String Build in Functions

Python String

- The Python string data type is a sequence made up of one or more individual characters that could consist of letters, numbers, whitespace characters, or symbols. Because a string is a sequence, it can be accessed in the same ways that other sequence-based data types are, through indexing and slicing.

Accessing values in String

A=‘Tech Wyvern!.’

Indexing

Print(a[4])

Print(a[-3])

Slicing

Print(a[0:4])

Print(a[-4:-1])

T	E	C	H		W	Y	V	E	R	N	!	.
0	1	2	3	4	5	6	7	8	9	10	11	12

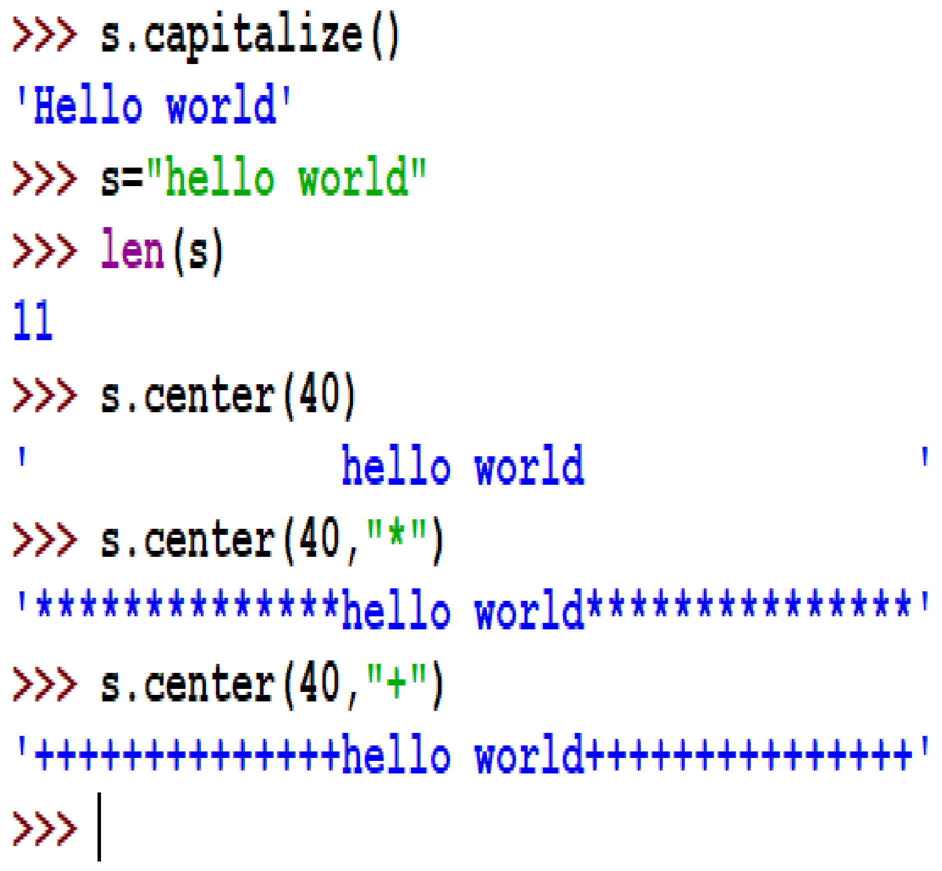
T	E	C	H		W	Y	V	E	R	N	!	.
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Accessing values in String by stride or step

```
>>> a="Tech Wyvern!"
>>> a[0:2:1]
'Te'
>>> #start,stop,step
>>> a[0:5:2]
'Tc '
>>> a[0:5:3]
'Th'
>>>
```

capitalize() and center()

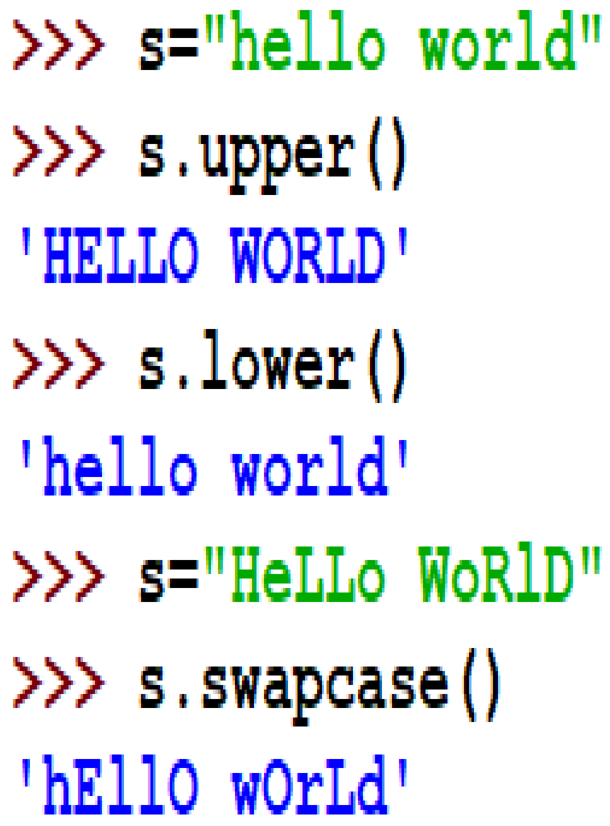
- capitalize() → to make 1st letter in caps
- center() → to make the string in centre and fill the remaining indexes with some other elements



```
>>> s.capitalize()
'Hello world'
>>> s="hello world"
>>> len(s)
11
>>> s.center(40)
'              hello world               '
>>> s.center(40,"*")
'**************hello world***************'
>>> s.center(40,"+")
'++++++++++++++hello world+++++++++++++++'
>>> 
```

upper(),lower() and swapcase()

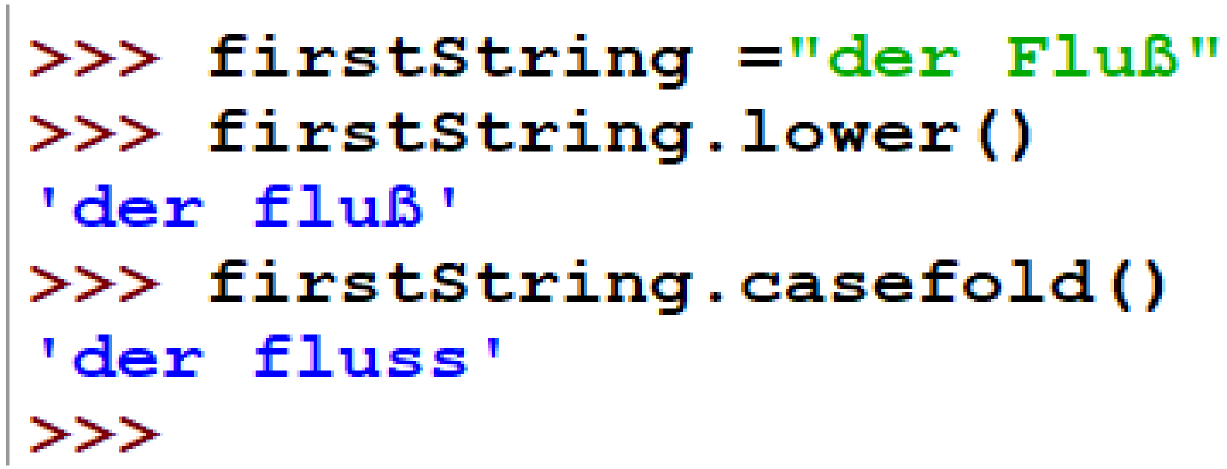
- upper () → used to convert to upper case
- lower () → used to convert to lower case
- swapcase() → used to convert lowercase to uppercase and uppercase to lowercase



```
>>> s="hello world"
>>> s.upper()
'HELLO WORLD'
>>> s.lower()
'hello world'
>>> s="HeLLo WoRlD"
>>> s.swapcase()
'hEllO wOrLd'
```

casefold()

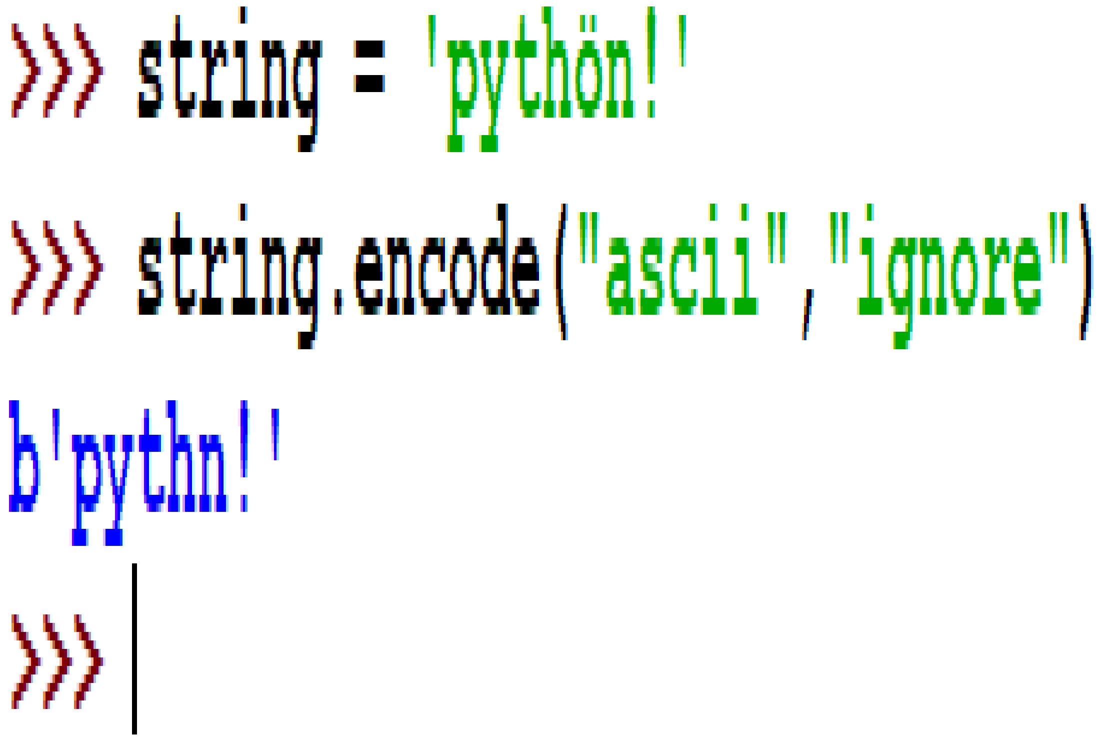
- Like lower case fold also converts to Lower case, but it will help to convert the words which is not comes under normal Alpha ("der Fluß“)



```
>>> firstString ="der Fluß"
>>> firstString.lower()
'der fluß'
>>> firstString.casefold()
'der fluss'
>>>
```

encode()

