



Main String Operations



Table of Contents



- ▶ Immutability
- ▶ Searching a String
- ▶ Changing a String
- ▶ Editing a String



1 Immutability

```
best = 'clarusway'
```

```
best.upper()
```

```
best.title()
```

The pre-class material is clear enough.



Students choose an option

Pear Deck Interactive Slide
Do not remove this bar



Introduction

- ▶ String processing is of **great importance** in the IT world. Moreover, there seems to be much more to do in this field.
- ▶ Good news! Python's **string** processing **skills** are very **advanced** that we will focus on some main parts of it.
- ▶ A significant thing to keep in mind is that **str** is an **immutable** data type. This means you can't just change the string **in place**, so most string methods return a copy of the string. Well, how can we do that?



Introduction



Tips:

- This is the way you should follow: You must create a new variable for the copy you made or assign the same name to the copy to save the changes made to the string for later use.



Introduction

```
var_string = 'ClarusWay'  
print(var_string.lower())  
print(var_string)  
var_string = 'ClarusWay'.lower()  
print(var_string)
```

We can't change the string

To change string, we should
reassign the new (changed)
string value to a variable

```
clarusway  
ClarusWay  
clarusway
```



Introduction

```
var_str = 'In God we Trust'  
var_str.lower()  
print(var_str)
```

What is the output? Try to figure out in your mind...





Introduction

```
var_str = 'In God we Trust'  
var_str.lower()  
print(var_str)
```



We couldn't change the string

In God we Trust



2

Searching a String



Searching a String

In & Not in operators

Operation	Result
<code>x in s</code>	True if an item of <code>s</code> is equal to <code>x</code> , else False
<code>x not in s</code>	False if an item of <code>s</code> is equal to <code>x</code> , else True

```
e_mail = 'bulent@clarusway.com'  
print ('@' in e_mail)
```



▶ Searching a String

- ▶ To search patterns in a string there are two useful methods called `.startswith()` and `.endswith()` that search for the particular pattern in the immediate beginning or end of a string and return `True` if the expression is found.



▶ Searching a String

- ▶ To search patterns in a string there are two useful methods called `.startswith()` and `.endswith()` that search for the particular pattern in the immediate beginning or end of a string and return `True` if the expression is found.

- `string.startswith()`

Starts searching from the **beginning** to the end.

- `string.endswith()`

Starts searching from the **very end** to the beginning.



Searching a String(review pre-class)



- ▶ Let's take a look these pre-class examples :

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('.com'))  
3 print(text.startswith('http:'))  
4
```

What is the output? Try to figure out in your mind...





▶ Searching a String(review pre-class)

- ▶ Let's take a look these pre-class examples :

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('.com'))  
3 print(text.startswith('http:'))  
4
```

```
1 True  
2 False  
3
```



Searching a String(review pre-class)



- ▶ Let's take a look these pre-class examples :

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('.com'))  
3 print(text.startswith('http:'))  
4
```

```
1 True  
2 False  
3
```

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('om'))  
3 print(text.startswith('w'))  
4
```

What is the output? Try to figure out in your mind...





Searching a String(review pre-class)



- ▶ Let's take a look these pre-class examples :

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('.com'))  
3 print(text.startswith('http:'))  
4
```

```
1 True  
2 False  
3
```

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('om'))  
3 print(text.startswith('w'))  
4
```

```
1 True  
2 True  
3
```



Searching a String(review pre-class)



- ▶ These methods have optional arguments **start** and **end**. We can specify the search by adding arguments so that the area of search is delimited by **start** and **end** arguments.

The formula syntaxes are :

- `.startswith(prefix[, start[, end]])`
- `.endswith(suffix[, start[, end]])`



Searching a String(review pre-class)



- ▶ These methods have optional arguments **start** and **end**. We can specify the search by adding arguments so that the area of search is delimited by **start** and **end** arguments.

Tips:

- Remember! Characters of string count from left to right and start with zero.
- `.startswith(prefix[, start[, end]])`
- `.endswith(suffix[, start[, end]])`



Searching a String(review pre-class)



- ▶ These methods have optional arguments **start** and **end**. We can specify the search by adding arguments so that the area of search is delimited by **start** and **end** arguments.

Tips:

- Remember! Characters of string count from left to right and start with zero.

- `.startswith(prefix[, start[, end]])`

- `.endswith(suffix[, start[, end]])`

ends not included



Searching a String(review pre-class)



- ▶ Consider this pre-class example :

```
1 email = "clarusway@clarusway.com is my e-mail address"  
2 print(email.startswith("@", 9))  
3 print(email.endswith("-", 10, 32))  
4 |
```

What is the output? Try to figure out in your mind...





