Manipulating Strings

- https://www.pythoncheatsheet.org/cheatsheet/manipulating-strings

Escape characters

An escape character is created by typing a backslash \ followed by the character you want to insert.

Escape character	Prints as
`\'`	Single quote
`\"`	Double quote
`\t`	Tab
`\n`	Newline (line break)
`\\`	Backslash
`\b`	Backspace
`\ooo`	Octal value
`\r`	Carriage Return

In [12]:
```
print("Hello there!\nHow are you?\nI\'m doing fine.")
# Hello there!
# How are you?
# I'm doing fine.
```

```
Hello there!
How are you?
I'm doing fine.
```

Raw strings

- A raw string entirely ignores all escape characters and prints any backslash that appears in the string.
- Raw strings are mostly used for regular expression definition.

In [14]:
```
print(r"Hello there!\nHow are you?\nI\'m doing fine.")
# Hello there!\nHow are you?\nI\'m doing fine.
```

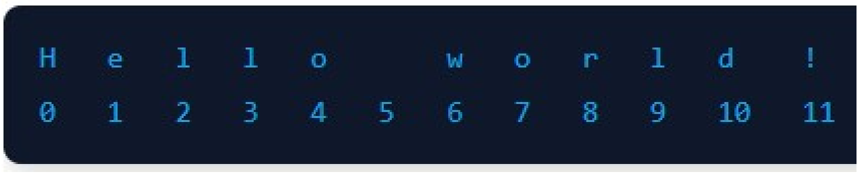
```
Hello there!\nHow are you?\nI\'m doing fine.
```

Multiline Strings

In [18]:
```
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
```

```
Dear Alice,

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

Sincerely,
Bob
```

Indexing and Slicing strings



Indexing

In [27]:
```
>>> spam = 'Hello world!'

>>> spam[0]
# 'H'
```

Out[27]:
```
'H'
```

In [29]:
```
>>> spam[4]
# 'o'
```

Out[29]:
```
'o'
```

In [31]:
```
>>> spam[-1]
# '!'
```

```
Out[31]: '!'
```

Slicing

```
In [38]: spam = 'Hello world!'

         spam[0:5]  #5 means n-1 i.e 4th index
         # 'Hello'

Out[38]: 'Hello'

In [40]: >>> spam[:5]
         # 'Hello'

Out[40]: 'Hello'

In [42]: >>> spam[6:]
         # 'world!'

Out[42]: 'world!'

In [44]: >>> spam[6:-1]
         # 'world'

Out[44]: 'world'

In [46]: >>> spam[:-1]
         # 'Hello world'

Out[46]: 'Hello world'

In [48]: >>> spam[::-1]
         # '!dlrow olleH'

Out[48]: '!dlrow olleH'

In [50]: >>> fizz = spam[0:5]
         >>> fizz
         # 'Hello'

Out[50]: 'Hello'
```

The in and not in operators

```
In [53]: >>> 'Hello' in 'Hello World'
         # True

Out[53]: True

In [55]: >>> 'Hello' in 'Hello'
         # True
```

```
Out[55]:  True
```

```
In [57]:  >>> 'HELLO' in 'Hello World'
          # False
```

```
Out[57]:  False
```

```
In [59]:  >>> '' in 'spam'
          # True
```

```
Out[59]:  True
```

```
In [61]:  >>> 'cats' not in 'cats and dogs'
          # False
```

```
Out[61]:  False
```

upper(), lower() and title()

Transforms a string to upper, lower and title case:

```
In [64]:  >>> greet = 'Hello world!'
          >>> greet.upper()
          # 'HELLO WORLD!'
```

```
Out[64]:  'HELLO WORLD!'
```

```
In [66]:  >>> greet.lower()
          # 'hello world!'
```

```
Out[66]:  'hello world!'
```

```
In [68]:  >>> greet.title()
          # 'Hello World!'
```

```
Out[68]:  'Hello World!'
```

isupper() and islower() methods

Returns True or False after evaluating if a string is in upper or lower case:

```
In [71]:  >>> spam = 'Hello world!'
          >>> spam.islower()
          # False
```

```
Out[71]:  False
```

```
In [73]:  >>> spam.isupper()
          # False
```

```
Out[73]:  False
```

