2020

Free Python Course

O. Elijah

onlinepythonclass@gmail.com

O. ELIJAH (MSc, Ph.D)

free python course

1 INTRODUCTION

Welcome to chapter four of the free python course. In this chapter, we shall look at reading complex data formats and outputting the formatted results to screen or file. Strings are simply created using single quotes or double quotes. Python treats single quotes the same as double quotes. First, we shall look at slicing string and followed by string manipulation.

What we intend to achieve

1. Learn how to slice strings

2. Learn how to manipulate stings with different built-in methods and functions in python

3 Learn how to format numbers

.

1.1 Review of chapter 3

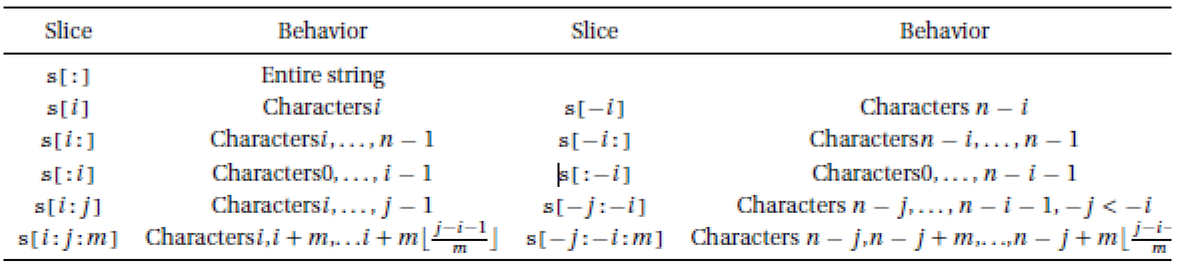
In chapter three, we looked at the different some of basic operators such as logical, bitwise and membership operators. We also studied on decision making and loops. I assume by now you know how to make use of *if...elif...else* for decision and you understand how to construct a for and while loop in python. Also, you should be conversant with the loop control statements such as *break, continue* and *pass.*

1.2 Slicing Strings

Slicing is used to access substrings within a string. Simply use the square brackets for slicing along with the index or indices to obtain your substring. The Table 3.1 shows the types of slices which can be used. The second column shows that slicing can use negative indices which essentially index the string backward.

Table 3.1 types of slicing

free python course



Slice	Behavior	Slice	Behavior
`s[:]`	Entire string		
`s[i]`	Characters$i$	`s[-i]`	Characters $n-i$
`s[i:]`	Characters$i,\ldots,n-1$	`s[-i:]`	Characters$n-i,\ldots,n-1$
`s[:i]`	Characters$0,\ldots,i-1$	`s[:-i]`	Characters$0,\ldots,n-i-1$
`s[i:j]`	Characters$i,\ldots,j-1$	`s[-j:-i]`	Characters $n-j,\ldots,n-i-1, -j<-i$
`s[i:j:m]`	Characters$i,i+m,\ldots i+m\lfloor\frac{j-i-1}{m}\rfloor$	`s[-j:-i:m]`	Characters $n-j,n-j+m,\ldots,n-j+m\lfloor\frac{j-i-}{m}$

Examples I will save the sentence **Python strings are sliceable.** in a variable called text. Note of you count the characters they are 29.

Codes	Output	Remarks
text = 'Python strings are sliceable.' print(text[0])	P	Prints out the character in index 0 of the string in variable text. Remember in indexing python starts at 0, so the last character will be indexed at 28
print(text[10])	i	Prints the character indexed at 10

L = len(text)
print(L)
print(text[L]) # Error
	
	29
	IndexError: string index out of range
		Determine the size of the array
		size of array in text is 29
		Shows an error because the last index is 28 (L-1)

print(text[L-1])	.	Prints the last character at index 28 which is .
print(text[:10])	Python str	Prints the first characters up to the index 10
print(text[10:])	ings are sliceable.	Prints the characters from index 10 to the last character

Note strings are immutable, and so it is not possible to replace part of a string. However, you can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.

Examples

```
var1 = 'Hello World!'

If I want to change the string to 'Hello Python'

var1[:6] = 'Python'  # Error because is not possible to replace part of a string, 
to do that you need to write

var1 = var1[:6] + 'Python'

print(var1)
```

Out: Hello Python

free python course

Escape Characters or sequence

Escape characters are used to arrange or format our characters in a special way.