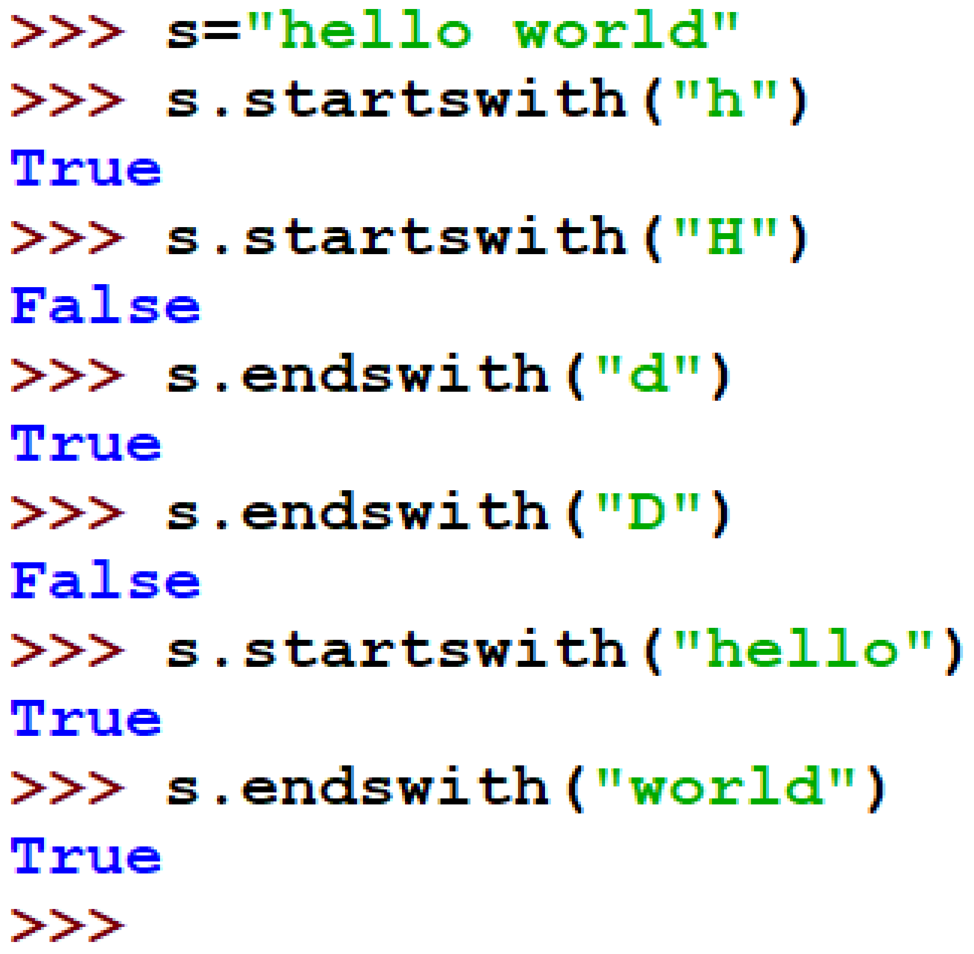
- Which is used to remove the some words which even can’t able to convert by case fold.
- In this Example ‘ö’ is can’t able to convert by case fold but encode can able the remove the word.



```
>>> string = 'pythön!'
>>> string.encode("ascii","ignore")
b'pythn!'
>>> 
```

startswith() and endswith()

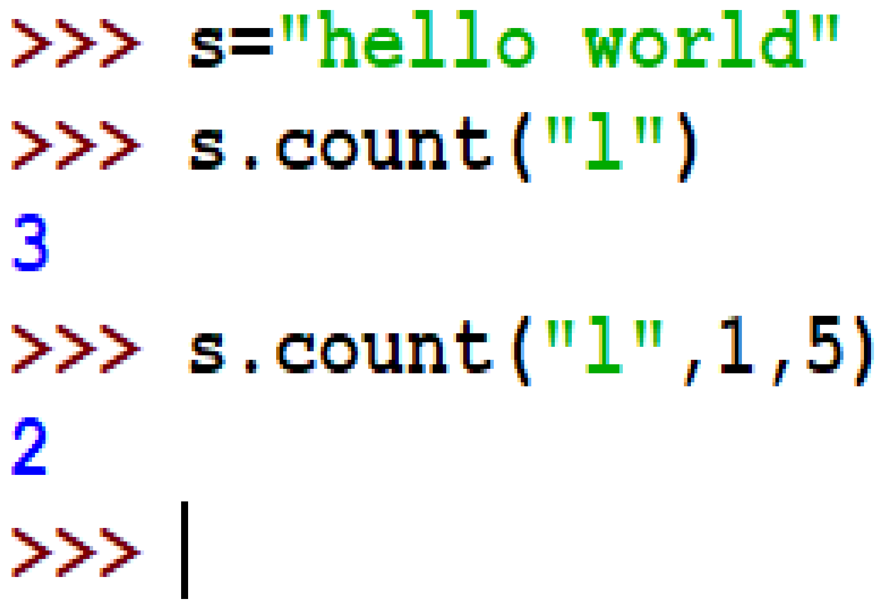
- startswith() → which is used to check either startwith the particular letter / Sequence or not
- endswith() → which is used to check either its endswith the particular letter/sequence



```
>>> s="hello world"
>>> s.startswith("h")
True
>>> s.startswith("H")
False
>>> s.endswith("d")
True
>>> s.endswith("D")
False
>>> s.startswith("hello")
True
>>> s.endswith("world")
True
>>>
```

Count()

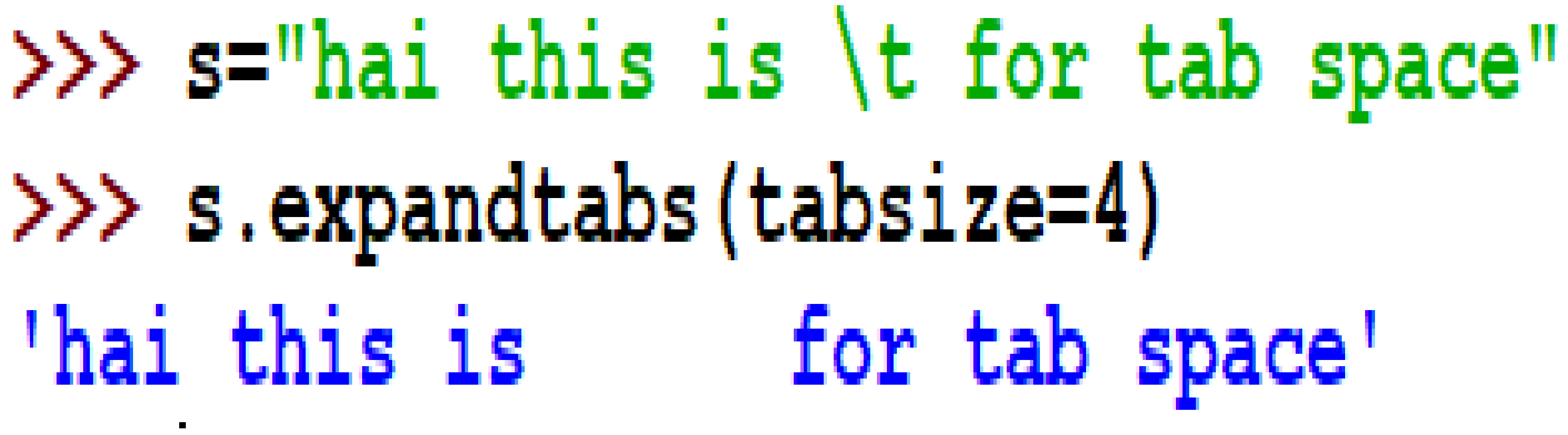
- To check how many times the elements presented in the sequence



```
>>> s="hello world"
>>> s.count("l")
3
>>> s.count("l",1,5)
2
>>> 
```

expandtabs()

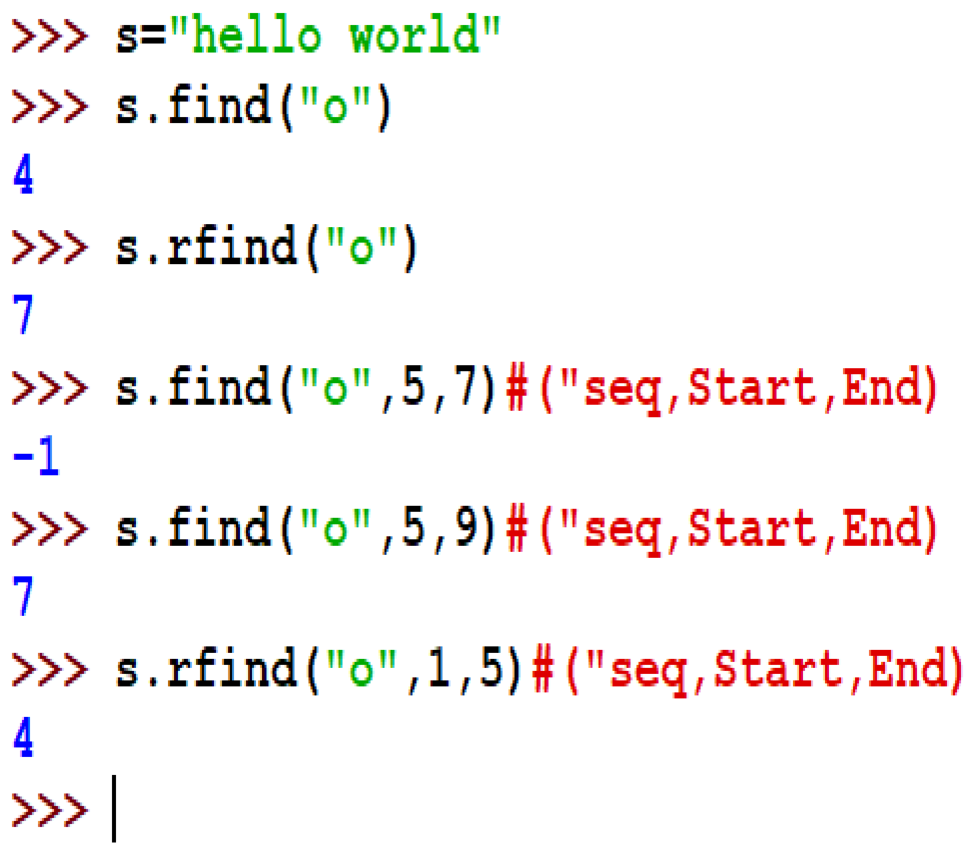
- Used to increase the Tab Spaces ‘\t’ Now its not working



```
>>> s="hai this is \t for tab space"
>>> s.expandtabs(tabsize=4)
'hai this is         for tab space'
```

find() and rfind()