In [77]:
```
>>> 'HELLO'.isupper()
# True
```

Out[77]:
```
True
```

In [79]:
```
>>> 'abc12345'.islower()
# True
```

Out[79]:
```
True
```

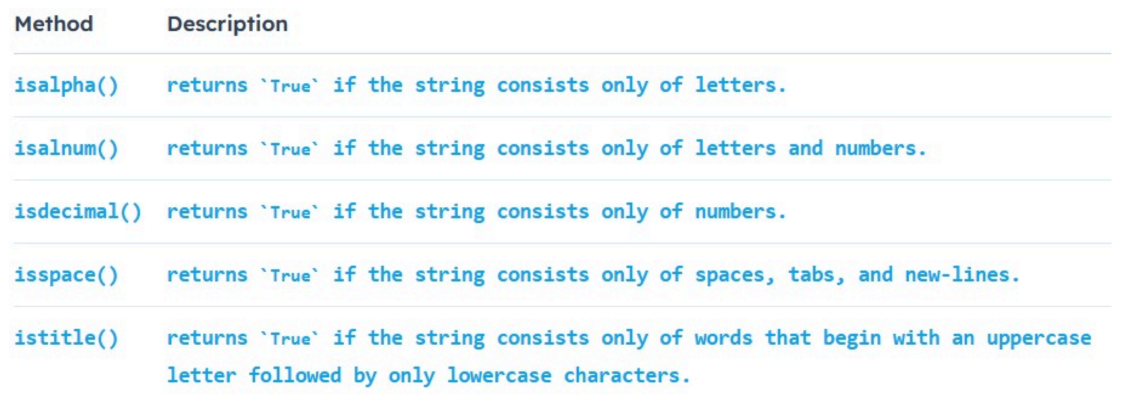
In [81]:
```
>>> '12345'.islower()
# False
```

Out[81]:
```
False
```

In [83]:
```
>>> '12345'.isupper()
# False
```

Out[83]:
```
False
```

The isX string methods



Method	Description
isalpha()	returns `True` if the string consists only of letters.
isalnum()	returns `True` if the string consists only of letters and numbers.
isdecimal()	returns `True` if the string consists only of numbers.
isspace()	returns `True` if the string consists only of spaces, tabs, and new-lines.
istitle()	returns `True` if the string consists only of words that begin with an uppercase letter followed by only lowercase characters.

startswith() and endswith()

In [88]:
```
>>> 'Hello world!'.startswith('Hello')
# True
```

Out[88]:
```
True
```

In [90]:
```
>>> 'Hello world!'.endswith('world!')
# True
```

Out[90]:
```
True
```

In [92]:
```
>>> 'abc123'.startswith('abcdef')
# False
```

Out[92]:
```
False
```

In [94]:
```
>>> 'abc123'.endswith('12')
# False
```
Out[94]:
```
False
```

In [96]:
```
>>> 'Hello world!'.startswith('Hello world!')
# True
```
Out[96]:
```
True
```

In [98]:
```
>>> 'Hello world!'.endswith('Hello world!')
# True
```
Out[98]:
```
True
```

join() and split()

- join()
- The join() method takes all the items in an iterable, like a list, dictionary, tuple or set, and joins them into a string. You can also specify a separator.

In [109…
```
''.join(['My', 'name', 'is', 'Simon'])   #''. is written before join, to define
'MynameisSimon'
```
Out[109…
```
'MynameisSimon'
```

In [111…
```
', '.join(['cats', 'rats', 'bats'])
# 'cats, rats, bats'
```
Out[111…
```
'cats, rats, bats'
```

In [113…
```
' '.join(['My', 'name', 'is', 'Simon'])
# 'My name is Simon'
```
Out[113…
```
'My name is Simon'
```

In [115…
```
'ABC'.join(['My', 'name', 'is', 'Simon'])
# 'MyABCnameABCisABCSimon'
```
Out[115…
```
'MyABCnameABCisABCSimon'
```

split()

The split() method splits a string into a list. By default, it will use whitespace to separate the items, but you can also set another character of choice:

In [119…
```
'My name is Simon'.split()
# ['My', 'name', 'is', 'Simon']
```

```
Out[119… ['My', 'name', 'is', 'Simon']
```

```
In [121… 'MyABCnameABCisABCSimon'.split('ABC')
         # ['My', 'name', 'is', 'Simon']
```

```
Out[121… ['My', 'name', 'is', 'Simon']
```

```
In [125… 'My??name??is??Simon'.split('??')
         # ['My', 'name', 'is', 'Simon']
```

```
Out[125… ['My', 'name', 'is', 'Simon']
```

```
In [127… 'My name is Simon'.split('m')
         # ['My na', 'e is Si', 'on']
```

```
Out[127… ['My na', 'e is Si', 'on']
```

```
In [129… ' My  name is  Simon'.split()
         # ['My', 'name', 'is', 'Simon']
```

```
Out[129… ['My', 'name', 'is', 'Simon']
```

```
In [131… ' My  name is  Simon'.split(' ')
         # ['', 'My', '', 'name', 'is', '', 'Simon']
```

```
Out[131… ['', 'My', '', 'name', 'is', '', 'Simon']
```

Justifying text with rjust(), ljust() and center()

```
In [141… 'Hello'.rjust(10)         #10 means right adjusting within 10 characters
         # '     Hello'
```

```
Out[141… '     Hello'
```

```
In [143… 'Hello'.rjust(20)         #20 means right adjusting within 20 characters
         # '               Hello'
```

```
Out[143… '               Hello'
```

```
In [139… 'Hello World'.rjust(20)
         # '         Hello World'
```

```
Out[139… '         Hello World'
```

```
In [149… 'Hello'.ljust(10)   #10 means here left adjusting within 10 characters
         # 'Hello     '
```

```
Out[149… 'Hello     '
```

```
In [151… 'Hello'.center(20)
         # '       Hello       '
```

```
Out[151…  '       Hello        '
```

An optional second argument to rjust() and ljust() will specify a fill character apart from a space character:

```
In [154…  'Hello'.rjust(20, '*')
          # '***************Hello'
```

```
Out[154…  '***************Hello'
```

```
In [156…  'Hello'.ljust(20, '-')
          # 'Hello---------------'
```

```
Out[156…  'Hello---------------'
```

```
In [158…  'Hello'.center(20, '=')
          # '=======Hello========'
```

```
Out[158…  '=======Hello========'
```

Removing whitespace with strip(), rstrip(), and lstrip()

- The strip() method in Python removes leading and trailing characters from a string. By default, it removes whitespace characters (spaces, tabs, newlines). It can also be used to remove other specified characters.
- The rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.
- The lstrip() method removes any leading characters (space is the default leading character to remove)

```
In [1]:  spam = '    Hello World     '
         spam.strip()
         # 'Hello World'
```

```
Out[1]:  'Hello World'
```

```
In [3]:  spam.lstrip()
         # 'Hello World     '
```

```
Out[3]:  'Hello World     '
```

```
In [5]:  spam.rstrip()
         # '    Hello World'
```

```
Out[5]:  '    Hello World'
```

```
In [7]:  spam = 'SpamSpamBaconSpamEggsSpamSpam'
         spam.strip('ampS')
```

```
# 'BaconSpamEggs'
```

Out[7]:
```
'BaconSpamEggs'
```

The Count Method

Counts the number of occurrences of a given character or substring in the string it is applied to. Can be optionally provided start and end index.

In [16]:
```
sentence = 'one sheep two sheep three sheep four'
sentence.count('sheep')
# 3
```

Out[16]:
```
3
```

In [18]:
```
sentence.count('e')
# 9
```

Out[18]:
```
9
```

In [22]:
```
sentence.count('e', 6)   #6th index means n-1th index i.e 5 i.e after h
# 8
# returns count of e after 'one sh' i.e 6 chars since beginning of string
```

Out[22]:
```
8
```

In [24]:
```
sentence.count('e', 7)
# 7
```

Out[24]:
```
7
```

Replace Method

Replaces all occurences of a given substring with another substring. Can be optionally provided a third argument to limit the number of replacements. Returns a new string.

In [28]:
```
text = "Hello, world!"
text.replace("world", "planet")
# 'Hello, planet!'
```

Out[28]:
```
'Hello, planet!'
```

In [30]:
```
fruits = "apple, banana, cherry, apple"
fruits.replace("apple", "orange", 1)
# 'orange, banana, cherry, apple'
```

Out[30]:
```
'orange, banana, cherry, apple'
```

In [32]:
```
sentence = "I like apples, Apples are my favorite fruit"
sentence.replace("apples", "oranges")
# 'I like oranges, Apples are my favorite fruit'
```

```
Out[32]: 'I like oranges, Apples are my favorite fruit'
```

```
In [ ]:
```