Searching a String(review pre-class)

- Consider this example :

```
1 email = "clarusway@clarusway.com is my e-mail address"  
2 print(email.startswith("@", 9))  
3 print(email.endswith("-", 10, 32))  
4
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
c	l	a	r	u	s	w	a	y	@	c	l	a	r	u	s	w	a	y	.	c	o	m
	i	s		m	y		e	-	m	a	i	l		a	d	d	r	e	s	s		
23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42			



Searching a String

- Consider this example :

```
1 email = "clarusway@clarusway.com is my e-mail address"  
2 print(email.startswith("@", 9))  
3 print(email.endswith("-", 10, 32))  
4 |
```

```
1 True  
2 True  
3
```

"clarusway@clarusway.com is my e-mail address"

@ is the 9th and m is the 32nd
letter starting from zero



Searching a String

- Try to figure out the output of this code :

```
1 text = 'www.clarusway.com'  
2 print(text.endswith('.co'))  
3 print(text.startswith('w.'))  
4  
5  
6 |
```





Searching a String

- ▶ The output :

```
1 text = 'www.clarusway.com'
2 print(text.endswith('.co'))
3 print(text.startswith('w.'))
4
5
6 |
```

Output

```
False
False
```



Searching a String

- ▶ To **count words** in a sentence : with **count()** method.

```
1 text = ""
2 Data preprocessing is an important task in text classification.
3 This program counts words in a sentence. It starts with space separated words.
4 ""
5 print("To count words in a sentence : ", text.count(" ") + 1)
```

space

What is the output? Try to figure out in your mind...



Students, write your response!

REINVENT YOURSELF

Pear Deck Interactive Slide

Do not remove this bar

26



▶ Searching a String

- ▶ `str.find()`: The `find()` method returns the index of the first occurrence of a substring in the given string (case-sensitive). If the substring is not found it returns **-1**.
- ▶ also `str.rfind()`, **starts the search from the right.**
- ▶ `str.index()`: Like `find()`, but raise `ValueError` when the substring is not found.
- ▶ also `str.rindex()`, Like `rfind()` but raises `ValueError` when the substring *sub* is not found.



▶ Searching a String

```
1 text = 'www.clarusway.com'
2 print(text.index('com'))
3 print(text.find('com'))
```

14

14



3

Changing a String



Changing a String

- ▶ The changing a string methods return the copy of the `str` with some changes made.
- ▶ How does the syntax work?
- ▶ A string is given first (or the name of a variable that represents a string), then comes a period followed by the method name and parentheses in which arguments are listed.
- ▶ The formula syntax 🙌

```
string.method(arguments)
```



Changing a String

- ▶ The summary of some common and the most important methods are as follows 🙋

- `str.replace(old, new[, count])` replaces all occurrences of old with the new.

The count argument is optional, and if the optional argument count is given, only the first count occurrences are replaced. `count`: Maximum number of occurrences to replace. `-1` (the default value) means replace all occurrences.

- `str.swapcase()` converts upper case to lower case and vice versa.
- `str.capitalize()` changes the first character of the string to the upper case and the rest to the lower case.
- `str.upper()` converts all characters of the string to the upper case.
- `str.lower()` converts all characters of the string to the lower case.
- `str.title()` converts the first character of each word to upper case.



Changing a String

- ▶ Let's grasp these methods through the examples 🙋

```
1 sentence = "I live and work in Virginia"
2
3 print(sentence.upper())
4
5 print(sentence.lower())
6
7 print(sentence.swapcase())
8
9 print(sentence) # note that, source text is unchanged
10
```

What is the output? Try to figure out in your mind...





Changing a String

- ▶ Let's grasp these methods through the examples 🙌

```
1 sentence = "I live and work in Virginia"
2
3 print(sentence.upper())
4
5 print(sentence.lower())
6
7 print(sentence.swapcase())
8
9 print(sentence) # note that, source text is unchanged
10
```

```
1 I LIVE AND WORK IN VIRGINIA
2 i live and work in virginia
3 i LIVE AND WORK IN vIRGINIA
4 I live and work in Virginia
5
```

▶ Changing a String



Note that,

All these methods return `str` type.
So we can use the following syntax.