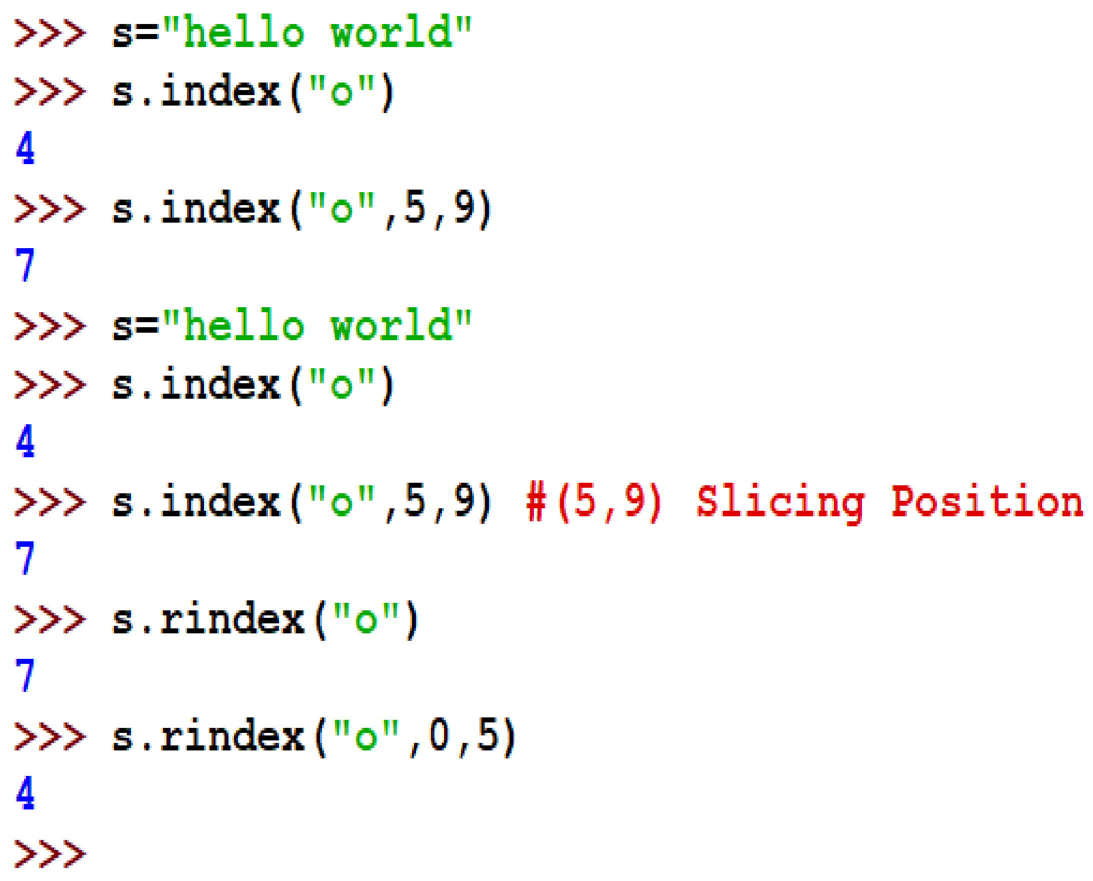
- To find the index position of element if output is -1 then the element is not available in sequence
- find → works left to right
- rfind→ works Right to Left



```
>>> s="hello world"
>>> s.find("o")
4
>>> s.rfind("o")
7
>>> s.find("o",5,7)#("seq,Start,End)
-1
>>> s.find("o",5,9)#("seq,Start,End)
7
>>> s.rfind("o",1,5)#("seq,Start,End)
4
>>> |
```

index() and rindex()

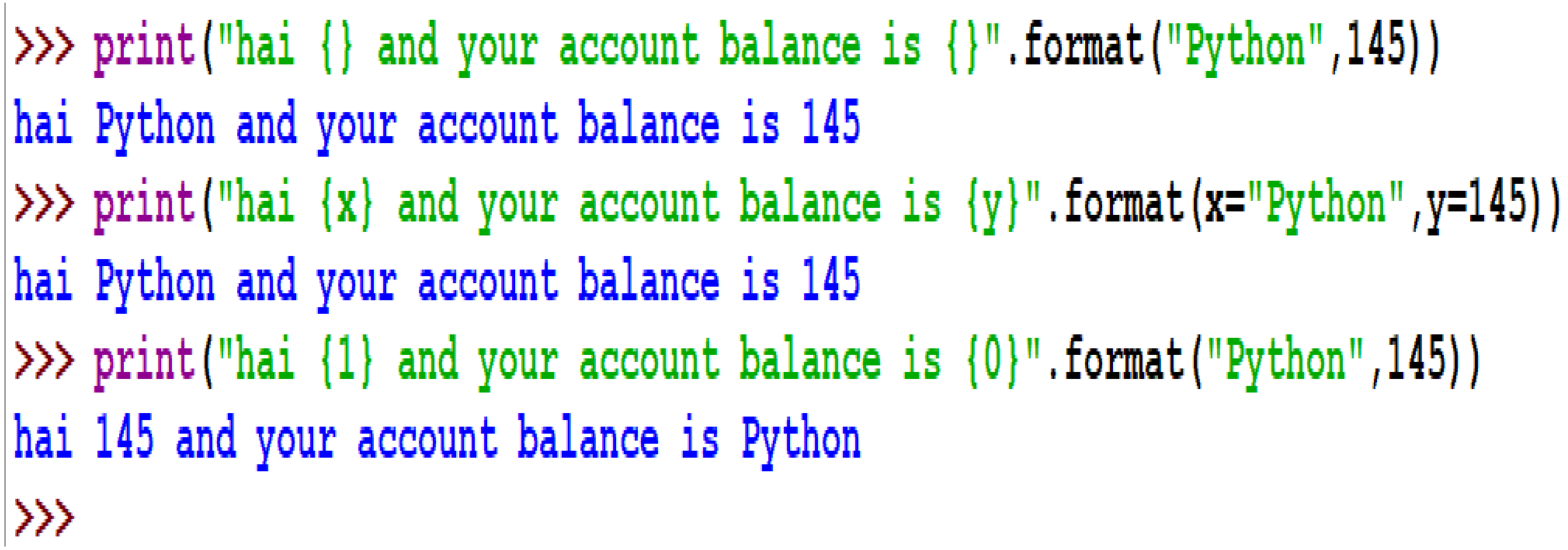
- ***index*** → Which is used to found the index position of one letter or sequence starts from left
- ***rindex*** →Which is used to found the index position of one letter or sequence starts from right



```
>>> s="hello world"
>>> s.index("o")
4
>>> s.index("o",5,9)
7
>>> s="hello world"
>>> s.index("o")
4
>>> s.index("o",5,9) #(5,9) Slicing Position
7
>>> s.rindex("o")
7
>>> s.rindex("o",0,5)
4
>>>
```

Format

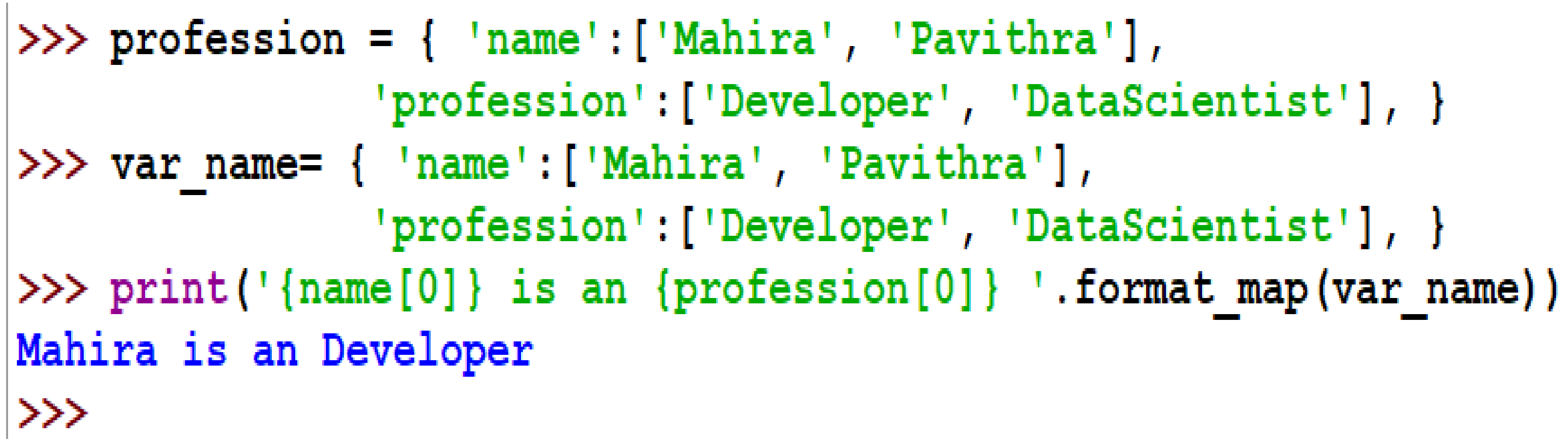
- Which is used to make the default template and with different values



```
>>> print("hai {} and your account balance is {}".format("Python",145))
hai Python and your account balance is 145
>>> print("hai {x} and your account balance is {y}".format(x="Python",y=145))
hai Python and your account balance is 145
>>> print("hai {1} and your account balance is {0}".format("Python",145))
hai 145 and your account balance is Python
>>>
```

formatmap()