Examples

I want to print the following sentence in python using the print function

Hello   John   # note the space between the two word is a tab

How are you   # Line 2

print('hello\tJohn how\nare you ')

```
hello    John how
are you
```

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline

free python course

\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

free python course

Special String operators

Adding Strings

Strings are concatenated using +.

```
a = ’Python is’
b = ’a rewarding language.’
Print(a + ’ ’ + b)
```

```
Out: ’Python is a rewarding language.’
```

While + is a simple method to join strings, the modern method is to use join. join is a string method which joins a list of strings (the input) using the object calling the string as the separator.

```
a = 'Python is'
b = 'a rewarding language.'
print( ' '.join([a,b]))
Out: Python is a rewarding language.
```

Alternatively, the same output may be constructed using an empty string ''.

```
a = 'Python is'
b = 'a rewarding language. '
print( ''.join([a, '  ',b]))
Out: Python is a rewarding language.
```

join is also useful for producing comma separated lists.

```
 words = ['Python','is','a','rewarding','language']
print(', '.join(words))
Out: Python,is,a,rewarding,language
```

Multiplying strings

Strings, like lists, can be repeated using *.

```
 a = ’Python is ’
print(2*a)
Out: Python is Python is
```

Formatting Numbers

Formatting numbers when converting to a string allows for automatic generation of tables and well formatted screen output. Numbers are formatted using the format function, which is used in conjunction with a format specifier. For example, consider these examples which format π.

```
 from math import pi
 pi
3.141592653589793
 '{:12.5f}'.format(pi)
'     3.14159'
 '{:12.5g}'.format(pi)
'      3.1416'
 '{:12.5e}'.format(pi)
' 3.14159e+00'
```

These all provide alternative formats and the difference is determined by the letter in the format string. The generic form of a format string is {*n*:*f a swc .p t* } or {*n*:*f a swcmt* }. To understand the various choices, consider the output produced by the basic output string '{0:}'

```
'{0:}'.format(pi)
'3.14159265359'
```

• *n* is a number 0,1,. . . indicating which value to take from the format function

```
 '{0:}, {1:} and {2:} are all related to pi'.format(pi,pi+1,2*pi)
’3.14159265359, 4.14159265359 and 6.28318530718 are all related to pi'
'{2:}, {0:} and {1:} reorder the output. '.format(pi,pi+1,2*pi)
'6.28318530718, 3.14159265359 and 4.14159265359 reorder the output. '
```

• *f a* are fill and alignment characters, typically a 2 character string. Fill may be any character except }, although space is the most common choice. Alignment can < (left) ,> (right), ^ (center) or = (pad to the right of the sign). Simple left 0-fills can omit the alignment character so that *f a* = 0.

```
>>> '{0:0<20}'.format(pi) # Left, 0 padding, precion 20
'3.141592653590000000'

>>> '{0:0>20}'.format(pi) # Right, 0 padding, precion 20
'00000003.14159265359'

>>> '{0:0^20}'.format(pi) # Center, 0 padding, precion 20
'0003.141592653590000'

>>> '{0: >20}'.format(pi) # Right, space padding, precion 20
'       3.14159265359'

>>> '{0:$^20}'.format(pi) # Center, dollar sign padding, precion 20
'$$$3.14159265359$$$$'
```

• *s* indicates whether a sign should be included. + indicates always include sign, - indicates only include if needed, and a blank space indicates to use a blank space for positive numbers, and a - sign for negative numbers – this format is useful for producing aligned tables.

```
>>> '{0:+}'.format(pi)
'+3.14159265359'

>>> '{0:+}'.format(-1.0 * pi)
'-3.14159265359'

>>> '{0:-}'.format(pi)
'3.14159265359'

>>> '{0: }'.format(pi)
' 3.14159265359'

>>> '{0: }'.format(-1.0 * pi)
'-3.14159265359'
```

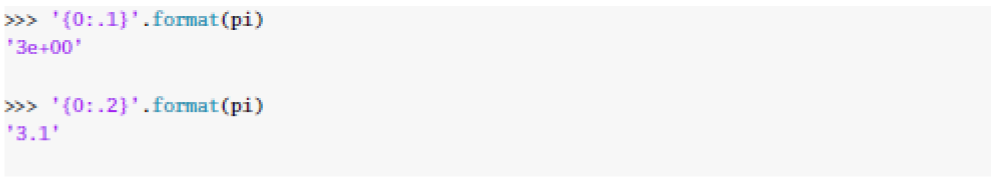
• *m* is the minimum total size of the formatted string

```
>>> '{0:10}'.format(pi)
'3.14159265359'

>>> '{0:20}'.format(pi)
'       3.14159265359'

>>> '{0:30}'.format(pi)
'                 3.14159265359'
```

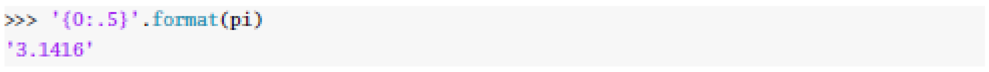
• *c* may be , or omitted. , produces numbers with 1000s separated using a ,. In order to use *c* it is necessary to include the . before the precision.

```
>>> '{0:.10}'.format(1000000 * pi)
'3141592.654'

>>> '{0:,.10}'.format(1000000 * pi)
'3,141,592.654'
```

• *p* is the precision. The interpretation of precision depends on *t*. In order to use *p*, it is necessary to include a . (dot). If not included, *p* will be interpreted as *m*.

