String Operations ... ¶

In [1]:
```
myString = 'fall 7 times stand up 8!'
```

In [2]:
```
len(myString)
```

Out[2]:
```
24
```

In [3]:
```
myString.center(len(myString)+1)
```

Out[3]:
```
' fall 7 times stand up 8!'
```

In [4]:
```
myString.center(len(myString)+3)
```

Out[4]:
```
'  fall 7 times stand up 8! '
```

In [8]:
```
myParagraph = "fall 7 times stand up 8!\nfall 7 times stand up 8!\nfall 7 time
s stand up 8!"
```

In [9]:
```
print myParagraph
```

```
fall 7 times stand up 8!
fall 7 times stand up 8!
fall 7 times stand up 8!
```

In [14]:
```
myParagraph.splitlines()
```

Out[14]:
```
['fall 7 times stand up 8!',
 'fall 7 times stand up 8!',
 'fall 7 times stand up 8!']
```

In [16]:
```
myString
```

Out[16]:
```
'fall 7 times stand up 8!'
```

In [17]:
```
myString.encode('base64')
```

Out[17]:
```
'ZmFsbCA3IHRpbWVzIHN0YW5kIHVwIDgh\n'
```

In [18]:
```
myString.encode('base64', 'strict')  # 'strict' option results in raising  Uni
codeError, for encoding errors
```

Out[18]:
```
'ZmFsbCA3IHRpbWVzIHN0YW5kIHVwIDgh\n'
```

In [19]:
```
enString = myString.encode('base64', 'strict')
```

In [20]:
```
enString.decode('base64')
```

Out[20]:
```
'fall 7 times stand up 8!'
```

In [21]:
```
myString.encode('big5', 'strict')  # big5 is for chinese encoding
```

Out[21]:
```
'fall 7 times stand up 8!'
```

In [22]:
```
myString.encode('cp866', 'strict')  # cp866 is for Russian
```

Out[22]:
```
'fall 7 times stand up 8!'
```

In [23]:
```
myString.encode('utf_8', 'strict')
```

Out[23]:
```
'fall 7 times stand up 8!'
```

In [24]:
```
unicode(myString)
```

Out[24]:
```
u'fall 7 times stand up 8!'
```

In [25]:
```
myString.center(len(myString)+1)
```

Out[25]:
```
' fall 7 times stand up 8!'
```

In [26]:
```
myString.center(len(myString)+2)
```

Out[26]:
```
' fall 7 times stand up 8! '
```

In [27]:
```
myString.center(len(myString)+3)  # first ljust() takes place, followed by rjust()
```

Out[27]:
```
'  fall 7 times stand up 8! '
```

In [28]:
```
myString.ljust(len(myString)+3)
```

Out[28]:
```
'fall 7 times stand up 8!   '
```

In [29]:
```
myString.rjust(len(myString)+3)
```

Out[29]:
```
'   fall 7 times stand up 8!'
```

In [30]:
```
"Python Programming".index('P')
```

Out[30]:
```
0
```

In [31]:
```
"Python Programming".rindex('P')
```

Out[31]:
```
7
```

In [32]:
```
"Python Programming".index('y')
```

Out[32]:
```
1
```

In [33]:
```
"Python Programming".rindex('y')   # as there is one occurrance of 'y'
```

Out[33]:
```
1
```

In [34]:
```
myString.capitalize().center(len(myString)+3)
```

Out[34]:
```
'  Fall 7 times stand up 8! '
```

Interview Question 1: what is the difference between string.index() and string.find()?

str.find() returns -1 when it does not find the substring; whereas str.index() wil through an exception

In [35]:
```
myString
```

Out[35]:
```
'fall 7 times stand up 8!'
```

In [36]:
```
myString.find('?')
```

Out[36]:
```
-1
```

In [37]:
```
myString.index('?')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-37-6a55b27524f4> in <module>()
----> 1 myString.index('?')

ValueError: substring not found
```

In [38]:
```
"abc".find("b")
```

Out[38]:
```
1
```

In [39]:
```
"abc".index(0)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-0cae3d0d0c23> in <module>()
----> 1 "abc".index(0)

TypeError: expected a string or other character buffer object
```

In [40]:
```
[1,2,3].index(2)
```

Out[40]:
```
1
```

In [41]:
```
"abc".find("d")
```

Out[41]:
```
-1
```

In [42]:
```
[1,2,3].index(4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-42-9ac0ec7d802f> in <module>()
----> 1 [1,2,3].index(4)

ValueError: 4 is not in list
```

Interview Question 2: What is the difference between string.find() and in (membership test) operator?

