Строка str (string = текст)

Основы синтакса

В питоне текст может обозначаться как двойными кавычками `"`, так и одиночными `'`.

```
some_text = "Text"
print(type(text))

>> <class 'str'>

some_text = 'Text'
print(type(text))

>> <class 'str'>
```

Тексты можно сложить друг с другом при помощи оператора сложения `+`.

```
text1 = 'Hello'
text2 = 'User'

total_text = text1 + text2
print(total_text)

>> HelloUser

total_text2 = text1 + ', ' + text2
print(total_text2_

>> Hello, User
```

Выдается ошибка `TypeError`, если при сложении тип одного из значений не является `str`.

```
total_text = 'Hello' + 2
```

```
>> Traceback (most recent call last):
    File "<pyshell#1>", line 1, in <module>
     "2"+3
TypeError: can only concatenate str (not "int") to str

total_text = 'Hello' + str(2)

>> Hello2
```

Текст можно умножить на целое число (`int`), и тем самым скопировать текст несколько раз.

```
text = 'Hello'
new_text = text*4
print(new_text)

>> HelloHelloHelloHello
```

Умножив на `float`, выдается ошибка `TypeError`.

```
text = 'Hello'
new_text = text * 3.2

>> Traceback (most recent call last):
    File "<pyshell#5>", line 2, in <module>
     new_text = text*3.2
TypeError: cannot multiply sequence by non-int of type 'float'
```

Форматирование текста

Существует несколько способа для форматирования текста в Питоне. Текст форматируется по закругленным скобкам `{}`.

Предположим программу, которая запрашивает у пользователя 2 числа, и выволит результат терминал в форме `Первое число это X, а это - число У`. Эту программу можно написать следующими способами.

str.format(*args, **kwargs)

```
# Рандомные числа для примера
num1 = 23
num2 = 36.789

message = 'Первое число это {}, а это - число {}'.format(num1, num2)
print(message)

>> Первое число это 23, а это - число 36.789
```

В этом случае, по умолчанию, значения будут форматироваться по порядку аргументов. Порядок модно задать при помощи индексирования (начаиная с `0`).

```
# Рандомные числа для примера
num1 = 23
num2 = 36.789

message = 'Первое число это {1}, а это - число {0}'.format(num1, num2)
print(message)

>> Первое число это 36.789, а это - число 23
```

Также, в нутри скрученных скобок можно внести временную переменную. Значение этой переменной можно задать в качестве аргумена функции `format`.

```
# Рандомные числа для примера
num1 = 23
num2 = 36.789

message = 'Первое число это {second}, а это - число {first}'.format(first=num1, second=num2)
print(message)

>> Первое число это 36.789, а это - число 23
```

---

f-string

Поставив букву `f` перед кавычками, можно ввести название переменной напрямую в закрученные скобки.

```
# Рандомные числа для примера
num1 = 23
num2 = 36.789

message = f'Первое число это {num1}, а это - число {num2}'
print(message)
```

Полезные команды

str.count(*sub[, start[, end]]*)

Выдает количество появлений подтекста `sub`. Выборочно можно задать начальное `start` или конечное `end` место в тексте. (`end` не включительно).

- ***sub*: *str*** → Подтекст, который ищется
- ***start*: *int*** → Начальный индекс промежутка для поиска
- ***end*: *int*** → Конечный индекс промежутка для поиска (не включительно)

```
string = 'Aaaabbbcc'
print(string.count('a'))

>> 3
```

str.find(*sub[, start[, end]]*)

Выдает первый индекс, где находиться подтекст `sub`. Выборочно можно задать начальное `start` или конечное `end` место в тексте. (`end` не включительно). Если подтекст `sub` не найден в `str`, функция выдает `-1`

- ***sub*: *str*** → Подтекст, который ищется
- ***start*: *int*** → Начальный индекс промежутка для поиска

- ***end*: *int*** → Конечный индекс промежутка для поиска (не включительно)

```
string = 'Parallel'
print(string.find('a'))

>> 1

print(string.find('b'))

>> -1
```

---

str.index(*sub[, start[, end]]*)

Выдает первый индекс, где находиться подтекст `sub`. Выборочно можно задать начальное `start` или конечное `end` место в тексте. (`end` не включительно). Если подтекст `sub` не найден в `str`, функция выдает ошибку: `ValueError: substring not found` .