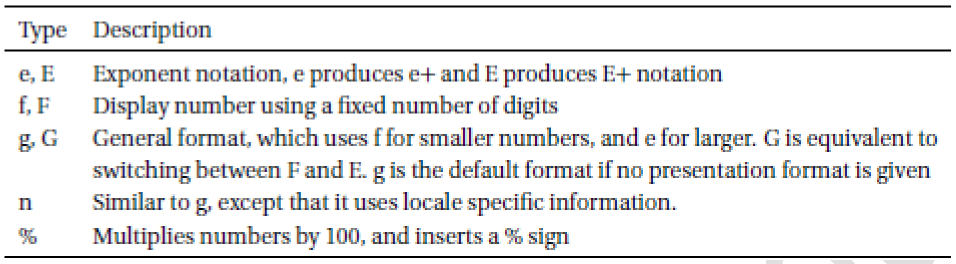


```
>>> '{0:.1}'.format(pi)
'3e+00'

>>> '{0:.2}'.format(pi)
'3.1'
```

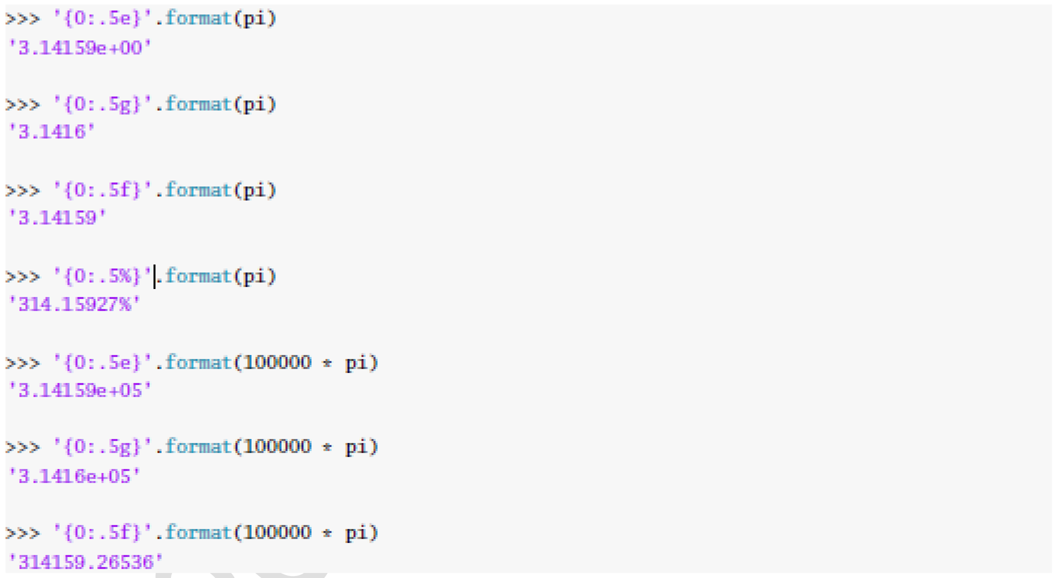


```
>>> '{0:.5}'.format(pi)
'3.1416'
```

• *t* is the type. Options include:

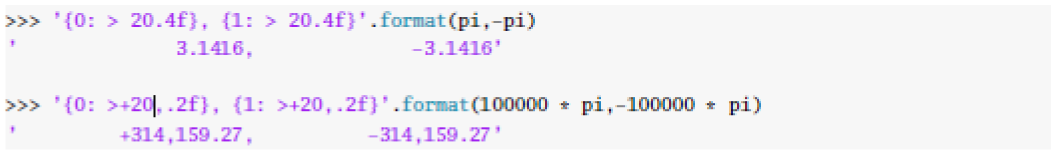


Type	Description
e, E	Exponent notation, e produces e+ and E produces E+ notation
f, F	Display number using a fixed number of digits
g, G	General format, which uses f for smaller numbers, and e for larger. G is equivalent to switching between F and E. g is the default format if no presentation format is given
n	Similar to g, except that it uses locale specific information.
%	Multiplies numbers by 100, and inserts a % sign



```
>>> '{0:.5e}'.format(pi)
'3.14159e+00'

>>> '{0:.5g}'.format(pi)
'3.1416'

>>> '{0:.5f}'.format(pi)
'3.14159'

>>> '{0:.5%}'.format(pi)
'314.15927%'

>>> '{0:.5e}'.format(100000 * pi)
'3.14159e+05'

>>> '{0:.5g}'.format(100000 * pi)
'3.1416e+05'

>>> '{0:.5f}'.format(100000 * pi)
'314159.26536'
```

Combining all of these features in a single format string produces complexly presented data.