```
string.method1().method2().method3()...
```



Changing a String

- ▶ Let's grasp these methods through the examples 

```
1 sentence = "I live and work in Virginia"
2 title_sentence = sentence.title()
3 print(title_sentence)
4
5 changed_sentence = sentence.replace("i", "+")
6 print(changed_sentence)
7
8 print(sentence) # note that, again source text is unchanged
9
```

What is the output? Try to figure out in your mind...



Changing a String

- ▶ Let's grasp these methods through the examples 

```
1 sentence = "I live and work in Virginia"
2 title_sentence = sentence.title()
3 print(title_sentence)
4
5 changed_sentence = sentence.replace("i", "+")
6 print(changed_sentence)
7
8 print(sentence) # note that, again source text is unchanged
9
```

```
1 I Live And Work In Virginia
2 I l+ve and work +n V+rg+n+a
3 I live and work in Virginia
4
```



Changing a String

► Task

- Let's change the **first letters** of **each words** to **uppercase** of the following text 

```
text = 'the better the family, the better the society'
```



▶ Changing a String

- ▶ **The code may look like** 🙋

```
text = text.title()  
print(text)
```

The Better The Family, The Better The Society



Changing a String

- ▶ Let's review the pre-class examples 🙌

```
1 sentence = "I live and work in Virginia"
2 swap_case = sentence.swapcase()
3 print(swap_case)
4 print(swap_case.capitalize()) # changes 'i' to uppercase and
5 # the rest to lowercase
6
```

What is the output? Try to figure out in your mind...



Changing a String

- ▶ Let's review the pre-class examples 🙌

```
1 sentence = "I live and work in Virginia"
2 swap_case = sentence.swapcase()
3 print(swap_case)
4 print(swap_case.capitalize()) # changes 'i' to uppercase and
5 # the rest to lowercase
6
```

```
1 i LIVE AND WORK IN vIRGINIA
2 I live and work in virginia
3
```




Changing a String

► Task 🖐️

- In the `text` below, accidentally the number 0(zero) is used instead of the letter 'o' (oh). Fix them using `.replace()` method and **change** the **value of the variable** considering the new text and print the result.

```
text = 'S0d0me and G0m0re'
```



Changing a String

- ▶ The code may look like 🙋

```
text = 'S0d0me and G0m0re'  
text = text.replace('0', 'o')  
print(text)
```

Sodome and Gomore



4 Editing a String



▶ Editing a String

- ▶ The editing a string methods remove the trailing characters (i.e. characters from the right side).
- ▶ The default for the argument chars is also whitespace. If the argument chars aren't specified, trailing whitespaces are removed.
- ▶ The formula syntax 🖱️

```
string.method(arguments)
```



▶ Editing a String

- ▶ The summary of some common and the most important methods are as follows 🙌
 - `string.strip()` : removes all spaces (or specified characters) from both sides.
 - `string.rstrip()` : removes spaces (or specified characters) from the right side.
 - `string.lstrip()` : removes spaces (or specified characters) from the left side.



Editing a String

`string.strip(arg)`



- ▶ Let's review the pre-class examples 

```
1 space_string = "    listen first    "  
2 print(space_string.strip()) # removes all spaces from both sides  
3
```



Editing a String

`string.strip(arg)`



- ▶ Let's review the pre-class examples 

```
1 space_string = "    listen first    "  
2 print(space_string.strip()) # removes all spaces from both sides  
3
```

```
1 listen first  
2
```



Editing a String

`string.strip(arg)`



- ▶ Let's review the pre-class examples 🙌

```
1 space_string = "    listen first    "  
2 print(space_string.strip()) # removes all spaces from both sides  
3
```

```
1 listen first  
2
```

```
1 source_string = "interoperability"  
2 print(source_string.strip("yi"))  
3 # removes trailing "y" or "i" or "yi" or "iy" from both sides  
4
```




Editing a String

`string.strip(arg)`



- ▶ Let's review the pre-class examples 🙋

```
1 space_string = "    listen first    "  
2 print(space_string.strip()) # removes all spaces from both sides  
3
```

```
1 listen first  
2
```

```
1 source_string = "interoperability"  
2 print(source_string.strip("yi"))  
3 # removes trailing "y" or "i" or "yi" or "iy" from both sides  
4
```

```
1 nteroperabilit  
2
```



Editing a String

- ▶ Let's review the pre-class examples 

```
string.lstrip(arg)  
string.rstrip(arg)
```



```
1 source_string = "interoperability"  
2 print(source_string.lstrip("in"))  
3 # removes "i" or "n" or "in" or "ni" from the left side  
4
```

What is the output? Try to figure out in your mind...



Students, write your response!

REINVENT YOURSELF



Editing a String

- ▶ Let's review the pre-class examples 

```
string.lstrip(arg)  
string.rstrip(arg)
```



```
1 source_string = "interoperability"  
2 print(source_string.lstrip("in"))  
3 # removes "i" or "n" or "in" or "ni" from the left side  
4
```

