Строки в Python3

Строки в Python3

Литералы строк

```
# Любые кавычки,
# лишь бы закрывающая была такой же
'hello'
"world"

s = "I'm user"
s = '"hello"'
```

Multiline strings :)

```
"hello
world"  # ошибка синтаксиса, строка должны быть в одной строке
         # целиком

"""hello
world"""
# всё хорошо, это строка, которую можно разносить на много строк.
# Такие строки часто используются для документации.

def square(x):
    """
    Функция возводит x в квадрат.
    """
    return x * x
```

Ввод строки и преобразования в строку

```
s = str(3)
s = str(3.14)
s = input()  # ввод строки без '\n' на конце
```

Строки и символы

Отдельного типа “символ” в Python3 нет. Символ – это строка из одного элемента.

```
s = 'abc'
print(type(s))  # str
print(len(s)  # 3
c = s[0]
print(c)   # a
print(type(c))  # str
print(len(c))  # 1
```

Функция len вычисляет длину последовательности, в том числе строки.

Спецсимволы

\0 в Python не является концом строки

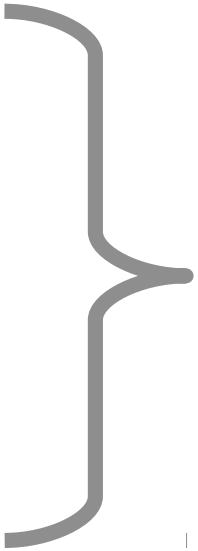
\n – перевод строки

\r – возврат каретки

\t – табуляция

\v – вертикальная табуляция

пробел



пробельные символы

\' - просто ' (не закрывает строку)

\” - просто " (не закрывает строку)

\\ - просто \

```
path = 'C:\\Documents\\Vasya'

# ещё один вид строкового литерала
# без обработки спецсимволов
path = r'C:\Documents\Vasya'
```

Оператор in

s1 in s2

True, если в s2 есть подстрока s1

False иначе

'yt' in 'Python' => True

'ty' in 'Python' => False

Доступ по индексу

s = 'Python'

s[0] => 'P'

s[1] => 'y'

s[4] => 'o'

s[5] => 'n'

s[-1] => 'n'

s[-2] => 'o'

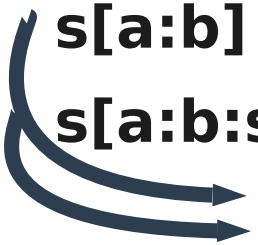
s[-6] => 'P'

s[len(s) - 6] => 'P'

Срезы

Результат среза – это копия строки!

```
s = 'Python'
s[0:4]   =>  'Pyth'
s1 = s[0:4]  # В s1 лежат копии первых четырёх
# символов s
s[:]    – полная копия
s[::-1]   => 'nohtyP'
s[::2]    => 'Pto'    # по чётным индексам
s[1::2]    => 'yhn'     # по нечётным индексам
s[a:b] =>   символы от a до (b-1)
s[a:b:step]  => символы от a с шагом step не включая b
```



Как в range(a, b), range(a, b, step)

Срезы с отрицательными индексами

```
s[-4:2]   => 'th'
s[1:-2]    => 'oth'
s[-5:5]    => 'ytho'
s[-2:-4:-1]    => 'oh'
```

Строки в Python3 неизменяемы

```
s = 'Python'
s[2] = 'T'  # Ошибка! Строки изменять нельзя!
s = s[0:2] + 'T' + s[3:]
# Операция + выделяет память под новую
# строку и копирует в неё операнды один за
# другим.
s   => 'PyThon'
```

Методы строк

Так как строки не изменяемы, методы не изменяют “свою” строку, а создают новую.

```
s = '  Hello,  \n world!\n'
s.strip()    => 'Hello,  \nworld!'
```

метод strip убирает из начала и конца строки пробельные символы

так как создана новая строка, чтобы сохранить результат нужно сделать

```
s1 = s.strip()
s.lstrip()    => 'Hello,  \n world!\n'
s.rstrip()    => '  Hello,  \n world!'
```

Метод split

```
s = '_abc__\n_def__xyz'
s.split()   => ['abc', 'def', 'xyz']
# split разбивает строку по пробельным символам
# на слова, вернёт список строк
s = 'ab,_cd,_ef,_'
s.split(',_')  => ['ab', 'cd', 'ef']
# Если у split есть аргумент, то в результате могут быть пустые строки.
```

Циклы по строкам

```
# Цикл по индексам:
for i in range(len(s)):
    print(s[i])

# Цикл по символам:
for c in s:
    print(c)
```

```
# Пример
for c in 'hello':
    print(c)

выведет:
h
e
l
l
o
```