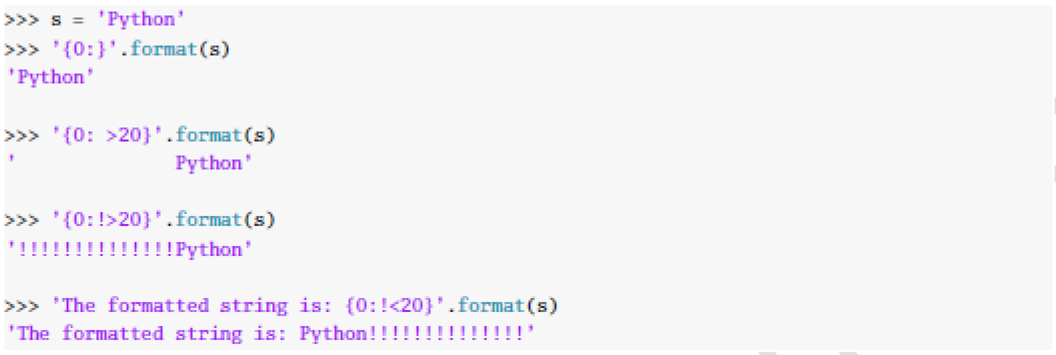
```
>>> '{0: > 20.4f}, {1: > 20.4f}'.format(pi,-pi)
'              3.1416,              -3.1416'

>>> '{0: >+20,.2f}, {1: >+20,.2f}'.format(100000 * pi,-100000 * pi)
'         +314,159.27,          -314,159.27'
```

In the first example, reading from left to right after the colon, the format string consists of:
1. Space fill (the blank space after the colon)
2. Right align (>)
3. Use no sign for positive numbers, - sign for negative numbers (the blank space after >)

4. Minimum 20 digits
5. Precision of 4 fixed digits
The second is virtually identical to the first, except that it includes a , to show the 1000s separator and a + to force the sign to be shown.

Formatting strings

format outputs formatted strings using a similar syntax to number formatting, although some options such as precision, sign, comma and type are not relevant.



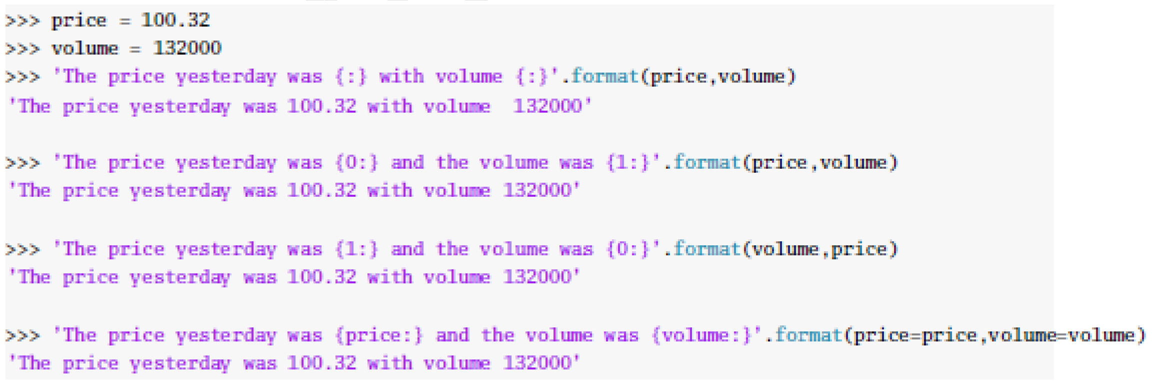
```
>>> s = 'Python'
>>> '{0:}'.format(s)
'Python'

>>> '{0: >20}'.format(s)
'              Python'

>>> '{0:!>20}'.format(s)
'!!!!!!!!!!!!!!Python'

>>> 'The formatted string is: {0:!<20}'.format(s)
'The formatted string is: Python!!!!!!!!!!!!!!'
```

Formatting multiple objects
format also formats multiple objects in the same string output. There are three methods to do this:

• No position arguments, in which case the objects are matched to format strings in order

• Numeric positional arguments, in which case the first object is mapped to '{0:}', the second to '{1:}', and so on.

• Named arguments such as '{price:}' and volume '{volume:}', which match keyword arguments inside format.



```
>>> price = 100.32
>>> volume = 132000
>>> 'The price yesterday was {:} with volume {:}'.format(price,volume)
'The price yesterday was 100.32 with volume  132000'

>>> 'The price yesterday was {0:} and the volume was {1:}'.format(price,volume)
'The price yesterday was 100.32 with volume 132000'

>>> 'The price yesterday was {1:} and the volume was {0:}'.format(volume,price)
'The price yesterday was 100.32 with volume 132000'

>>> 'The price yesterday was {price:} and the volume was {volume:}'.format(price=price,volume=volume)
'The price yesterday was 100.32 with volume 132000'
```

Old style format strings

Some Python code still uses an older style format string. Old style format strings have %(*map*)*flm.p t* , where:

• (*map*) is a mapping string containing a name, for example (price)

• *f l* is a flag which may be one or more of:

– 0: Zero pad

– (blank space)