```
1 interoperability  
2
```

Editing a String

- Let's review the pre-class examples

```
string.lstrip(arg)
string.rstrip(arg)
```

```
1 source_string = "interoperability"
2 print(source_string.lstrip("in"))
3 # removes "i" or "n" or "in" or "ni" from the left side
4
```

```
1 interoperability
2
```

```
1 space_string = "    listen first    "
2 print(space_string.rstrip()) # removes spaces from the right side
3
```

What is the output? Try to figure out in your mind...





Editing a String

```
string.lstrip(arg)  
string.rstrip(arg)
```



- Let's review the pre-class examples 🙋

```
1 source_string = "interoperability"  
2 print(source_string.lstrip("in"))  
3 # removes "i" or "n" or "in" or "ni" from the left side  
4
```


```
1 interoperability  
2
```

```
1 space_string = "    listen first    "  
2 print(space_string.rstrip()) # removes spaces from the right side  
3
```

```
1    listen first  
2
```

Editing a String

- Let's review the pre-class examples 



```
string.lstrip(arg)
string.rstrip(arg)
```

```
1 source_string = "interoperability"
2 print(source_string.rstrip("yt"))
3 # removes "y" or "t" or "yt" or "ty" from the right side
4
```

What is the output? Try to figure out in your mind...

Editing a String

- Let's review the pre-class examples 🙋

`string.lstrip(arg)`
`string.rstrip(arg)`

```
1 source_string = "interoperability"  
2 print(source_string.rstrip("yt"))  
3 # removes "y" or "t" or "yt" or "ty" from the right side  
4
```

```
1 interoperabili  
2
```

Editing a String

- Let's review the pre-class examples 

```
string.lstrip(arg)  
string.rstrip(arg)
```

```
1 source_string = "interoperability"  
2 print(source_string.rstrip("yt"))  
3 # removes "y" or "t" or "yt" or "ty" from the right side  
4
```

```
1 interoperabili  
2
```

```
1 source_string = "interoperability"  
2 print(source_string.rstrip("ty"))  
3
```

What is the output? Try to figure out in your mind...



Students, write your response!

Editing a String

`string.lstrip(arg)`
`string.rstrip(arg)`

- Let's review the pre-class examples 🙋

```
1 source_string = "interoperability"
2 print(source_string.rstrip("yt"))
3 # removes "y" or "t" or "yt" or "ty" from the right side
4
```

```
1 interoperabili
2
```

Either `ty` or `yt` works

```
1 source_string = "interoperability"
2 print(source_string.rstrip("ty"))
3
```

```
1 interoperabili
2
```



Editing a String

► Task 🖱️

- In the `text` below, accidentally there are additional letters. Remove the additional letters and make the all words uppercase.
- Except for the `print()` line, your code should consist of a single line.

```
text = 'tyou can learn almost everything in pre-classz'
```

desired output

```
YOU CAN LEARN ALMOST EVERYTHING IN PRE-CLASS
```



▶ Editing a String

- ▶ The code may look like 🙋

```
text = text.rstrip('z').lstrip('t').upper()  
print(text)
```

YOU CAN LEARN ALMOST EVERYTHING IN PRE-CLASS



Editing a String

► Task

- ▷ In the `text` below, accidentally the number the word “**we**” is wrong written. Remove the additional “**e**” letter and print the correct text.
- ▷ You can also use *indexing/slicing/methods* of string.

```
text = 'In God wee Trust'
```



Editing a String

- ▶ The code options might be like : 

```
1 text = 'In God wee Trust'
2
3 print(text.replace("ee", "e"))
4
5 text1 = text[:9]
6 text2 = text[10:]
7 print(text1 + text2)
8
9
10
11
```

Output

```
In God we Trust
In God we Trust
```



More String Methods

- ▶ Do not forget to follow for much:
- ▶ <https://docs.python.org/3/library/>

str.removeprefix(prefix, /)

If the string starts with the *prefix* string, return `string[len(prefix):]`. Otherwise, return a copy of the original string:

```
>>> 'TestHook'.removeprefix('Test')
'Hook'
>>> 'BaseTestCase'.removeprefix('Test')
'BaseTestCase'
```

New in version 3.9.

str.removesuffix(suffix, /)

If the string ends with the *suffix* string and that *suffix* is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string:

```
>>> 'MiscTests'.removesuffix('Tests')
'Misc'
>>> 'TmpDirMixin'.removesuffix('Tests')
'TmpDirMixin'
```



New in version 3.9.



THANKS!

End of the Lesson

(Main String Operations)

next Lesson

Lists

click above

