

Strings [immutable, iterable]

```
string = " hello world"                "hello world"
```

```
string = " hello world".capitalize()   "Hello world"
```

```
string = " Hello World".lower()        "hello world"
```

```
string = " Hello World".upper()        "HELLO WORLD"
```

```
string = " hello world".title()        "Hello Word"
```

```
string = " Hello wOrld".swapcase()     "hELLO WoRLD"
```

```
string = " ß".casefold()               "ss"
```

`.casefold()` is a more aggressive `.lower()` used for string comparisons.

```
int = " abaaba ababa".count(" aba ")   3
```

"abaaba" is counted to have 1 "aba"

```
int = " aca dac ad".find("c ad")       1
```

```
int = " aca dac ad".rfind(" cad ")     5
```

`string.find(value, start, end)` finds the *first* occurrence of the *value* after *start* before *end*.

`str.find(str2) == str.index(str2)`, except for when *str2* isn't found `.find()` returns 0 while `.index()` throws an error

```
string = "ab ab aa ab ab".replace("a b", "x x aa x ab", 3)
```

`str.replace(oldvalue, newvalue, count)` replaces *count* of *oldvalues* in the *string* with *newvalues*. Default *count* is all.

`str.format() == f'str'`

```
bool = " _,.__b an_ ana __..".strip("_,." - "ban_ana")
```

```
bool = " _,.__b an_ ana __..".lstrip("_,.") "ban_ana__.."
```

```
bool = " _,.__b an_ ana __..".rstrip("_,.") "ban_ana__.."
```

```
string = " hel lo".center(10, '.') ".hello.."
```

`str.center(int, char)` returns a string of *int* characters, with *string* in the middle and *chars* on the front and the end.

```
bool = " ab".isalpha()                 True
```

```
bool = " a1".isalnum()                 True
```

```
bool = " -12 ".isnumeric()             False
```

```
bool = " 1.2 ".isdigit()               False
```

```
bool = " aba ".endswith("ba ")         True
```

```
bool = " aba ".startswith(" ab")       True
```

```
bool = " a12 ".islower()               True
```

```
bool = " A12 ".isupper()               True
```

```
list = "Master Lordern".split("er")    ['Mast', ' Lord', 'n']
```

```
string = " -10.2".zfill(7)             "-0010.2"
```

`.zfill()` fills the beginning of the string with zeroes until the specified amount of characters are taken up, -, + and . included. Unlike `.rjust()`, `.zfill()` puts 0s after - and + signs.

```
string = " hello world".title()        "Hello Word"
```



By **HiBe** (HiBe)
cheatography.com/hibe/

Published 19th November, 2023.
Last updated 19th November, 2023.
Page 1 of 3.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish
Yours!
<https://apollopad.com>

Strings [immutable, iterable] (cont)

<code>string = " Hello wOrld".swapcase()</code>	"hELLO WoRLD"
<code>string = " hello world"[2:7]</code> <i>string slicing</i>	"llo w"

All string methods return a new string without mutating the original string

Lists [mutable, iterable]

<code>list = []</code>	[]
<code>list = list()</code>	[]
<code>list = [1, 2, 3]</code>	[1, 2, 3]
<code>list.append(4)</code>	[1, 2, 3, 4]
<code>list.insert(1, 8)</code>	[1, 8, 2, 3, 4]
<code>list.insert(ind, elem)</code> inserts <i>elem</i> at position <i>ind</i>	
<code>list.extend([8, 9, 1, 4, 3])</code>	[1, 8, 2, 3, 4, 8, 9, 1, 4, 3]
<code>list.extend(<iterable>)</code> extends the list by the elements of the <i>iterable</i>	
<code>list.pop()</code>	[1, 8, 2, 3, 4, 8, 9, 1, 3]
<code>list.pop(0)</code>	[8, 2, 3, 4, 8, 9, 1, 3]
<code>list.pop(ind)</code> pops the element at position <i>ind</i>	
<code>list.remove(8)</code>	[2, 3, 4, 8, 9, 1, 3]
<code>list.remove(elem)</code> removes the <i>first</i> instance of <i>elem</i>	
<code>list.index(3)</code>	1
<code>list.index(elem)</code> returns the index of the <i>first elem</i>	
<code>list.index()</code> <i>does not mutate</i> the original list	
<code>list.count(3)</code>	2
<code>list.count(elem)</code> returns the count of elements with value <i>elem</i>	
<code>list.count()</code> <i>does not mutate</i> the original list	
<code>list.sort()</code>	[1, 2, 3, 3, 4, 8, 9]
<code>list.sort()</code> cannot sort <i>int</i> against <i>str</i>	
<code>list.reverse()</code>	[9, 8, 4, 3, 3, 2, 1]
<i>list slicing:</i> <code>list2 = list[1:5]</code>	[8, 4, 3, 3]
<code>list2 = list.copy()</code>	list2 is [9, 8, 4, 3, 3, 2, 1]
<i>list slicing</i> or <code>list.copy()</code> <i>do not mutate</i> the original list, and create <i>shallow copies</i> .	
<code>list.clear()</code>	[]

Tuples [immutable, iterable]

<code>tuple = ()</code>	
<code>int = ("a", " a", " ab").count('a')</code>	2
<code>int = ("a", " a", " ab").index('a')</code>	0
<code>.index()</code> returns the position of the first instance	



Dictionaries [mutable, iterable]

<code>dictionary = {}</code>	<code>{}</code>
<code>dictionary = dict()</code>	<code>{}</code>
<code>dictionary = {key1: value1, key2: value2}</code>	<code>{key1: value1, key2: value2}</code>
<code>dictionary["str1"] = "str2"</code>	<code>{key1: value1, key2: value2, 'str1': 'str2'}</code>
<code>dictionary.pop('str1')</code>	<code>{key1: value1, key2: value2}</code>
<code>dictionary.popitem()</code>	<code>{key1: value1}</code>
<code>.popitem()</code> removes the last added element	
<code>dictionary2 = {key2: value2}</code>	
<code>dictionary.update(dictionary2)</code>	<code>{key1: value1, key2: value2}</code>
<code>.update(dict)</code> updates the dictionary with the key:value pairs of dict . In case of a conflict, dict is prioritized.	
<code>list = dictionary.keys()</code>	<code>[key1, key2]</code>
<code>list = dictionary.values()</code>	<code>[value1, value2]</code>
<code>list = dictionary.items()</code>	<code>[(key1, value1), (key2, value2)]</code>
<code>.items()</code> returns a list of tuples	
<code>something = dictionary.get(keyname, value)</code>	
if keyname in a dictionary does not exist as a key, value is returned.	
<code>dictionary2 = dictionary.copy()</code>	<code>{key1: value1, key2: value2}</code>
<code>dictionary.clear()</code>	<code>{}</code>

Functions

```
def function(parameter1, parameter2 = default value):
    <something>
    return value
function(argument1, argument2)
```