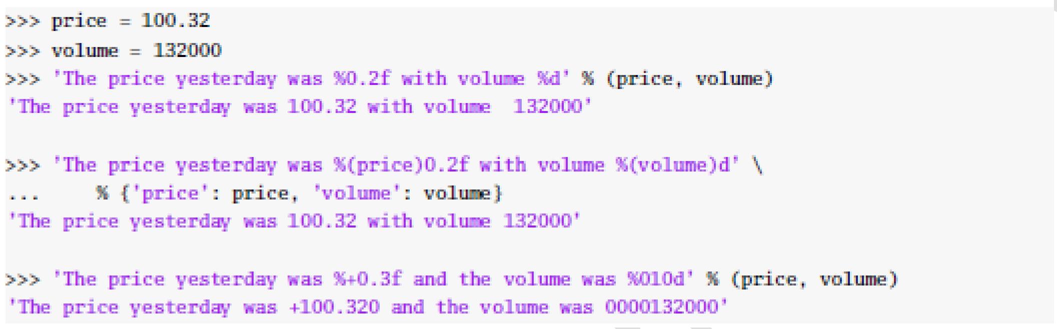
– Left

adjust output

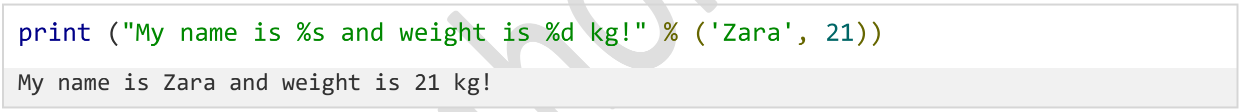
– + Include sign character

• *m*, *p* and *t* are identical to those of the new format strings.

In general, the old format strings should only be used when required by other code (e.g. matplotlib). Below are some examples of their use in strings.



```
>>> price = 100.32
>>> volume = 132000
>>> 'The price yesterday was %0.2f with volume %d' % (price, volume)
'The price yesterday was 100.32 with volume  132000'

>>> 'The price yesterday was %(price)0.2f with volume %(volume)d' \
...     % {'price': price, 'volume': volume}
'The price yesterday was 100.32 with volume 132000'

>>> 'The price yesterday was %+0.3f and the volume was %010d' % (price, volume)
'The price yesterday was +100.320 and the volume was 0000132000'
```



```
print ("My name is %s and weight is %d kg!" % ('Zara', 21))
My name is Zara and weight is 21 kg!
```

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer

%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

1.3 String Methods and Functions

There are several built-in methods in python that can be used to manipulate strings. See the list and examples as follows.

Methods and descriptions	Examples

capitalize()
Capitalizes first letter of string
	`str = "this is string example....wow!!!";`
	`print (str.capitalize())`
	Results
	`This is string example....wow!!!`

center(width, fillchar)
Returns a space-padded string with the original string centered to a total of width columns
	`str = "this is string example....wow!!!"`
	`print (str.center(40, 'a'))`
	Results
	`aaathis is string example....wow!!!aaaa`

free python course

count(str, beg= 0,end=len(string))
Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
 	
sub − This is the substring to be searched.
start − Search starts from this index. First character starts from 0 index. By default search starts from 0 index.
end − Search ends from this index. First character starts from 0 index. By default search ends at the last index.
	`str = "this is string example....wow!!!";`
	
	`sub = "i";`
	
	`print ("str.count(sub, 4, 40) : ", str.count(sub, 4, 40))`
	
	`str.count(sub, 4, 40) :  2`
	
	`sub = "wow";`
	
	`print ("str.count(sub) : ", str.count(sub))`
	
	Results
	
	`str.count(sub) :  1`

bytes.decode(encoding="utf-8", errors="strict")
Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
	`city = "Düsseldorf"`
	`utf8_encoded = city.encode('utf-8')`
	`print(type(utf8_encoded))  # bytes`
	`print(utf8_encoded)  # b'D\xc3\xbcsseldorf'`
	
	`decoded_city = utf8_encoded.decode('utf-8')`
	`print(type(decoded_city))  # str`
	`print(decoded_city)  # Düsseldorf`

str.encode(encoding="utf-8", errors="strict")
 	
Return a string decoded from the given bytes. Default encoding is 'utf-8'. errors may be given to set a different error handling scheme. The default for errors is 'strict', meaning that encoding errors raise a UnicodeError.
	`city = "Düsseldorf"`
	`utf8_encoded = city.encode('utf-8')`
	`print(type(utf8_encoded))  # bytes`
	`print(utf8_encoded)  # b'D\xc3\xbcsseldorf'`
	
	`decoded_city = utf8_encoded.decode('utf-8')`
	`print(type(decoded_city))  # str`
	`print(decoded_city)  # Düsseldorf`

endswith(suffix, beg=0, end=len(string))
Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
	`str = "this is string example....wow!!!";`
	
	`suffix = "wow!!!";`
	
	`print str.endswith(suffix)`
	
	`print str.endswith(suffix,20)`
	
	`suffix = "is";`

freepythoncourse

```
print (str.endswith(suffix, 2, 4))

print (str.endswith(suffix, 2, 6))
```

Results

```
True
True
True
False
```

expandtabs(tabsize=8)
Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.

```
str = "this is\tstring
example....wow!!!";

print ("Original string: " + str)

print ("Defualt exapanded tab: " +
str.expandtabs())

print ("Double exapanded tab: " +
str.expandtabs(16))
```

