Строки. Шпаргалка

Часть 1

Строки представляют собой неизменяемую последовательность символов и могут использоваться для хранения текстовой информации.

Какие кавычки используются в строках

Если строка не содержит переносов (записана в одну строчку), используют двойные или одинарные кавычки:

```
single_quoted = 'Это строка в одинарных кавычках.'
double_quoted = "Это строка в двойных кавычках."
```

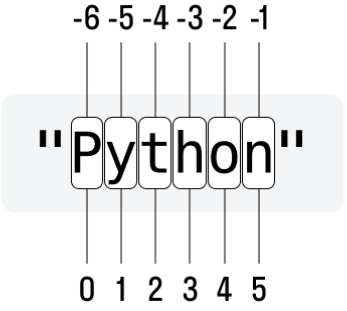
Если строка переносится на разные строчки, используют тройные двойные или тройные одинарные кавычки:

```
multi_line = '''Это многострочная
строка, заключенная в тройные одинарные кавычки.'''

docstring = """Это пример документационной строки.
Она может быть использована для описания функций или модулей."""
```

Индексы строк

По аналогии со списками каждый символ в строке имеет свой уникальный индекс, который начинается с `0`. Индексы могут быть отрицательными при подсчете с конца.



Получение первого символа строки по индексу:

```
text = "Hello, World!"
first_char = text[0]
print(first_char)
>>> H
```

Получение последнего символа с помощью отрицательного индекса:

```
text = "Hello, World!"
last_char = text[-1]
print(last_char)
>>> !
```

Получение символа по индексу 7:

```
text = "Hello, World!"
char_at_index_7 = text[7]
print(char_at_index_7)
>>> W
```

Получение подстроки с 0-го по 5-й индекс:

```
text = "Python is awesome"
substring = text[0:6]
print(substring)
>>> 'Python'
```

Получение подстроки с 7-го по 8-й индекс:

```
text = "Python is awesome"
substring = text[7:9]
print(substring)
>>> 'is'
```

Получение подстроки с 10-го индекса до конца строки:

```
text = "Python is awesome"
substring = text[10:]
print(substring)
>>> 'awesome'
```

Получение подстроки с начала до 4-го индекса:

```
text = "Python is awesome"
substring = text[:5]
print(substring)
>>> 'Pytho'
```

Получение подстроки с отрицательным начальным индексом (отсчет с конца строки):

```
text = "Python is awesome"
substring = text[-7:]
print(substring)
>>> 'awesome'
```

Получение подстроки с отрицательным конечным индексом (с начала строки до 7-го символа с конца):

```
text = "Python is awesome"
substring = text[:-7]
print(substring)
>>> 'Python is'
```

Получение строки в обратном порядке с использованием отрицательного шага:

```
text = "Python is awesome"
reversed_text = text[::-1]
print(reversed_text)
>>> 'emosewa si nohtyP'
```

Получение всех четных цифр с использованием среза с шагом:

```
text = "1234567890"

even_digits = text[1::2]
print(even_digits)
>>> '24680'
```

Получение всех нечетных цифр с использованием среза с шагом:

```
text = "1234567890"

odd_digits = text[::2]
print(odd_digits)
>>> '13579'
```

Итерация по строке с помощью цикла `for`:

```
text = "Hello, World!"

for char in text:
    print(char)
>>> H
    e
    l
    l
    o
    ,

    W
    o
    r
    l
    d
    !
```

Как определить длину строки (количество символов в строке):

```
text = "Python is awesome"

length = len(text)
print(length)
>>> 17
```

Итерация по строке с использованием индексов:

```
text = "Python"

for i in range(len(text)):
    print(f"Индекс {i}: {text[i]}")
>>> Индекс 0: P
    Индекс 1: y
    Индекс 2: t
    Индекс 3: h
    Индекс 4: o
    Индекс 5: n
```

Проверка вхождения подстроки в строку:

```
text = "Python is a powerful language"

if "power" in text:
    print("Подстрока 'power' найдена.")
else:
    print("Подстрока 'power' не найдена.")

>>> Подстрока 'power' найдена.
```

Проверка, что подстрока не входит в строку:

```
text = "Python is a powerful language"

if "Java" not in text:
    print("Подстрока 'Java' не найдена.")
else:
    print("Подстрока 'Java' найдена.")
>>> Подстрока 'Java' не найдена.
```

Проверка, на каком индексе в строке начинается подстрока:

```
text = "Python is a powerful language"
substring = "Java"

index = text.find(substring)
if index != -1:
    print(f"Подстрока '{substring}' найдена на позиции {index}.")
else:
    print(f"Подстрока '{substring}' не найдена.")

>>> Подстрока 'Java' не найдена.
```

Проверка, что строка состоит из букв или цифр:

```
text1 = "12345"
text2 = "Hello123"
text3 = "Python"
text4 = "42.0"

result1 = text1.isdigit()
result2 = text2.isdigit()
result3 = text3.isalpha()
result4 = text4.isdigit()

print(result1)
print(result2)
print(result3)
print(result4)

>>> True
    False
    True
    False
```

