Appendix A: Working With Strings

In the table below, the notation [start, [end]] means *start* and *end* are optional parameters. If only one number is provided as the parameter, it is taken to be *start*.

marks the start of a comment
‘’’ marks the start and end of a multiline comment
=> marks the start of the output

The actual code is in `monotype` font.

Function	Description	Sample Code

count (sub, [start, [end]])	Return the number of times the substring *sub* appears in the string.
	
	This function is case-sensitive.
		# In the examples below, ‘s’ occurs at index 3, 6 and 10
		
		# count the entire string
		`‘This is a string’.count(‘s’)`
		=> 3
		
		# count from index 4 to end of string
		`‘This is a string’.count(‘s’, 4)`
		=> 2
		
		# count from index 4 to 10-1
		`‘This is a string’.count(‘s’, 4,10 )`
		=> 1
		
		# count ‘T’. There’s only 1 ‘T’ as the function is case sensitive.
		`‘This is a string’.count(‘T’)`
		`=> 1`

endswith (suffix, [start, [end]])	Return True if the string ends with the specified *suffix*, otherwise return False.
	
	suffix can also be a tuple of suffixes to look for.
	
	This function is case-sensitive.
		# ’man’ occurs at index 4 to 6
		
		# check the entire string
		`‘Postman’.endswith(‘man’)`
		=> True
		
		# check from index 3 to end of string
		`‘Postman’.endswith(‘man’, 3)`
		=> True
		
		# check from index 2 to 6-1

		`'Postman'.endswith('man', 2, 6)`
		=> False
		
		# check from index 2 to 7-1
		`'Postman'.endswith('man', 2, 7)`
		=> True
		
		# Using a tuple of suffixes (check from index 2 to 6-1)
		`'Postman'.endswith(('man', 'ma'), 2, 6)`
		=> True

find/index (sub, [start, [end]])	Return the index in the string where the first occurrence of the substring *sub* is found.
	
	find() returns -1 if *sub* is not found.
	
	index() returns ValueError is *sub* is not found.
	
	This function is case-sensitive.
		# check the entire string
		`'This is a string'.find('s')`
		=> 3
		
		# check from index 4 to end of string
		`'This is a string'.find('s', 4)`
		=> 6
		
		# check from index 7 to 11-1
		`'This is a string'.find('s', 7,11 )`
		=> 10
		
		# Sub is not found
		`'This is a string'.find('p')`
		=> -1
		
		`'This is a string'.index('p')`
		=> ValueError

isalnum()	Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.
	
	Alphanumeric does not include whitespaces.
		`'abcd1234'.isalnum()`
		=> True
		
		`'a b c d 1 2 3 4'.isalnum()`
		=> False
		
		`'abcd'.isalnum()`
		=> True
		
		`'1234'.isalnum()`
		=> True

isalpha()	Return true if all characters in the string are alphabetic	`'abcd'.isalpha()` => True

	and there is at least one character, false otherwise.	'abcd1234'.isalpha() => False '1234'.isalpha() => False 'a b c'.isalpha() => False
isdigit()	Return true if all characters in the string are digits and there is at least one character, false otherwise.	'1234'.isdigit() => True 'abcd1234'.isdigit() => False 'abcd'.isdigit() => False '1 2 3 4'.isdigit() => False
islower()	Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.	'abcd'.islower() => True 'Abcd'.islower() => False 'ABCD'.islower() => False
isspace()	Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.	' '.isspace() => True 'a b'.isspace() => False
istitle()	Return true if the string is a titlecased string and there is at least one character	'This Is A String'.istitle() => True 'This is a string'.istitle() => False
isupper()	Return true if all cased characters in the string are uppercase and there is at least one cased character,	'ABCD'.isupper() => True 'Abcd'.isupper()

	false otherwise.	=> False `'abcd'.isupper()` => False

join()	Return a string in which the parameter provided is joined by a separator.	`sep = '-'`
		`myTuple = ('a', 'b', 'c')`
		`myList = ['d', 'e', 'f']`
		`myString = "Hello World"`
		
