Python - String Manipulation

Text is one of the most common forms of data your programs will handle. You already know how to concatenate two string values together with the + operator, but you can do much more than that.

Working with Strings

Double Quotes

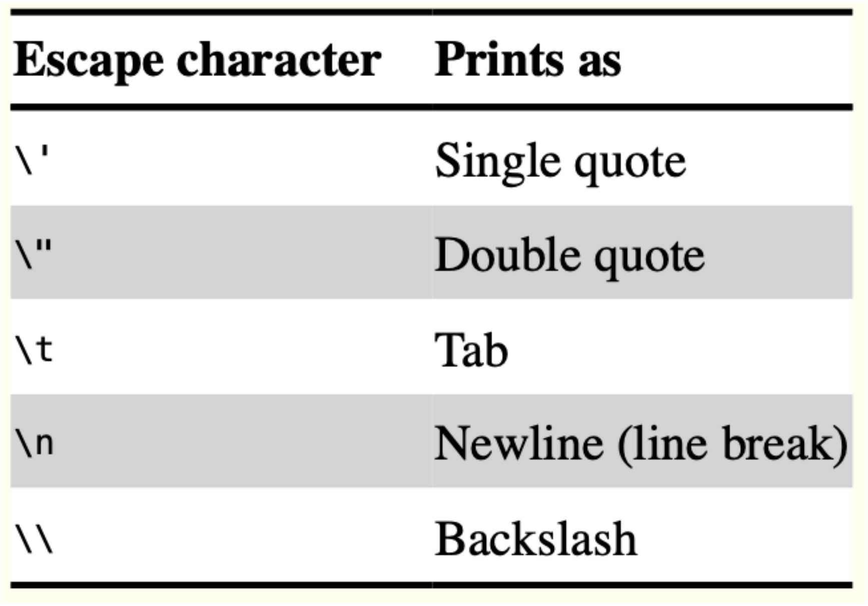
Strings can begin and end with double quotes, just as they do with single quotes. One benefit of using double quotes is that the string can have a single quote character in it.

```
>>> spam = "That is Alice's cat."
```

Escape Characters

An escape character consists of a backslash (`\`) followed by the character you want to add to the string.

```
>>> spam = 'Say hi to Bob\'s mother.'
```



Escape character	Prints as
`\'`	Single quote
`\"`	Double quote
`\t`	Tab
`\n`	Newline (line break)
`\\`	Backslash

Escape Characters

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

Raw Strings

You can place an `r` before the beginning quotation mark of a string to make it a raw string. A *raw string* completely ignores all escape characters and prints any backslash that appears in the string.

```
>>> print(r'That is Carol\'s cat.')
#That is Carol\'s cat.
```

Multiline Strings with Triple Quotes

While you can use the \n escape character to put a newline into a string, it is often easier to use multiline strings. A multiline string in Python begins and ends with either three single quotes or three double quotes. Any quotes, tabs, or newlines in between the "triple quotes" are considered part of the string.

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob''')
```

```
Dear Alice,


Eve's cat has been arrested for catnapping, cat burglary, and extortion.


Sincerely,

Bob
```

Output

💡 **Multiline Comments**
While the hash character (`#`) marks the beginning of a comment for the rest of the line, a multiline string is often used for comments that span multiple lines.

Indexing and Slicing Strings

Strings use indexes and slices the same way lists do. For instance, `'Hello, world!'` is a list and each character in the string as an item with a corresponding index.

```
'   H   e   l   l   o   ,       w   o   r   l    d    !    '
    0   1   2   3   4   5   6   7   8   9   10   11   12
```

```
[ start : end ]
```

inclusive   exclusive

The space and exclamation point are included in the character count, so `'Hello, world!'` is 13 characters long, from `H` at index 0 to `!` at index 12.

```
>>> spam = 'Hello, world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[7:]
'world!'
```

The in and not in Operators with Strings

The `in` and `not in` operators can be used with strings just like with list values.

```
>>> 'Hello' in 'Hello, World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello, World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

Putting Strings Inside Other Strings

Using *string interpolation*, in which the `%s` operator inside the string acts as a marker to be replaced by values following the string. One benefit of string interpolation is that `str()` doesn't have to be called to convert values to strings.

```
>>> name = 'Al'
>>> age = 4000
>>> 'My name is %s. I am %s years old.' % (name, age)
'My name is Al. I am 4000 years old.'
```

💡 Python 3.6 introduced *f-strings*, which is similar to string interpolation except that braces are used instead of %s, with the expressions placed directly inside the braces. Like raw strings, f-strings have an f prefix before the starting quotation mark.

```
>>> name = 'Al'
>>> age = 4000
>>> f'My name is {name}. Next year I will be {age + 1}.'
'My name is Al. Next year I will be 4001.'
```

Useful String Methods

The upper(), lower(), isupper(), and islower() Methods

The `upper()` and `lower()` string methods return a new string where all the letters in the original string have been converted to uppercase or lowercase, respectively.

```
>>> spam = 'Hello, world!'
>>> spam = spam.upper()
>>> spam
'HELLO, WORLD!'
>>> spam = spam.lower()
>>> spam
'hello, world!'
```

💡 The `isupper()` and `islower()` methods will return a Boolean `True` value if the string has at least one letter and all the letters are uppercase or lowercase, respectively.

