Strings

Strings

Strings are sequence of characters

In python specifically, strings are a sequence of Unicode Characters

- Create
- Accessing
- Editing
- Deleting
- Operations
- String Functions

Creating

```
# Creating
c = 'Hello'
```

Accessing

```
# Accessing
- INDEXING

c = 'Hello'

# H e l l o
# 0 1 2 3 4

print(c[0]) - H - print(c[-5])
print(c[1]) - e - print(c[-4])
print(c[2]) - l - print(c[-3])
print(c[3]) - l - print(c[-2])
print(c[4]) - o - print(c[-1])
```

Slicing

```
# Slicing
c = 'Hello World'

print(c[0:4]) - Hello
print(c[2:]) - llo World
print(c[:4]) - Hell
print(c[2:6:2]) - lo
print(c[0:8:3]) - HlW
print(c[0:6:-1]) - empty string
print(c[-5:-1:2]) - Wr
print(c[::-1]) - dlroW olleH # Reverse the string
print(c[-1:-5:-1]) - dlro
```

Editing & Deleting

Strings are a Immutable Data Type

```
# Editing
c = 'Hello World'

# Deletion
del c
print(c) - NameError

del c[0] - TypeError #because we cant mutate string.
```

Operations on Strings

- Arithmetic
- Relational
- Logical

- Loops
- Membership

```
#Arithmetic
'str1' + 'str2' # str1str2
```

```
# Relational
'Hello' == 'World' #False

'Mumbai' > 'Pune' # False because in alphabet M come first than P P > M

'kol' < 'Kol' # False as capital letters comes first thats why K < k
```

```
#Loops
c = 'hello'
for i in c:
        print(i)
h
e
l
l
o
```

String Function

Common Function

Len

```
c = 'hello'
len(c)  - 5
```

Max / Min

```
max(c) - o
min(c) - e
```

Sorted

```
sorted(c, reverse=True) - o, l, l, h, e
sorted(c) - e, h, l, l, o
```

Capitalize/Title/Upper/Lower

```
# Capitalize only make first alphabet of the string into a capital letter.
# It will not make any change in the original string.
```

```
c.capitalize() # Hello
c # hello
```

```
# Title Make first letter of every into capital

a = 'hello my name is aditya'
a.title() # Hello My Name Is Aditya
```

```
c.upper() # HELLO

b = 'NAME'
b.lower() # name
```

Count

`count()` method returns the number of times a specified value appears in the string.

```
string.count(value, start, end)
```

Parameter	Description
value	Required. A String. The string to value to search for
start	Optional. An Integer. The position to start the search. Default is 0
end	Optional. An Integer. The position to end the search. Default is the end of the string

```
a = 'hello my name is aditya'
a.count('a') # 3
```

- Search from position

```
txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple", 10, 24)

print(x) # 1
```

Find/Index

The `find()` method finds the first occurrence of the specified value.

The `find()` method returns -1 if the value is not found.

```
string.find(value, start, end)
```

```
'it is raining'.find('raining') # 6

'it is raining'.find('x') # -1
```

The `index()` method finds the first occurrence of the specified value.

The `index()` method raises an exception if the value is not found.

```
'it is raining'.index('raining') # 6

'it is raining'.index('x') # ValueError: substring not found
```

Endswith/Startswith

```
'it is raining'.endswith('jel') # False

'it is raining'.startswit('it') # True
```

Format

```
#named indexes:
txt1 = "My name is {fname}, I'm {age}".format(fname = "John", age = 36)
#numbered indexes:
txt2 = "My name is {0}, I'm {1}".format("John",36)
#empty placeholders:
txt3 = "My name is {}, I'm {}".format("John",36)

print(txt1) # My name is John, I'm 36
print(txt2) # My name is John, I'm 36
print(txt3) # My name is John, I'm 36

txt4 = "My name is {1}, I'm {0}".format("John",36)

print(txt4) # My name is 36, I'm John
```

isalnum / isalpha / isdecimal / isdigit / isidentifier

`isalnum`

- The `isalnum()` method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).
- Example of characters that are not alphanumeric: (space)!#%&? etc.

```
txt = "Company 12"

x = txt.isalnum()

print(x) # False
```

`isalpha`

- The `isalpha()` method returns True if all the characters are alphabet letters (a-z).
- Example of characters that are not alphabet letters: (space)!#%&? etc.

```
txt = "CompanyX"
txt2 = 'Company10'

x = txt.isalpha()
y = txt2.isalpha()
print(x, y) # True False
```

`isdigit`

- The `isdigit()` method returns True if all the characters are digits, otherwise False.
- Exponents, like ², are also considered to be a digit.