- Used to deal with dictionaries, difference is, the str.format(**mapping) copies the dict whereas str.format_map(mapping) makes a new dictionary during method call. This can be useful if you are working with a dict subclass.



```
>>> profession = { 'name':['Mahira', 'Pavithra'],
               'profession':['Developer', 'DataScientist'], }
>>> var_name= { 'name':['Mahira', 'Pavithra'],
               'profession':['Developer', 'DataScientist'], }
>>> print('{name[0]} is an {profession[0]} '.format_map(var_name))
Mahira is an Developer
>>>
```

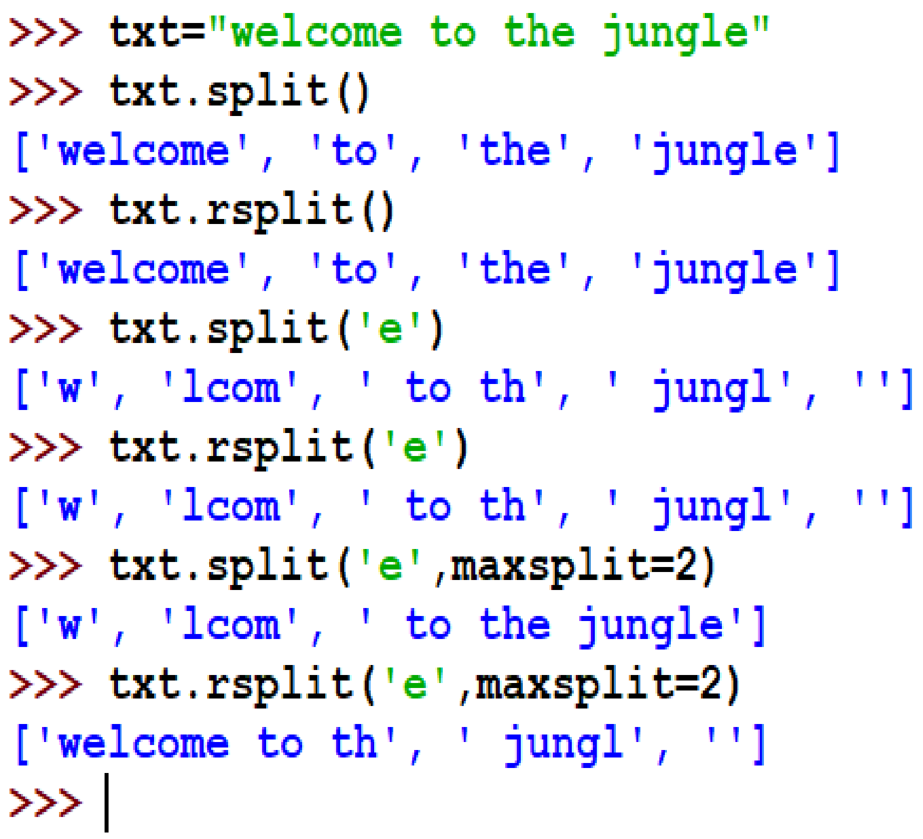
Split() and Splitlines()

- ***string.split(separator, max)***

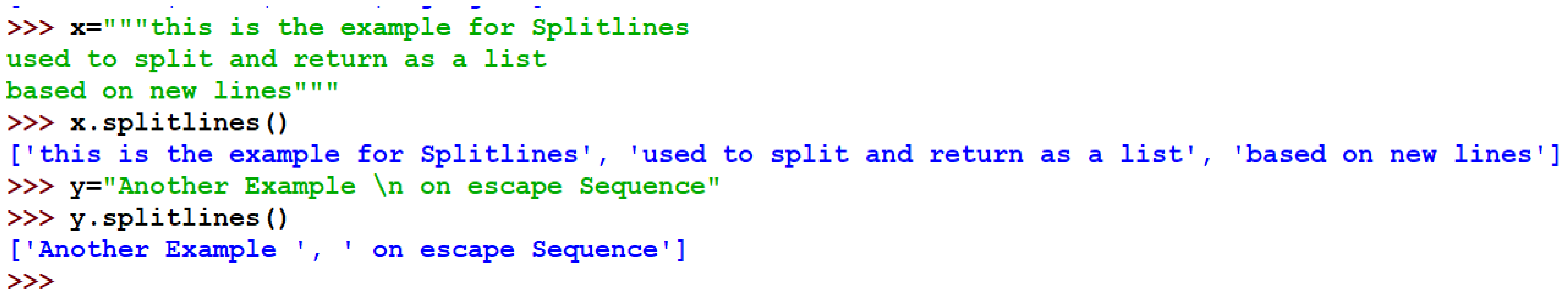
Splits the string at the specified separator, and returns a list

- ***string.splitlines(keeplinebreaks)***

The splitlines() method splits a string into a list. The splitting is done at line breaks



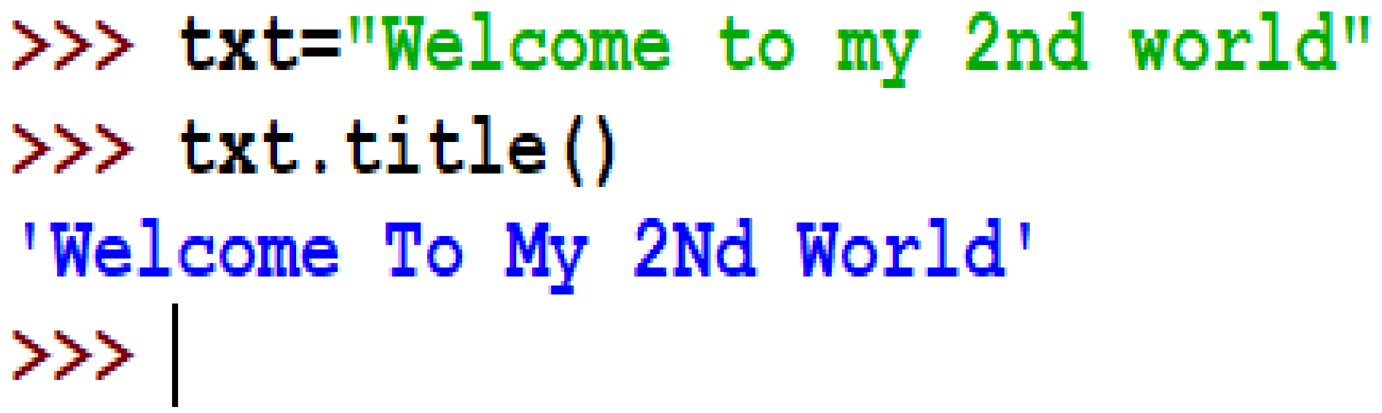
```
>>> txt="welcome to the jungle"
>>> txt.split()
['welcome', 'to', 'the', 'jungle']
>>> txt.rsplit()
['welcome', 'to', 'the', 'jungle']
>>> txt.split('e')
['w', 'lcom', ' to th', ' jungl', '']
>>> txt.rsplit('e')
['w', 'lcom', ' to th', ' jungl', '']
>>> txt.split('e',maxsplit=2)
['w', 'lcom', ' to the jungle']
>>> txt.rsplit('e',maxsplit=2)
['welcome to th', ' jungl', '']
>>> 
```



```
>>> x="""this is the example for Splitlines
used to split and return as a list
based on new lines"""
>>> x.splitlines()
['this is the example for Splitlines', 'used to split and return as a list', 'based on new lines']
>>> y="Another Example \n on escape Sequence"
>>> y.splitlines()
['Another Example ', ' on escape Sequence']
>>> 
```

title()

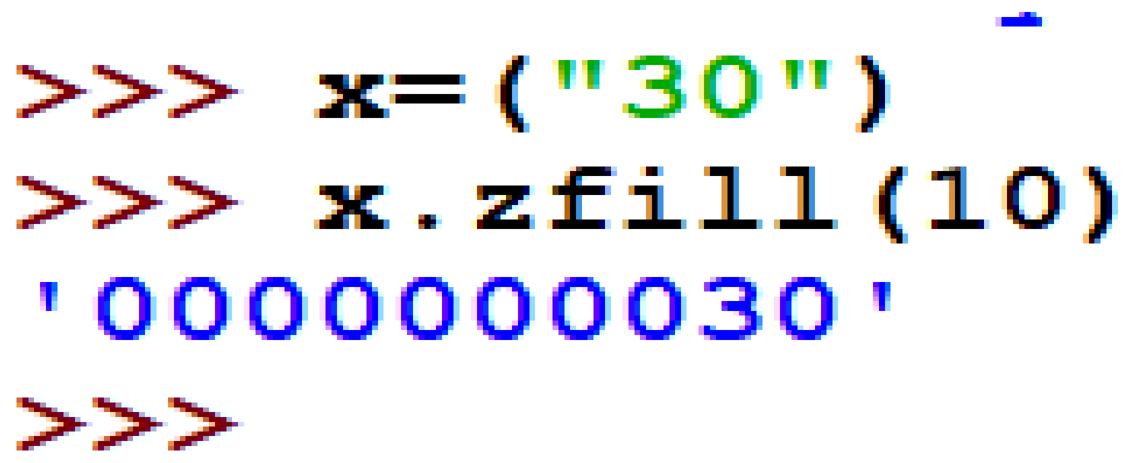
string.title() → The title() method returns a string where the first character in every word is upper case.



```
>>> txt="Welcome to my 2nd world"
>>> txt.title()
'Welcome To My 2Nd World'
>>> 
```

zfill()

***string*.zfill(*len*)** → zfill() method adds zeros (0) at the beginning of the string, until it reaches the specified length.