- ***sub*: *str*** → Подтекст, который ищется
- ***start*: *int*** → Начальный индекс промежутка для поиска
- ***end*: *int*** → Конечный индекс промежутка для поиска (не включительно)

```
string = 'Parallel'
print(string.find('l'))

>> 4

print(string.find('b'))

>> Traceback (most recent call last):
>>   File "<stdin>", line 5, in <module>
>>   ValueError: substring not found
```

str.startswith(*sub[, start[, end]]*)

Выдает `True` если текст `str` начинается на подтекст `sub`. В противном случае выдается `False`. `sub` может быть как типа

`str`, так и типа `tuple` подтекстов (в этом случае, если текст начинается хоть на один из подттекстов, будет выдан `True`). Выборочно можно задать начальное `start` или конечное `end` место в тексте. (`end` не включительно).

- ***sub*: *[str, tuple]*** → Подтекст, который ищется
- ***start*: *int*** → Начальный индекс промежутка для поиска
- ***end*: *int*** → Конечный индекс промежутка для поиска (не включительно)

```
string = 'Start end'
print(string.startswith('Start))

>> True

print(string.startswith(('Start', 'end')))

>> True
```

---

str.startswith(*sub[, start[, end]]*)

Выдает `True` если текст `str` заканчивается на подтекст `sub`. В противном случае выдается `False`. `sub` может быть как типа `str`, так и типа `tuple` подтекстов (в этом случае, если текст заканчивается хоть на один из подттекстов, будет выдан `True`). Выборочно можно задать начальное `start` или конечное `end` место в тексте. (`end` не включительно).

- ***sub*: *[str, tuple]*** → Подтекст, который ищется
- ***start*: *int*** → Начальный индекс промежутка для поиска
- ***end*: *int*** → Конечный индекс промежутка для поиска (не включительно)

```
string = 'Start end'
print(string.endswith('end'))

>> True

print(string.endswith('Start')
```

```
>> False
```

str.removepreffix(*prefix*)

Если текст начинается на `prefix`, то выдается текст `[len(prefix):]`. В противном случае выдается оригинальный текст.

- ***prefix*: *str*** → Пристака которую требуется убрать из текста

```
string = 'preto remove'
print(string.removepreffix('pre'))

>> 'to remove'
```

str.removesuffix(*suffix*)

Если текст заканчивается на `prefix`, то выдается текст `[len(prefix):]`. В противном случае выдается оригинальный текст.

- ***suffix*: *str*** → Пристака которую требуется убрать из текста

```
string = 'to removesuf'
print(string.removesuffix())

>> 'to remove'
```

str.capitalize()

Выдает копию текста с заглавной первой буквой, и с маленькими оставнимися буквами.

```
string = 'hey Guys'
print(string.capitalize())

>> 'Hey guys'
```

---

str.lower()

Выдает копию текста со всеми маленькими буквами.

```
string = 'ALL UPPER LETTERS'
print(string.lower())

>> 'all upper letters'
```

---

str.upper()

Выдает копию текста со всеми заглавными буквами.

```
string = 'all lower letters'
print(string.upper())

>> 'ALL LOWER LETTERS'
```

---

str.title()

Выдает копию текста, где первая буква в каждом слове заглавная, а все остальные маленькие.

```
string = 'This is the title'
print(string.capitalize())

>> 'This Is The Title'
```

str.swapcase()

Выдает копию текста, где каждая заглавная буква сменяется на маленькую, а маленькая - на заглавную.

```
string = 'aLL CaSEs ArE weIRd'
print(string.swapcase())

>> 'All cAseS aRe WEirD'
```

str.split(*sep=None, maxsplit=-1*)

Выдает список слов в тексте разделенные при помощи `sep`. Если дан `maxsplit`, то текст будет разделен максимум `maxsplit` количество раз (количество элементов будет `maxsplit`+1). Если `maxsplit` не дан или `-1`, то текст будет разделен максимально возможное количество раз. Если `sep` не дан или `None`, то текст будет разделен по пробелу или пустому месту