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for ²

print(a.isdigit()) # True
print(b.isdigit()) # True


txt = "50800"

x = txt.isdigit()

print(x) #True
```

`isdecimal`

- The `isdecimal()` method returns True if all the characters are decimals ()0-9).
- This method can also be used on Unicode objects.

```
a = "\u0030" #unicode for 0
b = "\u0047" #unicode for G

print(a.isdecimal()) # True
print(b.isdecimal()) # False
```

`isidentifier`

- The `isidentifier()` method returns True if the string is a valid identifier, otherwise False.
- A string is considered a valid identifier if it only contains alphanumeric letters (a-z) and (0-9), or underscores (_). A valid identifier cannot start with a number, or contain any spaces.

```
a = "MyFolder"
b = "Demo002"
c = "2bring"
d = "my demo"
```

```
print(a.isidentifier()) # True
print(b.isidentifier()) # True
print(c.isidentifier()) # False
print(d.isidentifier()) # False
```

`isnumeric`

- The `isnumeric()` method returns True if all the characters are numeric (0-9), otherwise False.
- Exponents, like ² and ¾ are also considered to be numeric values.
- `"-1"` and `"1.5"` are NOT considered numeric values, because *all* the characters in the string must be numeric, and the `-` and the `.` are not.

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for &sup2;
c = "10km2"
d = "-1"
e = "1.5"

print(a.isnumeric()) # True
print(b.isnumeric()) # True
print(c.isnumeric()) # False
print(d.isnumeric()) # False
print(e.isnumeric()) # False
```

Split

- The `split()` method splits a string into a list.
- You can specify the separator, default separator is any whitespace.

```
string.split(separator, maxsplit)
```

Parameter	Description
separator	Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator
maxsplit	Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

```
txt = "welcome to the jungle"

x = txt.split()

print(x) # ['welcome', 'to', 'the', 'jungle']
```

```
txt = "hello, my name is Peter, I am 26 years old"

x = txt.split(", ")

print(x) # ['hello', 'my name is Peter', 'I am 26 years old']
```

```
txt = "apple#banana#cherry#orange"

x = txt.split("#")

print(x) # ['apple', 'banana', 'cherry', 'orange']
```

```
txt = "apple#banana#cherry#orange"

# setting the maxsplit parameter to 1, will return a list with 2 elements!
x = txt.split("#", 1)

print(x) # ['apple', 'banana#cherry#orange']
```

Join

- The `join()` method takes all items in an iterable and joins them into one string.
- A string must be specified as the separator.

```
myDict = {"name": "John", "country": "Norway"}
mySeparator = "TEST"

x = mySeparator.join(myDict)

print(x) # nameTESTcountry
```

💡 **Note:** When using a dictionary as an iterable, the returned values are the keys, not the values.

```
myTuple = ("John", "Peter", "Vicky")

x = "#".join(myTuple)

print(x) # John#Peter#Vicky
```

Replace

The `replace()` method replaces a specified phrase with another specified phrase.

```
string.replace(oldvalue, newvalue, count)
```

Parameter	Description
oldvalue	Required. The string to search for
newvalue	Required. The string to replace the old value with
count	Optional. A number specifying how many occurrences of the old value you want to replace. Default is all occurrences

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three")

print(x) # three three was a race horse, two two was three too.
```

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three", 2)

print(x) # three three was a race horse, two two was one too.
```

Strip

- The `strip()` method removes any leading, and trailing whitespaces.
- Leading means at the beginning of the string, trailing means at the end.
- You can specify which character(s) to remove, if not, any whitespaces will be removed.

```
string.strip(characters)
```

```
txt = ",,,,,rrttgg.....banana....rrr"

x = txt.strip(",.grt")

print(x) # banana
```

- The argument `",.grt"` specifies the characters to be removed. This means that the commas, dots, and the letters "g", "r", and "t" will be removed from the beginning and end of the string.

  ```
  txt = ",,,,,rrztgg.....banana....rrr"

  x = txt.strip(",.grt")

  print(x) # ztgg.....banana
  ```

```
txt = "     banana     "

x = txt.strip()

print("of all fruits", x, "is my favorite")
# of all fruits banana is my favorite
```