In [43]:
```
"abc".find("c")
```

Out[43]: 2

In [44]:
```
"c" in "abc"   # returns the boolean
```

Out[44]: True

In [45]:
```
"abc".find("d")
```

Out[45]: -1

In [46]:
```
"d" in "abc"
```

Out[46]: False

In [47]:
```
"d" not in "abc"
```

Out[47]: True

String Concatenation

In [48]:
```
myString
```

Out[48]: 'fall 7 times stand up 8!'

In [49]:
```
myNewString = "It's time to wake up!!"
```

In [50]:
```
myString+myNewString
```

Out[50]: "fall 7 times stand up 8!It's time to wake up!!"

In [51]:
```
'#'.join([myString,myNewString])
```

Out[51]: "fall 7 times stand up 8!#It's time to wake up!!"

In [53]:
```
myString+'#'+myNewString
```

Out[53]: "fall 7 times stand up 8!#It's time to wake up!!"

In [55]:
```
print myString + str(1947)
```

```
fall 7 times stand up 8!1947
```

In [57]:
```
result = myString + str(True)+ str(' ')+ str(123.45)+ str(' ~!@#$%^&*')
```

In [58]:
```
print type(result), result
```

```
<type 'str'> fall 7 times stand up 8!True 123.45 ~!@#$%^&*
```

String Formating

Accessing Arguments by position

In [60]:
```
'{}, {} and {}'.format('a', 'b', 'c')
```

Out[60]:
```
'a, b and c'
```

In [61]:
```
'{}, {} and {}'.format('b', 'a', 'c')
```

Out[61]:
```
'b, a and c'
```

In [62]:
```
'{0}, {1} and {2}'.format('a', 'b', 'c')
```

Out[62]:
```
'a, b and c'
```

In [63]:
```
'{2}, {1} and {2}'.format('a', 'b', 'c')
```

Out[63]:
```
'c, b and c'
```

In [64]:
```
str1 = 'Yash'
```

In [65]:
```
str2 = 'Raj'
```

In [66]:
```
str3 = 'James'
```

In [67]:
```
print "The class is tuted by {0}, and attended by {1} and {2}".format(str1, st
r2, str3)
```

```
The class is tuted by Yash, and attended by Raj and James
```

In [68]:
```
print "The class is tuted by {0}, and attended by {1} and {2}".format(str2, st
r1, str3)
```

```
The class is tuted by Raj, and attended by Yash and James
```

In [69]:
```
'{2}, {1}, {0}'.format(*'abc')
```

Out[69]:
```
'c, b, a'
```

In [70]:
```
'{0}, {1}, {2}'.format(*'abc')
```

Out[70]:
```
'a, b, c'
```

accessing arguments by name

In [71]:
```
'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-1
15.81W')
```

Out[71]:
```
'Coordinates: 37.24N, -115.81W'
```

In [72]:
```
print "The class is tuted by {str1}, and attended by {str2} and
{str3}".format(str1="Udhay", str2= "Yash", str3= "James")
```

```
The class is tuted by Udhay, and attended by Yash and James
```

str() vs repr() | %s vs %r

Interview Question 3: what is the difference between str() and repr() ?

In [73]:
```
str9 = "C:\newFolder"
```

In [74]:
```
str9
```

Out[74]:
```
'C:\newFolder'
```

In [75]:
```
print str9
```

```
C:
ewFolder
```

In [76]:
```
str10 = "C:\\newFolder"   # to escape the \ character
```

In [77]:
```
str10
```

Out[77]:
```
'C:\\newFolder'
```

In [78]:
```
print str10
```

```
C:\newFolder
```

In [79]:
```
str11 = r"C:\newFolder"   #raw string representation
```

In [80]:
```
str11
```

Out[80]:
```
'C:\\newFolder'
```

In [81]:
```
print str11
```

```
C:\newFolder
```

In [82]:
```
type(str9), type(str10), type(str11)
```

Out[82]:
```
(str, str, str)
```

In [83]:
```
str12 = repr(str9)
```

In [84]:
```
str9, str12
```

Out[84]:
```
('C:\newFolder', "'C:\\newFolder'")
```

In [85]:
```
print str9
```

```
C:
ewFolder
```

In [86]:
```
print str12
```

```
'C:\newFolder'
```

In [87]:
```
type(str9), type(str12)
```

Out[87]:
```
(str, str)
```

repr() is representational, but results in string data type.

