here to learn python"
s.startswith("t")        #checks the starting alphabet,space counts
```

```
False
```

```
s.startswith("I")
```

```
True
```

```
s.startswith("i")
```

```
False
```

12)endswith()

```
s
```

```
'I am here to learn python'
```

```
s.endswith("n")     #checks the starting alphabet
```

```
True
```

```
s.endswith("N")
```

```
False
```

```
l=["chetana","nikita","sejal"]
for i in l:
    if i.endswith("a"):
        print(i)
```

```
chetana
nikita
```

13)isalpha()

```
#only alphabets counts space

print(s)
```

```
I am here to learn python
```

```
s
```

```
'I am here to learn python'
```

```
# checks the alphabets
#space matters means if space present it gives false answer
s.isalpha()
```

```
False
```

```
d="trupti"   #space not present hence true
d.isalpha()
```

```
True
```

```
s1="learn "
s1.isalpha()
```

```
False
```

14)isalnum()

```
#only alphabets, only digits, combination of both, space matters

s1="chetana"
s2="1234"
s3="chetana1233"
s4="chetana13 "
s5="chetana 234"
print(s1.isalnum())
print(s2.isalnum())
print(s3.isalnum())
print(s4.isalnum())
print(s5.isalnum())
```

```
True
True
True
False
False
```

15)isspace()

```
#only spaces

s6="dsbcfjs "
s7="    "
s8=""
print(s6.isspace())
print(s7.isspace())
print(s8.isspace())
```

```
False
True
False
```

16)isdigit()

```
#only digits, spaces matter

p1="11425"
p2="dh1212"
p3="3434 "
print(p1.isdigit())
print(p2.isdigit())
print(p3.isdigit())
```

```
True
False
False
```

17)islower()

```
#space dosent matter
k1="DATA SCIENCE"
k1.islower()
```

```
False
```

```
k2="data science"
k2.islower()
```

```
True
```

18)isupper()

```
k1.isupper()
```

```
True
```

```
k2.isupper()
```

```
False
```

19)index()

```
k1
```

```
'DATA SCIENCE'
```

```
k1.index("A")  # indexing starts with 0 and space counts
```

```
1
```

```
k1.index("s")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[39], line 1
----> 1 k1.index("s")

ValueError: substring not found
```

```
k1.index("L")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[38], line 1
----> 1 k1.index("L")

ValueError: substring not found
```

```
k1.find("D")
```

```
0
```

```
k1.index("D")
```

```
0
```

```
k1.index("d")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 k1.index("d")

ValueError: substring not found
```

```
k1.find("d")
```

```
-1
```

```
k1.find("P")
```

```
-1
```

```
k1.index("P")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
Cell In[19], line 1
----> 1 k1.index("P")

ValueError: substring not found
```

String slicing

```
#extracting part of the string
#[start:end:step size] by default step size is 1
p="I am Chetana working in ipcs"

#positive indexing and slicing
p[0:12]
```

```
'I am Chetana'
```

```
p[:12]
```

```
'I am Chetana'
```

```
p[:12:1]
```

```
'I am Chetana'
```

```
p[:12:2]
```

```
'Ia htn'
```

```
p[:12:3]
```

```
'Imha'
```

```
p[5:12]
```

```
'Chetana'
```

```
p[5:12:1]
```

```
'Chetana'
```

```
p[5:12:2]
```

```
'Ceaa'
```

```
p[::1]
```

```
'I am Chetana working in ipcs'
```

```
p
```

```
'I am Chetana working in ipcs'
```

```
#negative indexing and slicing
p[-7:]
```

```
'in ipcs'
```

```
q="I am Chetana "
q[-8:-1]
```

```
'Chetana'
```

```
#reverse using positive indexing and slicing :step -1
q[11:4:-1]
```

```
'anatehC'
```

```
q[::-1]
```

```
' anatehC ma I'
```

```
q[::]
```

```
'I am Chetana '
```

```
q
```

```
'I am Chetana '
```

```
#reverse using negative indexing and slicing :step -1
q[-2:-9:-1]
```

```
'anatehC'
```

```
q[::-1]
```

```
' anatehC ma I'
```

```
t="python and data science"
t[0]
```

```
'p'
```

f string methods

```
a=78
b=90
c=a+b
print("addition is this:",c)
```

```
addition is this: 168
```

```
a=78
b=90
```

```
c=a+b
print("addition of",a,"and",b,"is this:",c)
```

```
addition of? 78 and 90 is this: 168
```

```
#formatted string
a=78
b=9
c=a+b
print(f"addition of {a} and  {b} is this:{ c}")
```