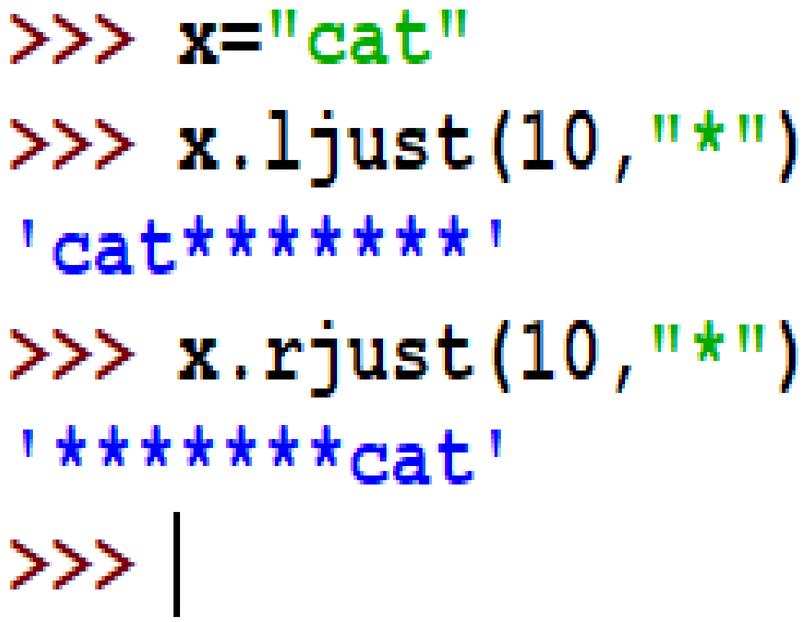


```
>>> x=("30")
>>> x.zfill(10)
'0000000030'
>>>
```

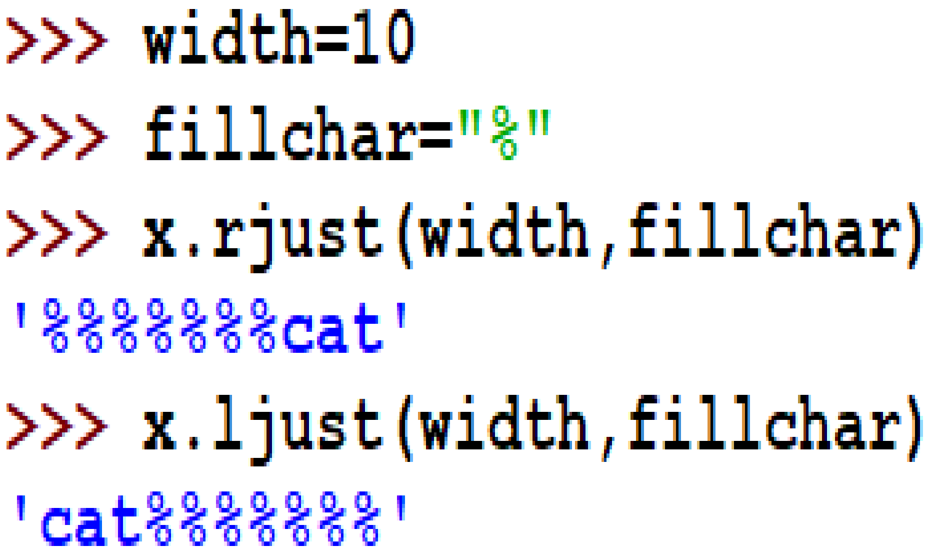
ljust() and rjust()

string.ljust(width[, fillchar])→ string ljust() method returns a left-justified string of a given minimum width.

string.rjust(width[, fillchar]) → string rjust() method returns a right-justified string of a given minimum width



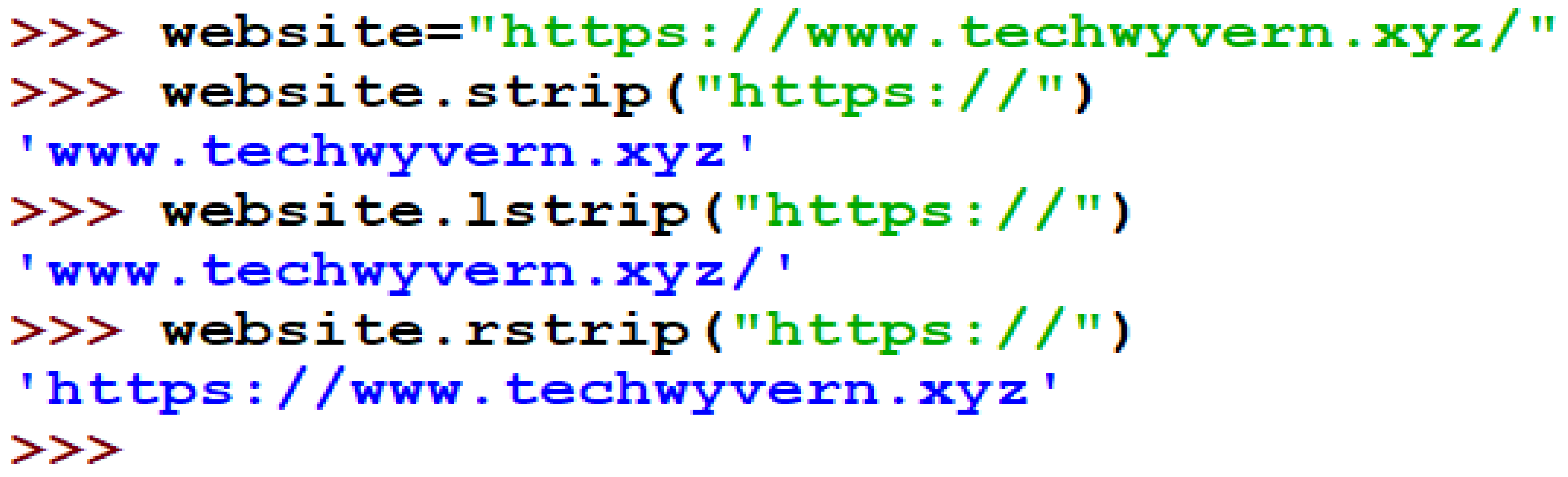
```
>>> x="cat"
>>> x.ljust(10,"*")
'cat*******'
>>> x.rjust(10,"*")
'*******cat'
>>>
```



```
>>> width=10
>>> fillchar="%"
>>> x.rjust(width,fillchar)
'%%%%%%%cat'
>>> x.ljust(width,fillchar)
'cat%%%%%%%'
```

Strip,lstrip and rstrip

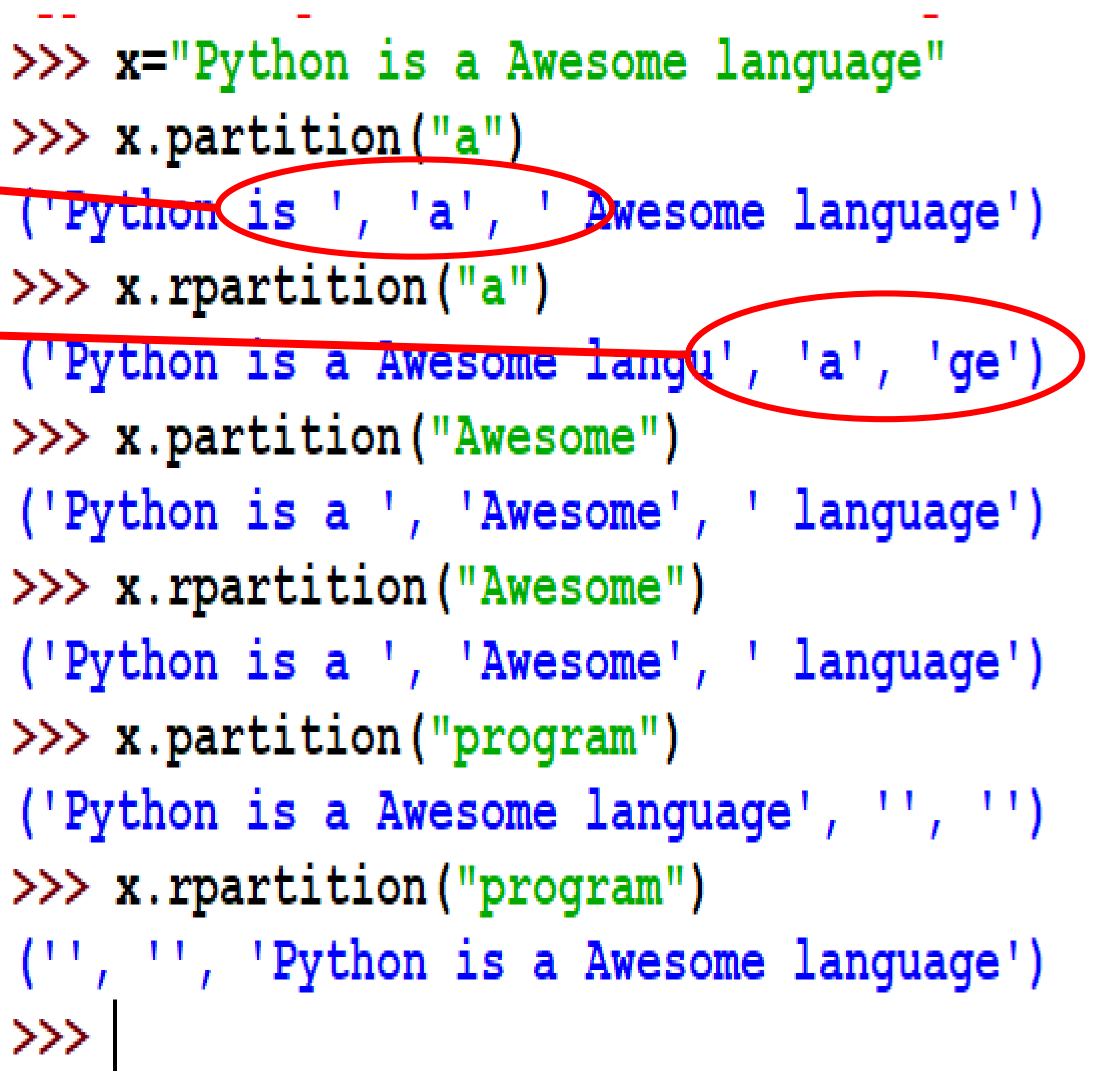
- strip→ Leading Characters Removed
- lstrip → Left Leading Characters Removed
- rstrip → Right Leading Characters Removed



```
>>> website="https://www.techwyvern.xyz/"
>>> website.strip("https://")
'www.techwyvern.xyz'
>>> website.lstrip("https://")
'www.techwyvern.xyz/'
>>> website.rstrip("https://")
'https://www.techwyvern.xyz'
>>>
```

partition() and rpartition()