- ***sep*: str → Разделитель. Символ или слово по которому разделяется тескс. По умолчанию `None`**
- ***maxsplit*: int → Указывает сколько раз разделять текст. По умолчанию `-1`**

```
string = 'Ayaz,Imad,Seymur'
print(string.split(','))

>> ['Ayaz', 'Imad', 'Seymur']
# s maxsplit
string = 'Ayaz,Imad,Seymur'
print(string.split(',', maxsplit=1)

>> ['Ayaz', 'Imad,Seymur']
# bez argumentov
string = '          Ayaz          Imad Seymur          '
print(string.split())
```

```
>> ['Ayaz', 'Imad', 'Seymur']
```

---

str.splitlines()

Выдает список слов разделенные по “newline” `\n`.

```
string = 'Ayaz\nImad\nSeymur'
print(string)
print('==============')
print(string.splitlines())

>> Ayaz
>> Imad
>> Seymur
>> ==============
>> ['Ayaz', 'Imad', 'Seymur']
```

---

str.replace(*old, new[, count]*)

Выдает копию текста, где все подтексты `old` заменены на `new`. Если дан аргумент `count`, то будут заменены только первые `count` количество подтекстов

- ***old*: str** → Подтекст, который нужно заменить
- ***new*: str** → Подтекст, на который нужно заменить
- ***count*: int** → Количество подтекстов, которое нужно заменить

```
string = 'Imad gandon'
print(string.replace('gandon', 'got')

>> Imad got
# s count
string = 'Imad gandon gandon i gandon'
print(string.replace('gandon', 'got', 2))

>> Imad got got i gandon
```

str.strip(*[chars]*)

Возвращает копию текста с удаленными начальными и конечными символами. Аргумент `chars` представляет собой строку, определяющую набор удаляемых символов. Если он не дан или `None`, аргумент `chars` по умолчанию удаляет пробелы. Аргумент `chars` не является префиксом или суффиксом; все комбинации его значений удаляются.

- ***chars*: *str*** → Коллекция символ на удаление

```
string = '           Some text                    '
print(string.strip())

>> Some text
# s argumentami
string = 'www.example.com'
print(string.strip('cmowz.'))

>> example
```

str.isalpha()

Выдается `True`, если все символы в тесте это буквы. В противном случае выдается `False`.

```
# Bez probela
string = 'sometext'
print(string.isalpha())

>> True
# S probelom
string = 'some text'
print(string.isalpha())

>> False
```

str.isnumeric()

Выдается `True`, если текст это число. В противном случае выдается `False`.

```
# Celoye chislo
string = '24567'
print(string.isnumeric())

>> True
# Drobnoye chislo
string = '34.67'
print(string.isnumeric())

>> False
```

str.isalnum()

Выдается `True`, если текст это число или буквы. В противном случае выдается `False`.

```
# Bez tochki
string  = '34okofe'
print(string.isalnum())

>> True

# S tochkoy
string = '345.34oflee'
print(string.isalnum())

>> False
```

str.isspace()

Выдается `True`, если текст это пробел. В противном случае выдается `False`.

```
string = '              '
print(string.isspace())

>> True

string = '\n\n\n\n\n'
print(string.isspace())

>> True
```

---

str.islower()

Выдает `True`, если все буквы в тексте маленькие. В противном случае выдается `False`.

```
string = 'all lower'
print(string.islower())

>> True

string = 'All lower'
print(string.islower())

>> False
```

---

str.isupper()

Выдает `True`, если все буквы в тексте заглавные. В противном случае выдается `False`.

```
string = 'ALL UPPER'
print(string.isupper())

>> True
```

```
string ='ALL LOwER'
print(string.isupper())

>> False
```

---

str.istitle()

Выдает `True`, если первая буква в каждом слове заглавная. В противном случае выдается `False`.

```
string = 'This Is The Title'
print(string.istitle())

>> True

string = 'This is the title'
print(string.istitle())

>> False
```