Часть 2

Конкатенация (сложение) строк:

```
word_1 = "counter"
word_2 = "clock"
word_3 = "wise"

result = word_1 + word_2 + word_3
print(result)

>>> "counterclockwise"
```

Сложение строк в цикле:

```
words = ["counter", "clock", "wise"]
result = ""

for word in words:
    result += word

print(result)

>>> "counterclockwise"
```

Фильтрация через конкатенацию строк:

```
elements = [1, 0, 1, 0, 2, 3, 1, 0]
result = ""

for el in elements:
    if el == 1 or el == 0:
          result += str(el)

print(result)

>>> "101010"
```

Как посчитать количество вхождений подстроки в строку:

```
text = "Python is a programming language. Python is easy to learn."
count_python = text.count("Python")
print(count_python)

>>> 2
```

Как преобразовать все символы строки в нижний регистр:

```
text = "Hello World"
lower_text = text.lower()
print(lower_text)

>>> "hello world"
```

Как преобразовать все символы строки в верхний регистр:

```
text = "Hello World"
upper_text = text.upper()
print(upper_text)

>>> "HELLO WORLD"
```

Как преобразовать строку, чтобы каждое слово начиналось с большой буквы, а остальные буквы были в нижнем регистре:

```
text = "python programming is fun"
formatted_text = text.title()
print(formatted_text)

>>> "Python Programming Is Fun"
```

Как заменить все вхождения одной подстроки другой подстрокой:

```
text = "I like apples, but I also like bananas."
new_text = text.replace("apples", "oranges")
print(new_text)

>>> "I like oranges, but I also like bananas."
```

Как удалить лишние символы из строки:

```
text = "Pyt?hon pro?gramm?ing is fun!"

formatted_text = text.replace("?", "")
print(formatted_text)

>>> "Python programming is fun!"
```

Как разделить строку в список, используя разделитель

Без указания конкретного разделителя строка будет разделена по пробелам:

```
text = "Это пример использования метода split"
words = text.split()
print(words)

>>> ['Это', 'пример', 'использования', 'метода', 'split']
```

Указываем разделитель для управления формированием списка. Разделитель передаем в скобках методу `split()`:

```
csv = "apple, banana, cherry, date"
fruits = csv.split(", ")
print(fruits)

>>> ['apple', 'banana', 'cherry', 'date']
```

Как ограничить количество разбиений строки:

```
text = "раз, раз, раз, раз, раз"
splits = text.split(", ", 3)
print(splits)

>>> ['раз', 'раз', 'раз', 'раз, раз']
```

Как объединить элементы списка в строку:

```
words = ["Python", "is", "awesome"]
text = " ".join(words)
print(text)

>>> "Python is awesome"
```

Классификация методов

Методы, которые возвращают числа — длину, позицию символа или подстроки

Метод	Что делает
`count(sub)`	Возвращает количество непересекающихся вхождений подстроки `sub`
`find(sub)`	Возвращает индекс первого вхождения подстроки `sub`. При неудаче возвращает −1
`index(sub)`	Возвращает индекс первого вхождения подстроки `sub`. При неудаче вызывает исключение

Методы, которые создают новые строки

Метод	Что делает
`lower()`	Вернет новую строку в нижнем регистре
`upper()`	Вернет новую строку в верхнем регистре
`title()`	Вернет новую строку в нижнем регистре, но с первой прописной буквой
`replace(old, new)`	Вернет новую строку, где все вхождения `old` заменены на `new`
`delimiter.join(list)`	Вернет новую строку, где склеивает список строк `list` с помощью разделителя `delimiter`

Метод, который возвращает списки

Метод	Что делает
`split(delimiter)`	Возвращает список слов в строке, используя `delimiter` в качестве строки-разделителя

Методы, которые возвращают булево значение

Метод	Что делает
`islower()`	Возвращает **True**, если все символы в строке в нижнем регистре
`isupper()`	Возвращает **True**, если все символы в строке в верхнем регистре
`istitle()`	Возвращает **True**, если в строке есть хотя бы один символ и все слова в строке начинаются с заглавной буквы (слова — подстроки, разделенные пробелом)
`isspace()`	Возвращает **True**, если все символы в строке — пробелы

Метод	Что делает
`isprintable()`	Возвращает **True**, если в строке нет непечатных символов (например, \n)
`startswith(prefix)`	Возвращает **True**, если найдено вхождение подстроки `prefix` в начале строки
`endswith(suffix)`	Возвращает **True**, если найдено вхождение подстроки `suffix` в конце строки
`isalpha()`	Возвращает **True**, если строка не пустая и все символы в строке являются буквами
`isdigit()`	Возвращает **True**, если строка не пустая и все символы в строке являются цифрами