Interview Question 3: In which cases, repr() is preferred to str()?

str() is comfortable for Humans; whereas repr() is for the machine

In [88]:
```
"repr() shows quotes: {!r}; str() doesn't : {!s}".format('udhay', 'udhay')
```

Out[88]:
```
"repr() shows quotes: 'udhay'; str() doesn't : udhay"
```

In [89]:
```
str("udhay")
```

Out[89]:
```
'udhay'
```

In [90]:
```
repr("udhay")
```

Out[90]:
```
"'udhay'"
```

In [91]:
```
str('udhay')
```

Out[91]:
```
'udhay'
```

In [92]:
```
repr('udhay')
```

Out[92]:
```
"'udhay'"
```

In [93]:
```
print str("udhay"), repr("udhay"), str('udhay'), repr('udhay')
```

```
udhay 'udhay' udhay 'udhay'
```

In [94]:
```
name = repr("Udhay")
```

In [96]:
```
print "My name is ", name
```

```
My name is  'Udhay'
```

aligning the text and specifying a width

In [97]:
```
'{:<30}'.format('Python Programming')  # ljust
```

Out[97]:
```
'Python Programming            '
```

In [98]:
```
'{:>30}'.format('Python Programming')   # rjust
```

Out[98]:
```
'            Python Programming'
```

In [99]:
```
'{:^30}'.format('Python Programming') # center
```

Out[99]:
```
'      Python Programming      '
```

Logical Operations

In [100]:
```
a = 12;
```

In [101]:
```
a>9
```

Out[101]:
```
True
```

In [102]:
```
a<34
```

Out[102]:
```
True
```

In [103]:
```
(a>9) and (a<34)
```

Out[103]:
```
True
```

In [104]:
```
(a>9) and (a>34)
```

Out[104]:
```
False
```

In [105]:
```
(a>9) or (a>34)
```

Out[105]:
```
True
```

In [106]:
```
(a<9) or (a>34)
```

Out[106]:
```
False
```

In [107]:
```
not ((a<9) or (a>34))
```

Out[107]: True

In [108]:
```
(a<9), not (a<9)
```

Out[108]: (False, True)

Bitwise operations

In [110]:
```
4<<1       # bitwise left-shift - shifts their corresponding binary digits lef
t side by one position. # binary notation 8421
```

Out[110]: 8

In [111]:
```
4>>1       # bitwise right-shift
```

Out[111]: 2

In [112]:
```
4 & 8      # bitwise AND    # 4 - 0100        8 - 1000
```

Out[112]: 0

In [113]:
```
4 | 8      # bitwise OR
```

Out[113]: 12

In [114]:
```
4 ^ 8      # bitwise XOR
```

Out[114]: 12

In [115]:
```
4, ~4      # bitwise not
```

Out[115]: (4, -5)

Identity Operations

In [116]:
```
4 is 4
```

Out[116]: True

In [117]:
```
a = 12
```

In [118]:
```
b = 234
```

In [119]:
```
a is b
```

Out[119]: False

In [120]:
```
b = 12
```

In [121]:
```
a is b
```

Out[121]: True

Interview Question 4: why 'is' operator results in False for two objects with same value above 257, and True for values less than or equal to 256?

In [122]:
```
a = 257
```

In [123]:
```
b = 257
```

In [124]:
```
a is b
```

Out[124]: False

In [125]:
```
a = 256
```

In [126]:
```
b = 256
```

In [127]:
```
a is b
```

Out[127]: True

Assignment 3: Try this with float values, and try to find the boundary?

Boolean Operations

True, False

In [129]:
```
True
```

Out[129]: True

In [130]:
```
true            # interpreter treats it as an user-defined identifier
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-130-74d9a83219ca> in <module>()
----> 1 true

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

In [131]:
```
False
```

Out[131]: False

In [132]:
```
True is True
```

Out[132]: True

In [133]:
```
True == 1
```

Out[133]: True

In [134]:
```
True is 1
```

Out[134]: False

In [135]:
```
True is not 1
```

Out[135]: True

In [136]:
```
True is True
```

Out[136]: True

In [137]:
```
False is False
```

Out[137]: True

In [138]:
```
True is False
```

Out[138]: False

In [139]:
```
False is True
```

Out[139]: False

In [140]:
```
0 == False
```

Out[140]: True

In [141]:
```
bool(23>45)    # result in boolean value
```

Out[141]: False

In [142]:
```
bool('')   # empty or null elements result in False
```

Out[142]: False

In [143]:
```
bool('python')
```

Out[143]: True

In [145]:
```
bool(0), bool(1)
```

Out[145]: (False, True)

In [146]:
```
bool([]), bool({}), bool(())
```

Out[146]: (False, False, False)

In [147]:
```
bool([2,3]), bool({23, 34}), bool((122, 345))
```

Out[147]: (True, True, True)

order of evaluation

operators	descriptions
(), [], {}, ‘’	tuple, list, dictionnary, string
x.attr, x[], x[i:j], f()	attribute, index, slide, function call
+x, -x, ~x	unary negation, bitwise invert
**	exponent
*, /, %	multiplication, division, modulo
+, -	addition, substraction
<<, >>	bitwise shifts
&	bitwise and
^	bitwise xor
|	bitwise or
<, <=, >=, >,==, !=	comparison operators
is, is not, in,not in	comparison operators (continue)
not	boolean NOT
and	boolean AND
or	boolean OR
lambda	lambda expression

Conditional Operations

PEP 8 recommends 4 spaces for indentation

In [128]:
```
if 34<56:
    print '34<56'
```