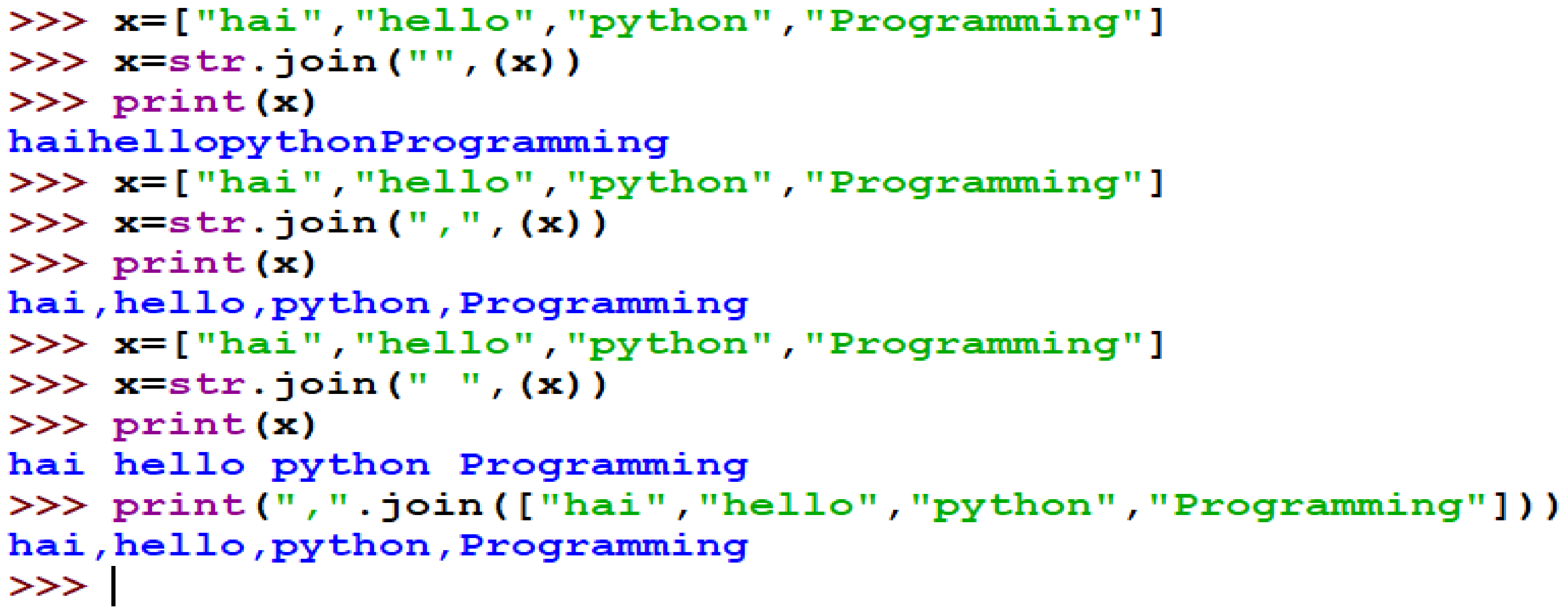
- partition to split string to tuple with three strings
- rpartition work based on right



```
>>> x="Python is a Awesome language"
>>> x.partition("a")
('Python is ', 'a', ' Awesome language')
>>> x.rpartition("a")
('Python is a Awesome langu', 'a', 'ge')
>>> x.partition("Awesome")
('Python is a ', 'Awesome', ' language')
>>> x.rpartition("Awesome")
('Python is a ', 'Awesome', ' language')
>>> x.partition("program")
('Python is a Awesome language', '', '')
>>> x.rpartition("program")
('', '', 'Python is a Awesome language')
>>>
```

join()

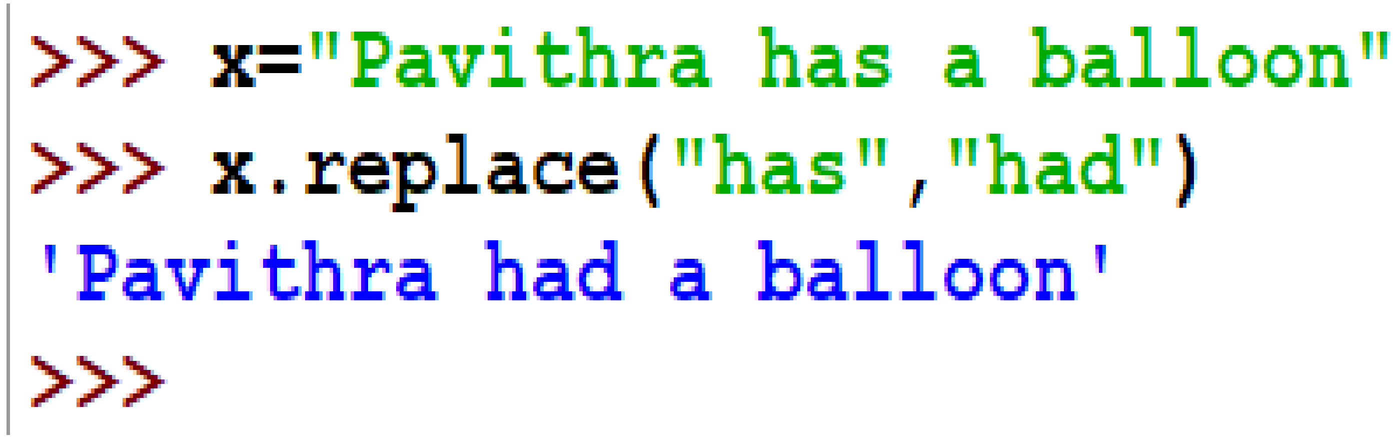
Join used to convert string to List



```
>>> x=["hai","hello","python","Programming"]
>>> x=str.join("",(x))
>>> print(x)
haihellopythonProgramming
>>> x=["hai","hello","python","Programming"]
>>> x=str.join(",",(x))
>>> print(x)
hai,hello,python,Programming
>>> x=["hai","hello","python","Programming"]
>>> x=str.join(" ",(x))
>>> print(x)
hai hello python Programming
>>> print(",".join(["hai","hello","python","Programming"]))
hai,hello,python,Programming
>>> 
```

replace()

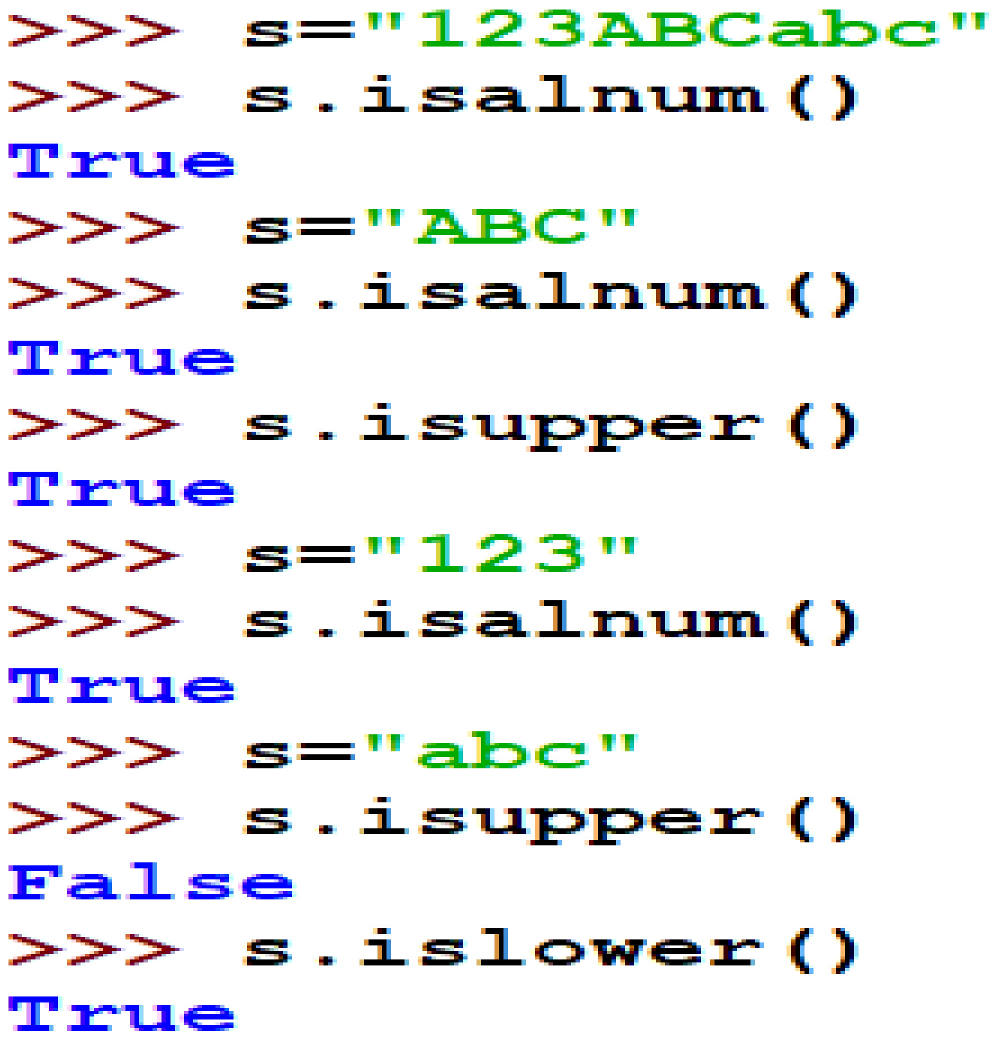
- Used to replace one string with another one



```
>>> x="Pavithra has a balloon"
>>> x.replace("has","had")
'Pavithra had a balloon'
>>>
```

isalnum(), is upper(), and islower()