The isX() Methods

Along with `islower()` and `isupper()`, there are several other string methods that have names beginning with the word *is*.

`isalpha()` Returns `True` if the string consists only of letters and isn't blank `isalnum()` Returns `True` if the string consists only of letters and numbers and is not blank

`isdecimal()` Returns `True` if the string consists only of numeric characters and is not blank

`isspace()` Returns `True` if the string consists only of spaces, tabs, and newlines and is not blank

`istitle()` Returns `True` if the string consists only of words that begin with an uppercase letter followed by only lowercase letters

```
+-------------+-----------+-------------+----------------------------------+
| isdecimal() | isdigit() | isnumeric() |          Example                 |
+-------------+-----------+-------------+----------------------------------+
|    True     |    True   |    True     | "038", "०३८", "０３８"             |
|  False      |    True   |    True     | "⁰³⁸", "🄁⒊⒏", "⓪③⑧"          |
|  False      |  False    |    True     | "↉⅛⅘", "ⅠⅢⅧ", "⑩⑬㊿", "壹貳參"  |
|  False      |  False    |  False      | "abc", "38.0", "-38"             |
+-------------+-----------+-------------+----------------------------------+
```

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters and numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters and numbers.')
```

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t
```

The startswith() and endswith() Methods

The `startswith()` and `endswith()` methods return `True` if the string value they are called on begins or ends (respectively) with the string passed to the method;

```
>>> 'Hello, world!'.startswith('Hello')
True
>>> 'Hello, world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
False
>>> 'Hello, world!'.startswith('Hello, world!')
True
>>> 'Hello, world!'.endswith('Hello, world!')
True
```

The join() and split() Methods

The `join()` method is useful when you have a list of strings that need to be joined together into a single string value.

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
```

```
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

The `split()` method does the opposite:

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

Splitting Strings with the partition() Method

The `partition()` string method can split a string into the text before and after a separator string.

```
>>> 'Hello, world!'.partition('w')
('Hello, ', 'w', 'orld!')
>>> 'Hello, world!'.partition('world')
('Hello, ', 'world', '!')
```

Justifying Text with the rjust(), ljust(), and center() Methods

```
#---------------------------
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello, World'.rjust(20)
'         Hello, World'
>>> 'Hello'.ljust(10)
'Hello     '

#---------------------------
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'

#---------------------------
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'

#---------------------------
```

💡 These methods are especially useful when you need to print tabular data that has correct spacing.

Removing Whitespace with the strip(), rstrip(), and lstrip() Methods

The `strip()` string method will return a new string without any whitespace characters at the beginning or end. The `lstrip()` and `rstrip()` methods will remove whitespace characters from the left and right ends, respectively.

```
>>> spam = '    Hello, World    '
>>> spam.strip()
'Hello, World'
>>> spam.lstrip()
'Hello, World    '
>>> spam.rstrip()
'    Hello, World'

#----------------------------
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Copying and Pasting Strings with the pyperclip Module

```
import pyperclip

>>> pyperclip.paste()

#'For example, if I copied this sentence to the clipboard and then called
#paste(), it would look like this:'
```

Practice Questions

1. What are escape characters?
2. What do the `\n` and `\t` escape characters represent?
3. How can you put a `\` backslash character in a string?
4. The string value `"Howl's Moving Castle"` is a valid string. Why isn't it a problem that the single quote character in the word `Howl's` isn't escaped?
5. If you don't want to put `\n` in your string, how can you write a string with newlines in it?
6. What do the following expressions evaluate to?
    - `'Hello, world!'[1]`
    - `'Hello, world!'[0:5]`
    - `'Hello, world!'[:5]`
    - `'Hello, world!'[3:]`
7. What do the following expressions evaluate to?
    - `'Hello'.upper()`
    - `'Hello'.upper().isupper()`
    - `'Hello'.upper().lower()`
8. What do the following expressions evaluate to?
    - `'Remember, remember, the fifth of November.'.split()`
    - `'-'.join('There can be only one.'.split())`
9. What string methods can you use to right-justify, left-justify, and center a string?
10. How can you trim whitespace characters from the beginning or end of a string?