Results

```
Original string: this is        string
example....wow!!!
Defualt exapanded tab: this is string
example....wow!!!
Double exapanded tab: this is          string
example....wow!!!
```

find(str, beg=0 end=len(string))
Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

```
str1 = "this is string
example....wow!!!";

str2 = "exam";



print (str1.find(str2))

print (str1.find(str2, 10))

print (str1.find(str2, 40))

Results
```

```
15
15
-1
```

index(str, beg=0, end=len(string))
Same as find(), but raises an exception if str not found.

```
str1 = "this is string
example....wow!!!";

str2 = "exam";
```

free python course

	```
	print str1.index(str2)

	print str1.index(str2, 10)

	print str1.index(str2, 40)
	```
	Results
	```
	15
	15
	Traceback (most recent call last):
	   File "test.py", line 8, in
	   print str1.index(str2, 40);
	ValueError: substring not found

	shell returned 1
	```

isalnum()
Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
	```
	str = "this2009";  # No space in this string

	print (str.isalnum())


	str = "this is string example....wow!!!";

	print (str.isalnum())
	```
	Results
	```
	True
	False
	```

isalpha()
Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
	```
	str = "this2009";  # No space in this string

	print (str.isalnum())


	str = "this is string example....wow!!!";

	print (str.isalnum())
	```
	Results
	```
	True
	False
	```

isdigit()
Returns true if string contains only digits and false otherwise.
	```
	str = "123456";  # Only digit in this string

	print (str.isdigit())
	```

free python course

	`str = "this is string example....wow!!!";`
	
	`print (str.isdigit())`
	
	Results
	
	`True`
	`False`

islower()
Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
	`str = "THIS is string example....wow!!!";`
	
	`print (str.islower())`
	
	`str = "this is string example....wow!!!";`
	
	`print (str.islower())`
	
	Results
	
	`False`
	`True`

isnumeric()
Returns true if a unicode string contains only numeric characters and false otherwise.
	`str = u"this2009";`
	
	`print (str.isnumeric())`
	
	`str = u"23443434";`
	
	`print (str.isnumeric())`
	
	Results
	
	`False`
	`True`

isspace()
Returns true if string contains only whitespace characters and false otherwise
	`str = "       ";`
	
	`print (str.isspace())`
	
	`str = "This is string example....wow!!!";`
	
	`print (str.isspace())`
	
	Results
	
	`True`
	`False`

free python course

istitle()
Returns true if string is properly "titlecased" and false otherwise
	`str = "This Is String Example...Wow!!!";`
	
	`print (str.istitle())`
	
	`str = "This is string example....wow!!!";`
	
	`print (str.istitle())`
	
	Results
	
	`True`
	`False`

isupper()
Returns true if string is properly "titlecased" and false otherwise. Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise
	`str = "THIS IS STRING EXAMPLE....WOW!!!";`
	
	`print (str.isupper())`
	
	`str = "THIS is string example....wow!!!";`
	
	`print (str.isupper())`
	
	Results
	
	`True`
	`False`

join(seq)
Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string
	`s = "-";`
	
	`seq = ("a", "b", "c"); # This is sequence of strings.`
	
	`print (s.join( seq ))`
	
	`a-b-c`

len(string)
Returns the length of the string
	`str = "this is string example....wow!!!";`
	
	`print ("Length of the string: ", len(str))`
	
	Results
	
	`Length of the string:  32`

free python course

ljust(width[, fillchar])
Returns a space-padded string with the original string left-justified to a total of width columns.
	`str = "this is string example....wow!!!";`
	
	`print (str.ljust(50, '0'))`
	
	Results
	`this is string example....wow!!!000000000000000000`

lower()
Converts all uppercase letters in string to lowercase
	`str = "THIS IS STRING EXAMPLE....WOW!!!";`
	
	`print (str.lower())`
	
	Results
	`this is string example....wow!!!`

lstrip()
Removes all leading whitespace in string or returns a copy of the string in which all chars have been stripped from the beginning of the string
	`str = "     this is string example....wow!!!     ";`
	
	`print (str.lstrip())`
	
	`str = "88888888this is string example....wow!!!8888888";`
	
	`print (str.lstrip('8'))`
	
	Results
	`this is string example....wow!!!`
	`this is string example....wow!!!8888888`

maketrans()
Returns a translation table to be used in translate function.
	`intab = "aeiou"`
	
	`outtab = "12345"`
	
	`trantab = str.maketrans(intab, outtab)`
	
	`str = "this is string example....wow!!!"`
	
	`print (str.translate(trantab))`

max(str)
Returns the max alphabetical character from the string str.
	`str = "this is really a string example....wow!!!";`
	