```
34<56
```

In [149]:
```
if True:
    print 'Hello!'
```

```
Hello!
```

In [150]:
```
if False:
    print "It's False"
```

By default, the emphasis is True

In [151]:
```
if 1:
    print 'It is true'
else:
    print 'It is false'
```

```
It is true
```

In [153]:
```
a = False
```

In [154]:
```
if a == True:
    print 'It is true'
else:
    print 'It is false'
```

```
It is false
```

In [155]:
```
if a:                                   # equivalent to a == True
    print 'It is true'
else:
    print 'It is false'
```

```
It is false
```

In [157]:
```
if not a:
    print 'It is true'
else:
    print 'It is false'
```

```
It is true
```

In [158]:
```
a = 23
```

In [159]:
```
if a:
    print 'It is not zero'
else:
    print 'It is zero'
```

```
It is not zero
```

In [160]:
```
a = -23
```

In [161]:
```
if a:
    print 'It is not zero'
else:
    print 'It is zero'
```

```
It is not zero
```

In [162]:
```
a = False   # equivalent to a = 0
```

In [163]:
```
if a:
    print 'It is not zero'
else:
    print 'It is zero'
```

```
It is zero
```

Let us create a script for lottery game

In [164]:
```
#!/usr/bin/python

#class4_elif.py

#Lottery Game

lotteryNumber = 8997

number = int(raw_input('Please enter lottery number :'))

if number> lotteryNumber:
    print "You got a higher number"
elif number < lotteryNumber:
    print "You got a lower number"
else:
    print "COngrats! You won the lottery!!!"
```

```
Please enter lottery number :8997
COngrats! You won the lottery!!!
```

range() and xrange()

In [166]:
```
range(12)
```

Out[166]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

In [167]:
```
range(0, 12, 1)   # range(initialValue, finalValue, increment/decrement)
```

Out[167]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

In [169]:
```
print range(34)      # default initialValue = 0, and increment; increment = 1
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
```

In [170]:
```
range(34, 0)  # becoz default is increment; increment = 1
```

Out[170]:
```
[]
```

In [172]:
```
print range(34,0,-1)
```

```
[34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16,
 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [173]:
```
print range(0,34, 2)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32]
```

In [174]:
```
print range(0,34, 4)
```

```
[0, 4, 8, 12, 16, 20, 24, 28, 32]
```

In [175]:
```
print range(34, 4, -2)
```

```
[34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6]
```

In [176]:
```
print range(34, 4, -4)
```

```
[34, 30, 26, 22, 18, 14, 10, 6]
```

In [177]:
```
print range(34, -4, -4)
```

```
[34, 30, 26, 22, 18, 14, 10, 6, 2, -2]
```

In [178]:
```
print range(-44, -4, 4)
```

```
[-44, -40, -36, -32, -28, -24, -20, -16, -12, -8]
```

In [179]:
```
print range(-4, -44, -4)
```

```
[-4, -8, -12, -16, -20, -24, -28, -32, -36, -40]
```

range() will result in a list data type. It will store the resulting data in the buffer. For larger values, it will cost more buffer (heap) memory to store all those values. It is memory inefficient.

In [180]:
```
xrange(34)   # It is memory efficient. It will result in one value at one tim
e. Need to iterate to get the values, in it.
```

Out[180]:
```
xrange(34)
```

In [182]:
```
for i in xrange(34):
    print i,                                # , is a newline suppressor
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33
```

In [183]:
```
for i in xrange(6):
    print i
```

```
0
1
2
3
4
5
```

Both xrange() and range() will result the same end output

In [184]:
```
for i in range(6):
    print i
```

```
0
1
2
3
4
5
```