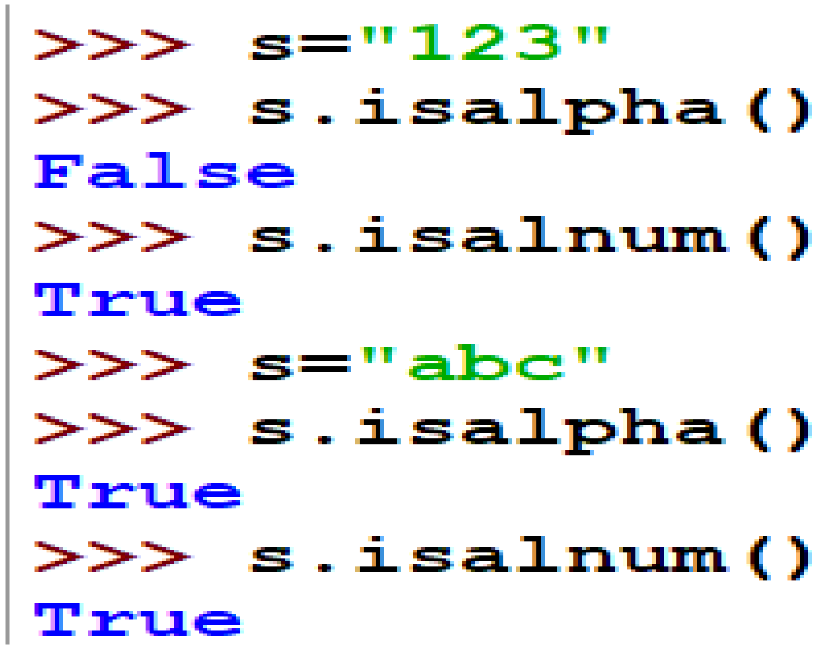
- AlphaNumeric:A character that is either a letter or a number.
- isupper()→ if all the character are in upper case
- islower() → all the character are in lower case



```
>>> s="123ABCabc"
>>> s.isalnum()
True
>>> s="ABC"
>>> s.isalnum()
True
>>> s.isupper()
True
>>> s="123"
>>> s.isalnum()
True
>>> s="abc"
>>> s.isupper()
False
>>> s.islower()
True
```

Isalpha()

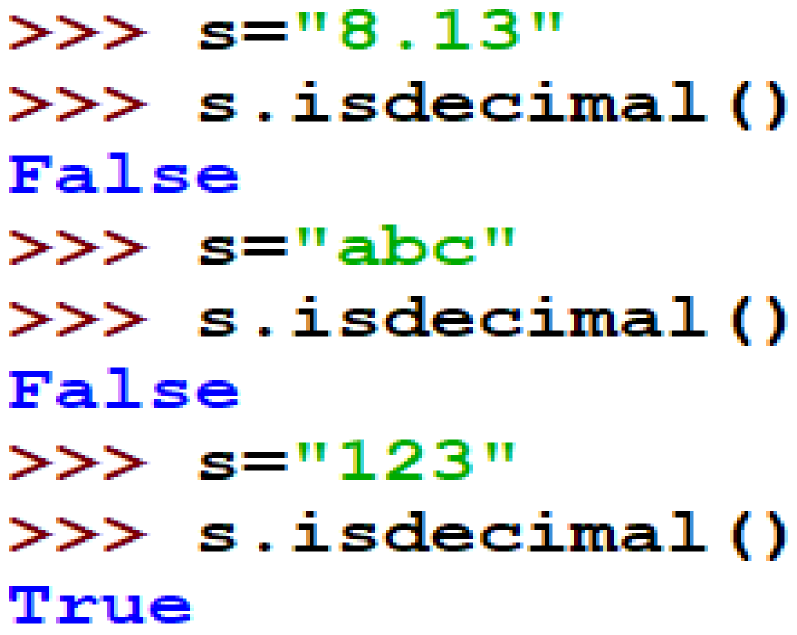
isalpha()True if all characters in the string are alphabets (can be both lowercase and uppercase). False if at least one character is not alphabet.



```
>>> s="123"
>>> s.isalpha()
False
>>> s.isalnum()
True
>>> s="abc"
>>> s.isalpha()
True
>>> s.isalnum()
True
```

isdecimal()

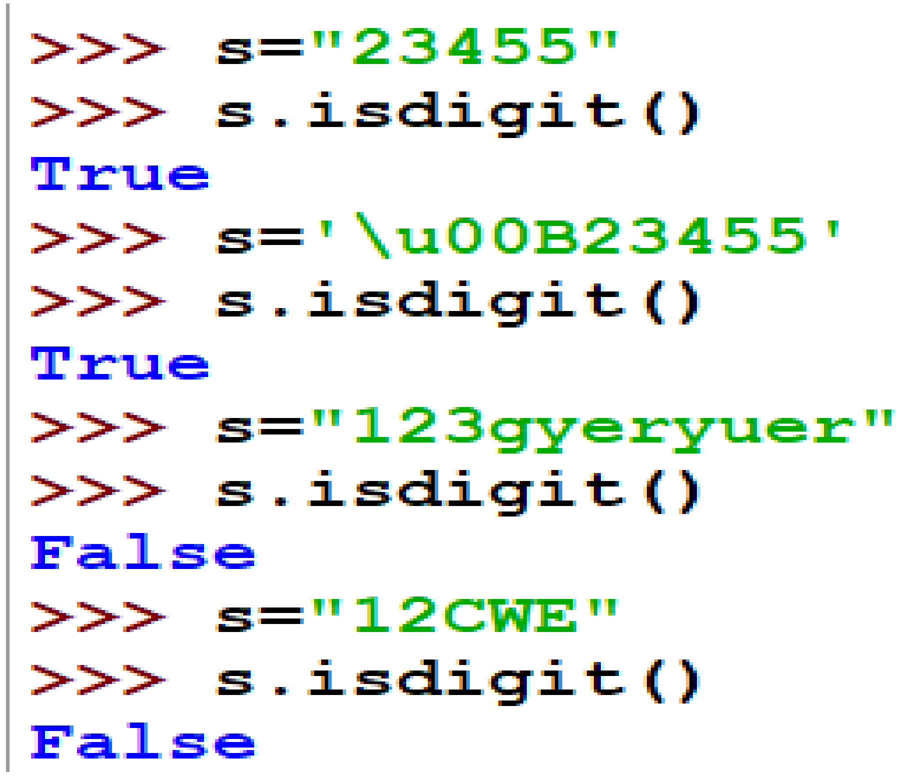
string.isdecimal() **True** if all characters in the string are decimal characters, **False** if at least one character is not decimal character.



```
>>> s="8.13"
>>> s.isdecimal()
False
>>> s="abc"
>>> s.isdecimal()
False
>>> s="123"
>>> s.isdecimal()
True
```

is.digit()

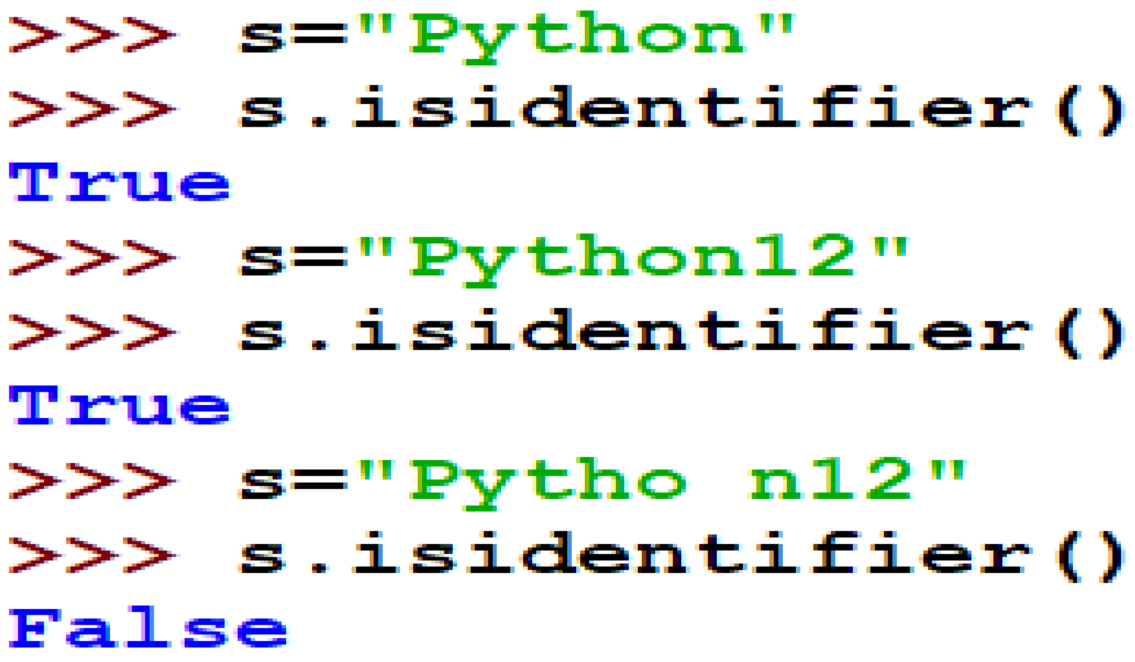
string.isdigit()→returns True if all characters in a string are digits. If not, it returns False.



```
>>> s="23455"
>>> s.isdigit()
True
>>> s='\u00B23455'
>>> s.isdigit()
True
>>> s="123gyeryuer"
>>> s.isdigit()
False
>>> s="12CWE"
>>> s.isdigit()
False
```

isidentifier()