	`print ("Max character: " + max(str))`

free python course

	```
	str = "this is a string
	example....wow!!!";

	print ("Max character: " + max(str))

	Results

	Max character: y
	Max character: x
	```

min(str)
Returns the min alphabetical character from the string str
	```
	str = "this-is-real-string-
	example....wow!!!";

	print ("Min character: " + min(str))


	str = "this-is-a-string-
	example....wow!!!";

	print ("Min character: " + min(str))

	Results

	Min character: !
	Min character: !
	```

replace(old, new [, max])
Replaces all occurrences of old in string with new or at most max occurrences if max given
	```
	str = "this is string example....wow!!!
	this is really string"

	print (str.replace("is", "was"))

	print (str.replace("is", "was", 3))

	Results

	thwas was string example....wow!!! thwas was
	really string
	thwas was string example....wow!!! thwas is
	really string
	```

rfind(str, beg=0,end=len(string))
Same as find(), but search backwards in string
	```
	str1 = "this is really a string
	example....wow!!!";

	str2 = "is";

	print (str1.rfind(str2))

	print (str1.rfind(str2, 0, 10))

	print (str1.rfind(str2, 10, 0))
	```

free python course

	```
	print (str1.find(str2))

	print (str1.find(str2, 0, 10))

	print (str1.find(str2, 10, 0))
	```
	Results
	```
	5
	5
	-1
	2
	2
	-1
	```

rindex(str, beg=0, end=len(string))
Same as index(), but search backwards in string
	```
	str1 = "this is string
	example....wow!!!";

	str2 = "is";

	print (str1.rindex(str2))

	print (str1.index(str2))
	```
	Results
	```
	5
	2
	```

rjust(width,[, fillchar])
Returns the string right justified in a string of length width. Padding is done using the specified fillchar (default is a space). The original string is returned if width is less than len(s).
	```
	str = "this is string
	example....wow!!!";

	print (str.rjust(50, '0'))
	```
	Results
	```
	000000000000000000this is string
	example....wow!!!
	```

rstrip()
Returns a copy of the string in which all chars have been stripped from the end of the string (default whitespace characters).
	```
	str = "     this is string
	example....wow!!!     ";

	print (str.rstrip())

	str = "88888888this is string
	example....wow!!!8888888";

	print (str.rstrip('8'))
	```
	Results
	```
	this is string example....wow!!!
	88888888this is string example....wow!!!
	```

free python course

split(str="", num=string.count(str))
Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given

```
str = "Line1-abcdef \nLine2-abc
\nLine4-abcd";

print (str.split( ))

print (str.split(' ', 1 ))

Results
```

```
['Line1-abcdef', 'Line2-abc', 'Line4-abcd']
['Line1-abcdef', '\nLine2-abc \nLine4-abcd']
```

splitlines(num=string.count('\n'))
Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed

```
str = "Line1-a b c d e f\nLine2- a b
c\n\nLine4- a b c d";

print (str.splitlines( ))

print (str.splitlines( 0 ))

print (str.splitlines( 3 ))

print (str.splitlines( 4 ))

print (str.splitlines( 5 ))

Results
```

```
['Line1-a b c d e f', 'Line2- a b c', '',
'Line4- a b c d']
['Line1-a b c d e f', 'Line2- a b c', '',
'Line4- a b c d']
['Line1-a b c d e f\n', 'Line2- a b c\n',
'\n', 'Line4- a b c d']
['Line1-a b c d e f\n', 'Line2- a b c\n',
'\n', 'Line4- a b c d']
['Line1-a b c d e f\n', 'Line2- a b c\n',
'\n', 'Line4- a b c d']
```

startswith(str, beg=0,end=len(string))
Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise

```
str = "this is string
example....wow!!!";

print (str.startswith( 'this' ))

print (str.startswith( 'is', 2, 4 ))

print (str.startswith( 'this', 2, 4 ))

Results
```

```
True
True
False
```

free python course

strip([chars])
Performs both lstrip() and rstrip() on string
	`str = "0000000this is string example....wow!!!0000000";`
	
	`print (str.strip( '0' ))`
	
	Results
	`this is string example....wow!!!`

swapcase()
Inverts case for all letters in string
	`str = "this is string example....wow!!!";`
	
	`print (str.swapcase())`
	
	`str = "THIS IS STRING EXAMPLE....WOW!!!";`
	
	`print (str.swapcase())`
	
	Results
	`THIS IS STRING EXAMPLE....WOW!!!`
	`this is string example....wow!!!`

title()
Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase
	`str = "this is string example....wow!!!";`
	
	`print str.title()`
	
	Results
	`This Is String Example....Wow!!!`

translate(table, deletechars="")
Translates string according to translation table str(256 chars), removing those in the del string
 	
example to delete 'x' and 'm' characters from the string
	`intab = "aeiou"`
	
	`outtab = "12345"`
	
	`trantab = str.maketrans(intab, outtab)`
	
	`str = "this is string example....wow!!!";`
	
	`print (str.translate(trantab))`
	
	Results
	`th3s 3s str3ng 2x1mpl2....w4w!!!`
	
	`str = "this is string example....wow!!!";`
	
	`print (str.translate(trantab, 'xm'))`
	`th3s 3s str3ng 21pl2....w4w!!!`

freepython course

upper()
Converts lowercase letters in string to uppercase
	```
	str = "this is string example....wow!!!";

	print ("str.capitalize() : ", str.upper())
	```
	`str.capitalize() :  THIS IS STRING EXAMPLE....WOW!!!`

zfill (width)
Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
	```
	str = "this is string example....wow!!!";

	print (str.zfill(40))

	print (str.zfill(50))

	Results
	```
	```
	00000000this is string example....wow!!!
	000000000000000000this is string example....wow!!!
	```

