Comparison Operator

Python standard comparison operators <, <=, >, >=, ==, !=, <> apply to strings also. The comparison using these operators are based on standard character by character comparison rule for ASCII or Unicode ie dictionary order.

```
"abc" == "abc"          True
"a"   <>   "abc"        True
"ABC" == "abc"          False
```

Equality and on equality in string are easier to determine because it goes for exact character matching for individual letter including the case of letter.
But for less than(<) or greater than we should know the ASCII value or unicode values.

Character	ASCII or Unicode
0-9	48 - 57
A - Z	65 - 90
a - z	97 - 122

therefore the uppercase letters are considered smaller than lower case letters.

`'a' < 'A'` False because ASCII value of lowercase letters as higher than upper case letters.

`'ABC' > 'AB` True

Determine ASCII or Unicode value of a single character

Python offers built in function ord() that takes a single character and return the corresponding ASCII value of unicode value.

```
ord(single char)
ord('A')
```

To know unicode value of letter 'A' you will write

`ord(u'A')` and Python return the corresponding unicode value.

The opposite of ord() function is char() ie the char() function takes the ASCII value in integer form and return the character corresponding to that ASCII value.

String Functions and methods

Every string object that you create in Python is actually an instance of string class (built in class). The string manipulation method that can be applied to string are as follows:

<string-object>.<method-name>()

i) string.capitalize()

Return a copy of the string with first character capitalized

```
'true'.capitalize()
'True'
```

ii) strng.find(sub, [start, end])

This method return the lowest index in the string where the substring sub is found within the slice range of start and end.

Return -1 if sub is not found.

```
>>> string = 'it goes as - ringa ringa roses'
>>> sub = "ring"
>>> string.find(sub)
13
>>> string.find(sub, 15, 22)
-1
>>> string.find(sub, 15, 25)
19
```

iii) string.isalnum()
Return True if the character in the string are alphanumeric (alphabets or numbers) and there is at least on character. False other wise.

```
string  = 'abc123'
str2 = 'hello'
str3 = '12345'
>>> str4 = ''
>>> string.isalnum()
True
```

```
>>> str2.isalnum()
True
>>> str3.isalnum()
True
>>> str4.isalnum()
False
```

iv) string.isalpha()

Return True if all the characters in the string are alphabetic and there is at least one character, False otherwise.

```
string  = 'abc123'
str2 = 'hello'
str3 = '12345'
>>> str4 = ''
>>> string.isalpha()
False
>>> str2.isalpha()
True
>>> str3.isalpha()
False
>>> str4.isalpha()
False
```

v) isdigit()

Return True if all the characters in string are digits. There must be at least one character, otherwise it return False.

```
string  = 'abc123'
str2 = 'hello'
str3 = '12345'
>>> str4 = ''
>>>string.isdigit()
False
>>>str2.isdigit()
False
>>>str3.isdigit()
True
>>>str4.isdigit()
False
```

vi) islower()

Return True if all cased characters in the string are lowercase. There must be at least one cased character. False otherwise

```
str1 = 'hello'
str2 = 'THERE'
str3 = 'Goldy'
>>>str1.islower()
True
>>>str2.islower()
False
>>>str3.islower()
False
```

vii) isspace()

Return True if there are only whitespace characters in the string. There must be al least one character.

```
>>>str = "   "
>>>str2 = ""
>>>str.isspace()
True
>>>str2.isspace()
False
```

viii) string.isupper()

Test whether all cased characters in the string are uppercase and require tat there be at least one cased character. Return True if so and False otherwise.

```
>>>str1 = 'HELLO'
>>>str2 = 'There'
>>>str3 = 'goldy'
>>>str4 = 'U123'
>>> str5 = '123f'
>>> str1.isupper()
True
>>>str2.isupper()
False
>>>str3.isupper()
False
>>>str4.isupper()
True
>>>str5.isupper()
False
```

ix) string.lower()

Return a copy of string converted to lowercase.

```
>>> str1 = 'HELLO'
>>> str2 = 'There'
>>> str3 = 'goldy'
>>> str4 = 'U123'
>>> str1.lower()
'hello'
>>> str2.lower()
'there'
>>> str3.lower()
'goldy'
>>> str4.lower()
'u123'
```

x) string.upper()

Return a copy of string converted to uppercase.

```
>>> str1 = 'HELLO'
>>> str2 = 'There'
>>> str3 = 'goldy'
>>> str1.upper()
'HELLO'
>>> str2.upper()
'THERE'
>>> str3.upper()
'GOLDY'
```

xi) string.lstrip([chars])

Return a copy of string with leading characters removed. If used without any argument it removes the leading whitespaces.

One can use the optional chars argument to specify a set of characters to be removed.

xii) string.rstrip([chars])

Return a copy of string with trailing characters removed. If used without any argument it remove leading whitespaces.

```
>>> string2 = " There"
>>> string2.lstrip()   # No argument supplied to lstrip(),
hence it removed leading                # whitespaces of the
string.
'There'
>>> string2.lstrip('the')
```

```
'There'
```

''' All possible substrings of given argument 'the' are matched with left of the string2 if found removed. That is 'the', 'th', 'he', 't', 'h', 'e' and there reverse string are matched. If any of these is found remove from left of the string.'''

xiii) s.split('delim') -- returns a list of substrings separated by the given delimiter. The delimiter is not a regular expression, it's just text.

```
>>> 'aaa,bbb,ccc'.split(',')
['aaa', 'bbb', 'ccc']
```

As a convenient special case s.split() (with no arguments) splits on all whitespace chars.

xiv) s.join(list) -- opposite of split(), joins the elements in the given list together using the string as the delimiter. `>>> '---'.join(['aaa', 'bbb', 'ccc'])`
`Output -> aaa---bbb---ccc`