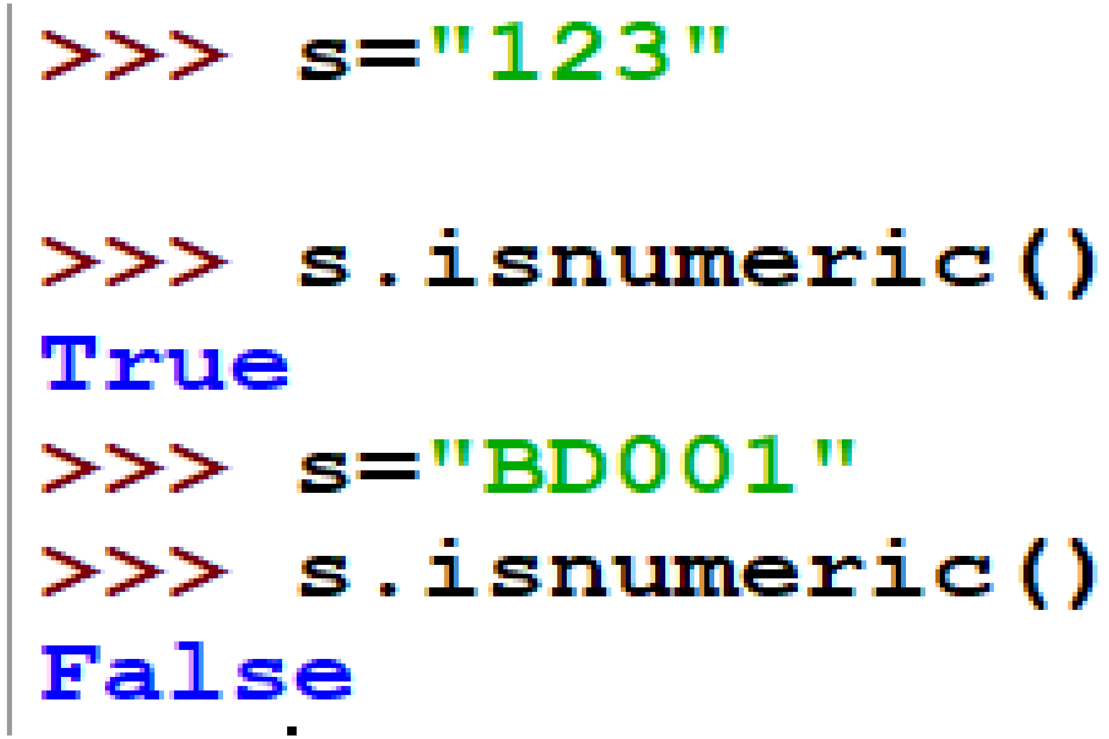
isidentifier() → True if the string is a valid identifier in Python. If not, it returns False.



```
>>> s="Python"
>>> s.isidentifier()
True
>>> s="Python12"
>>> s.isidentifier()
True
>>> s="Pytho n12"
>>> s.isidentifier()
False
```

isnumeric()

isnumeric() →True if all characters in a string are numeric characters.

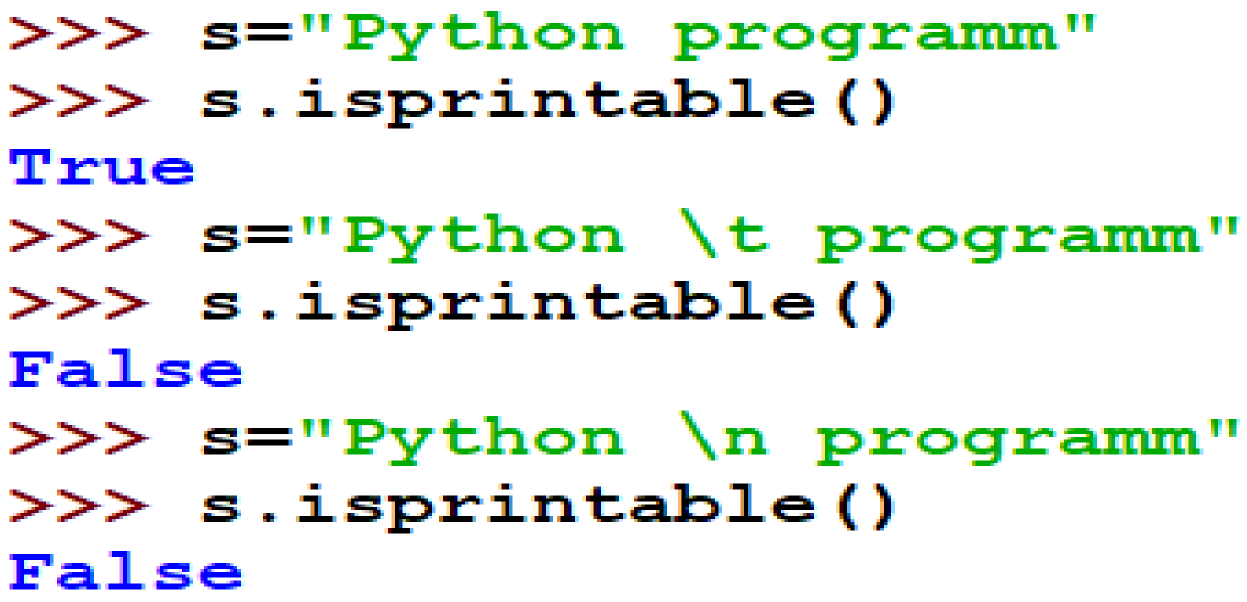


```
>>> s="123"

>>> s.isnumeric()
True
>>> s="BD001"
>>> s.isnumeric()
False
```

isprintable()

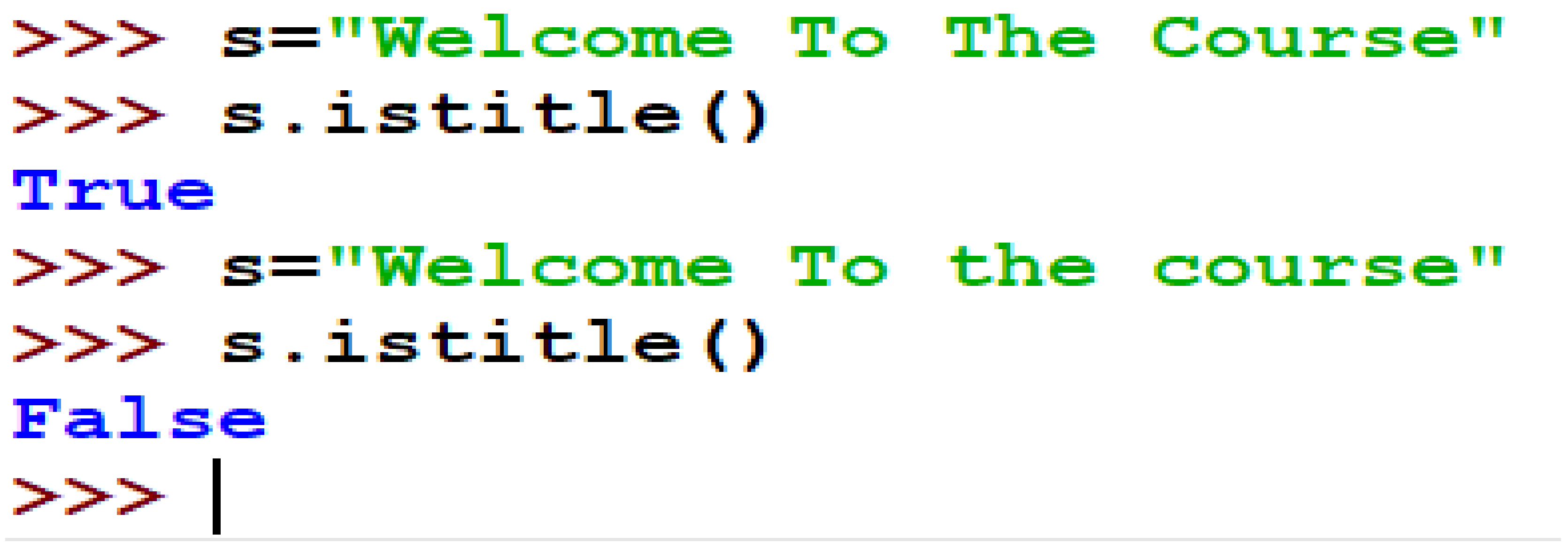
isprintable() → True if all characters in the string are printable or the string is empty. If not, it returns False.



```
>>> s="Python programm"
>>> s.isprintable()
True
>>> s="Python \t programm"
>>> s.isprintable()
False
>>> s="Python \n programm"
>>> s.isprintable()
False
```

istitle()

istitle() →if the string is a titlecased string. If not, it returns False



```
>>> s="Welcome To The Course"
>>> s.istitle()
True
>>> s="Welcome To the course"
>>> s.istitle()
False
>>> |
```

String Operators

Concatenation operator ‘+’
Repetition of string using * operator
Repetition of string using * operator
Range of slice
String operator ‘in’
String operator ‘not in’

Concatenation operator ‘+’

```
var1 = 'Welcome '
var2 = 'John'
var3 = var1 + var2
print ("'welcome ' + 'John' = " + var3)
```

Repetition of string using * operator

```
var1 = 'Welcome '
var2 = var1*6
print ("'welcome ' * 6 = " + var2)
```

Retrieving character from a given index of string

```
var1 = 'Welcome '
print ("The second character in string 'welcome ' is " + var1[1])
```

Range of slice

```
var1 = 'Welcome ‘
print ("The second to fourth characters in string 'welcome ' are " + var1[1:5])
```

String operator ‘in’

```
var1 = 'Welcome’
if 'e' in var1:
        print ("'e' exists in the word 'welcome’”)
 else:
        print ("'e' does not exist in the word 'welcome'")
```

Not in

```
var1 = 'Welcome'
if 'e' not in var1:
          print ("'e' does not exist in the word 'welcome'")
 else:
          print ("'e' exists in the word 'welcome'")
```