isdecimal()
Returns true if a unicode string contains only decimal characters and false otherwise
	```
	str = u"this2009";

	print (str.isdecimal();)

	str = u"23443434";

	print (str.isdecimal();)

	Results
	```
	```
	False
	True
	```

free python course

1.4 Applications

Example 1

We want to create an editor in python that allows us to format our text using a menu. When a user enters a value, the text will be formatted based on the number the user entered.

```
# Example 1 Creating an editor
# Program menu
print(" ********** MENU  **************")
print("  (1) Upper case                ")
print("  (2) Lower Case                ")
print("  (3) Titlecased                ")
print("  (4) Swap Case                 ")
print("  (5) Exit\n                    ")
print(" ********** MENU  **************")
menu = int(input(" Enter the option from the MENU:  "))



while (menu !=5):
    text = input(" Enter  your sentence:\n  ")
    if (menu == 1):
        print(text.upper()) # converts text to upper case
    elif (menu == 2):
        print(text.lower()) # converts text to lower case
    elif (menu == 3):
        print(text.title()) # converts text to titled case
    elif (menu == 4):
        print(text.swapcase()) # inverts the text to upper or lower case

    menu = int(input("Enter the option from the MENU  ")) # repeat the menu
print("Thank you and bye!") # exit the program
```

Result

```
********** MENU  **************
  (1) Upper case
  (2) Lower Case
  (3) Titlecased
  (4) Swap Case
  (5) Exit
```

free python course

```
 ********** MENU  **************

 Enter the option from the MENU:  1

 Enter  your sentence:
  I am enjoying python programming. I want to be a professional
I AM ENJOYING PYTHON PROGRAMMING. I WANT TO BE A PROFESSIONAL

Enter the option from the MENU
```

Example 2

I want to print a table of numbers in python using the tab and newline sequence

Names	Total score
John	165
Annis	168
Fatimah	163
Caleb	164
Aruna	153

There are different ways to do this. First method

free python course

```
# Example 2 print a table

#create a List of the table

record = [["John",165],["Annis",168],["Fatimah",163],["Caleb",164],["Aruna",153]]

print(record)

print("Names\t\t\tTotal score") # print the header

#print the rows and colums

for item in record:

    print(item[0], "\t\t\t",item[1])
```

Result

```
Names                    Total score
John                      165
Annis                     168
Fatimah                           163
Caleb                     164
Aruna                     153
```

Notice the Line three is not properly aligned. We can do this in a different way

```
# Example 2 print a table

#create a List of the table

record = [["John",165],["Annis",168],["Fatimah",163],["Caleb",164],["Aruna",153]]

print(record)

print("Names\t\tTotal score")

for item in record:

    print(item[0]," "*(14-len(item[0])), item[1])
```

Result

```
Names           Total score
John            165
Annis           168
Fatimah         163
Caleb           164
Aruna           153
```

free python course

Example 3

```
# Example 3 I want to write a program for the user to enter a random number if the
number is incorrect make a bell

from random import * # special library to generate random numbers

getrand = randint(1,10) # syetem to generate a random number between 1 and 10

print(getrand)  #you can comment this line if you don't want to see the random no.

num = int(input("Enter a random number between 1 and 10: "))

if (num == getrand):

    print(" well done correct number!")

else:

    print("\a")

    print("incorrect")

    print("The correct number is {:}, Please try again!!".format(getrand))
```

Result

```
Enter a random number between 1 and 10: 8
well done correct number!
```

free python course

1.5 Summary

In this chapter, we have looked at how to manipulate strings using the built-in methods and functions in Python. These include slicing of string, string manipulation and formatting numbers in string. Examples have been shown for purpose of reference.

1.6 Reference

[1] https://www.tutorialspoint.com/python/python_strings.htm

[2] https://www.w3resource.com/python-exercises/string/

[3] https://docs.python.org/3/library/stdtypes.html

1.7 Exercise

1. Write a program to request first name and last name from a user. Create an initial using the first and last name. Example first name is **JAMES** and last name is **BOND**. The initial should be **J.B**
   Hint: https://www.w3resource.com/python-exercises/string/python-data-type-string-exercise-3.php

2. Write a program that returns the length of the following sentence and prints the longest one
   "I am learning python programming"
   "Python program is simple"
3. Write a python program to get the last part of this string
   "https://www.tutorialspoint.com/python/python_strings.htm" before the character /

   Hint: https://www.w3resource.com/python-exercises/string/python-data-type-string-exercise-19.php
4. Write a Python program to print the following integers with zeros on the left of specified width 6
   1
   23
   450
   2456
5. Write a Python program to display the number 05 in left, right and center aligned of width 10
6. Write a Python program to display formatted text (width=20) as output.
   "Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time."

   Hint: https://www.w3resource.com/python-exercises/string/python-data-type-string-exercise-26.php
7. Write a Python program to swap comma and dot in a string "$456.045,00"

   Please refer to your notes and the provided links to answer the questions.

free python course

free python course