```
addition of 78 and  9 is this:87
```

```
var="bark"
print(f"a dog says {var}")
print(f'a dog says {var}')
print(f"""a dog says {var}""")
```

```
a dog says bark
a dog says bark
a dog says bark
```

Escape character

```
text="we are the so called "vikings" from the north"
```

```
  Cell In[23], line 1
    text="we are the so called "vikings" from the north"
                               ^
SyntaxError: invalid syntax
```

```
text="we are the so called , vikings from the north"
text
```

```
'we are the so called , vikings from the north'
```

```
text="we are the so called \"vikings\" from the north"
print(text)
```

```
we are the so called "vikings" from the north
```

```
text="we are the so called vikings \"from\" the north"
print(text)
```

```
we are the so called vikings "from" the north
```

```
text="we are the so called vikings \'from\' the north"
print(text)
```

```
we are the so called vikings 'from' the north
```

```
text="we are the so called vikings \from\? the north"   #f got erase bcz of \(backslash)
print(text)
```

```
we are the so called vikings rom\? the north
```

```
#The r prefix tells Python to treat the string literally,
#meaning that backslashes will not be interpreted as escape characters.
text = r"we are the so called vikings from? the north"   #? is not special character so escape character doest not work on it
print(text)
```

```
we are the so called vikings from? the north
```

```
text="we are the so called \\vikings\\ from the north"
print(text)
```

```
we are the so called \vikings\ from the north
```

\n:new line

```
text="we are the so called \nvikings from the north"
print(text)
```

```
we are the so called
vikings from the north
```

\t:tab

```
text="we are the so called \tvikings from the north"
print(text)
```

```
we are the so called  vikings from the north
```

\b:backspace

```
text="we are the so calle\bd\b  vikings\b from the north"
print(text)
```

```
we are the so call  viking from the north
```

\r:carriage return

```
#carriage return
#carriage return left side=right side,space counts
text="we are the so called  \rvikings from the north"
print(text)
```

```
vikings from the north
```

```
text="we are the so called vikings from \rthe north" #34 = 9
print(text)
```

```
the northe so called vikings from
```

```
text="we are \rthe so called vikings from the north"  #34-9=25
print(text)
```

```
the so called vikings from the north
```

```
s="my name is \rchetana"  #11-7=4
print(s)
```

```
chetana is
```

```
y="I am aa\rhere"
print(y)
```

```
here aa
```

Strip()

remove any leading and trailing whitespace ,characters (such as spaces, tabs, and newlines)

```
text = "  Hello, Welcome!  "
result = text.strip()
print(result)  # Output: "Hello, World!"
```

```
Hello, Welcome!
```

```
text = "@@@Hello, World!@@@"
result = text.strip("@")
print(result)  # Output: "Hello, World!"
```

```
Hello, World!
```

```
text = "@@@Hello, World!@@@"
result = text.lstrip("@")
print(result)  # Output: "Hello, World!"
```

```
Hello, World!@@@
```

```
text = "@@@Hello, World!eee"
result = text.rstrip("e")
print(result)  # Output: "Hello, World!"
```

```
@@@Hello, World!
```

Split()

it splits the string at whitespace characters (spaces, tabs, or newlines)

```
text = "Python and Data Science "
result = text.split()
print(result)
```

```
['Python', 'and', 'Data', 'Science']
```

```
text = "Python,and,Data,Science "
result = text.split(",")
print(result)
```

```
['Python', 'and', 'Data', 'Science ']
```

```
text = "one two three four"
result = text.split(" ",2)
print(result)  # Output: ['one', 'two', 'three four']
#the string is split at the first two spaces
```

```
['one', 'two', 'three four']
```