		`sep.join(myTuple)`
		=> 'a-b-c'
		
		`sep.join(myTuple)`
		=> 'd-e-f'
		
		`sep.join(myString)`
		=> 'H-e-l-l-o- -W-o-r-l-d''

lower()	Return a copy of the string converted to lowercase.	`'Hello Python'.lower()` => 'hello python'

replace(old, new[, count])	Return a copy of the string with all occurrences of substring old replaced by new.
	
	count is optional. If given, only the first *count* occurrences are replaced.
	
	This function is case-sensitive.
		# Replace all occurences
		`'This is a string'.replace('s', 'p')`
		=> 'Thip ip a ptring'
		
		# Replace first 2 occurences
		`'This is a string'.replace('s', 'p', 2)`
		=> 'Thip ip a string'

split([sep [,maxsplit]])	Return a list of the words in the string, using *sep* as the delimiter string.
	
	sep and *maxsplit* are optional.
	
	If *sep* is not given, whitespace is used as the delimiter.
	
	If *maxsplit* is given, at most *maxsplit* splits are done.
		'''
		Split using comma as the delimiter
		Notice that there's a space before the words 'is', 'a' and 'string' in the output.
		'''
		`'This, is, a, string'.split(',')`
		=> ['This', ' is', ' a', ' string']
		
		# Split using whitespace as delimiter
		`'This is a string'.split()`
		=> ['This', 'is', 'a', 'string']
		
		# Only do 2 splits

	This function is case-sensitive.	`'This, is, a, string'.split(',' 2)` => ['This', ' is', ' a, string']

splitlines ([keepends])	Return a list of the lines in the string, breaking at line boundaries.
	
	Line breaks are not included in the resulting list unless *keepends* is given and true.
		# Split lines separated by \n
		`'This is the first line.\nThis is the second line'.splitlines()`
		=> ['This is the first line.', 'This is the second line.']
		
		# Split multi line string (e.g. string that uses the ‘’’ mark)
		`'''This is the first line. This is the second line.'''.splitlines()`
		=> ['This is the first line.', 'This is the second line.']
		
		# Split and keep line breaks
		`'This is the first line.\nThis is the second line.'.splitlines(True)`
		=> ['This is the first line.\n', 'This is the second line.']
		
		`'''This is the first line. This is the second line.'''.splitlines(True)`
		=> ['This is the first line.\n', 'This is the second line.']

startswith (prefix[, start[, end]])	Return True if string starts with the prefix, otherwise return False.
	
	prefix can also be a tuple of prefixes to look for.
	
	This function is case-sensitive.
		# ’Post’ occurs at index 0 to 3
		
		# check the entire string
		`'Postman'.startswith('Post')`
		=> True
		
		# check from index 3 to end of string
		`'Postman'.startswith('Post', 3)`
		=> False
		
		# check from index 2 to 6-1
		`'Postman'.startswith('Post', 2, 6)`
		=> False
		
		# check from index 2 to 6-1

		`'Postman'.startswith('stm', 2, 6)`
		=> True
		
		# Using a tuple of prefixes (check from index 3 to end of string)
		`'Postman'.startswith(('Post', 'tma'), 3)`
		=> True

strip ([chars])	Return a copy of the string with the leading and trailing characters *char* removed.
	
	If *char* is not provided, whitespaces will be removed.
	
	This function is case-sensitive.
		# Strip whitespaces
		`'   This is a string   '.strip()`
		=> 'This is a string'
		
		# Strip 's'. Nothing is removed since 's' is not at the start or end of the string
		`'This is a string'.strip('s')`
		=> 'This is a string'
		
		# Strip 'g'.
		`'This is a string'.strip('g')`
		=> 'This is a strin'

upper( )	Return a copy of the string converted to uppercase.	`'Hello Python'.upper()` => 'HELLO PYTHON'