Python: строки | Я.Шпора

Строка — упорядоченная неизменяемая коллекция, которая хранит набор текстовых символов.

Как объявить строку

При объявлении строк их замыкают в одинарные, двойные, тройные одинарные или тройные двойные кавычки.

Особенности использования кавычек:

```
# В строках можно применять вложенные кавычки.
info_string = 'Когда вы едите сельдерей, вы "сжигаете" калории.'

# Если вложенные кавычки не отличаются от обрамляющих,
# строка превратится в несколько отдельных строк и возникнет ошибка.
info_string = 'Когда вы окучиваете грядку, вы 'сжигаете' калории.'

# А кавычки-ёлочки Python воспринимает как обычный текстовый символ:
# внутри строки их можно применять как угодно, но обрамлять ими строку не
info_string = 'Да вы вообще всегда «сжигаете» калории.'
```

Ещё один способ объявления строки — функция `str()`:

```
int_number = 100
# Преобразовать число в строку.
str_number = str(int_number)
print(str_number)
# Вывод в терминал: 100
print(type(str_number))
# Вывод в терминал: <class 'str'>
```

При объявлении длинной строки следует разбить её на несколько частей. В Python не принято делать строчки кода длиннее 79 символов.

Перенос строк выполняется по следующим правилам:

- вся строка целиком заключается в скобки;

- текст внутри скобок разбивается построчно, на фрагменты нужной длины;
- новую строку кода не начинают с пробела;
- каждая получившаяся строчка заключается в кавычки;
- для читаемости все строчки отбиваются четырьмя пробелами от начала строки кода.

```
info_string = (
    'Когда картофель '
    'был впервые привезён в Европу, '
    'его выращивали исключительно как '
    'декоративное растение.'
)
```

Индексы

Элементы строки — текстовые символы. У каждого элемента есть порядковый номер — индекс. Отсчёт начинается с 0.

```
message = 'Hello'
print(message[2])
# Вывод в терминал: l
```

К элементам последовательности можно обратиться, отсчитывая их с конца. Для этого применяют отрицательные индексы: последний элемент строки можно получить по индексу `[-1]`, предпоследний — по индексу `[-2]` и так далее.

```
lang_name = 'Python'
# P   y   t   h   o   n  (строка - набор символов).
# -6 -5  -4  -3  -2  -1  (значения отрицательных индексов для каждого с
имвола).

print(lang_name[-6] + lang_name[-2] + lang_name[-1] + lang_name[-5])

# Вывод в терминал: Pony
```

Если обратиться к индексу, который выходит за пределы строки, Python выдаст

ошибку `IndexError`.

Работа со строками

> Функции, изменяющие набор символов в строке, не изменяют исходную строку, а возвращают её изменённую копию.

Изменить регистр символов в строке

Чтобы вернуть копию строки, в которой все символы переведены в нижний или верхний регистр, используются операции `str.lower(<str>)` и `str.upper(<str>)`.

```
status_string = 'Урожай ВЕЛИКОЛЕПЕН'
print(str.lower(status_string))
# Вывод в терминал: урожай великолепен

alarm_string = 'шелковица поражена болезнью!'
print(str.upper(alarm_string))
# Вывод в терминал: ШЕЛКОВИЦА ПОРАЖЕНА БОЛЕЗНЬЮ!
```

Операция `str.capitalize(<str>)` возвращает копию строки, в которой первый символ переведён в верхний регистр, а остальные — в нижний.

```
harvest_string = 'сегодня ВЫ собрали 12 КГ огурцов'
print(str.capitalize(harvest_string))
# Вывод в терминал: Сегодня вы собрали 12 кг огурцов
```

Операция `str.title(<str>)` возвращает копию строки, в которой первый символ каждого слова переведён в верхний регистр, а остальные — в нижний.

```
gardener_name_string = 'садовод тоня'
print(str.title(gardener_name_string))
# Вывод в терминал: Садовод Тоня
```

Операция `str.swapcase()` возвращает копию строки, в которой регистр символов инвертирован.

```
inverted_string = 'зА СеЗоН СоБрАнО ОгУрЦоВ: 185'
```

```
print(str.swapcase(inverted_string))
# Вывод в терминал: За сЕзОн сОбРаНо оГуРцОв: 185
```

Поиск и замена подстрок

Операция `str.find(<str>, sub)` ищет подстроку `sub` в строке `<str>` и, если находит — возвращает индекс первого символа подстроки в исходной строке.

```
yield_weight = 'Сегодня вы собрали 500 граммов помидоров'
print(str.find(yield_weight, 'граммов'))
# Вывод в терминал: 23
```

Команда `str.replace(<str>, sub, new)` ищет подстроку `sub` в строке `<str>` и возвращает копию исходной строки, где подстрока `sub` заменена на подстроку `new`.

```
tomato_motto = 'Помидоры полезны. Помидоры красивые и весёлые.'

cucumber_motto = str.replace(tomato_motto, 'Помидоры', 'Огурцы')
print(cucumber_motto)
# Вывод в терминал: Огурцы полезны. Огурцы красивые и весёлые.

cucumber_motto = str.replace(tomato_motto, 'Помидоры', 'Огурцы', 1)
print(cucumber_motto)
# Вывод в терминал: Огурцы полезны. Помидоры красивые и весёлые.
```

Операция `str.strip(<str>, chars)` возвращает копию исходной строки, удаляя из её начала и конца символы, перечисленные во втором аргументе.

```
garden_message = ' !! == Помидоры созрели! Урожай великолепен! == !! '
text = str.strip(garden_message, '=! ')
print(text)
# Вывод в терминал: Помидоры созрели! Урожай великолепен
```

Разделение строки: str.split(<str>, delimeter)

Разбивает на части строку, переданную в аргументах, и возвращает список этих частей.

```
vegetables = 'Помидоры; Огурцы; Баклажаны; Перцы; Кабачки'
# Разделитель может состоять из нескольких символов.
print(str.split(vegetables, '; '))

# Вывод в терминал:: ['Помидоры', 'Огурцы', 'Баклажаны', 'Перцы', 'Каба
чки']
# Обратите внимание: комбинация "точка с запятой" удалена из списка.
```

Если второй аргумент не указан, строка будет разделена по пробелам.

Формирование строки: str.join(delimeter, <sequence>)

Преобразует последовательность строк в строку. В первом аргументе передаётся разделитель, который будет вставлен в сгенерированной строке между значениями элементов. Разделителем должна быть строка, но эта строка-разделитель может быть и пустой — `''`.

```
veggie_list = ['Помидоры', 'Огурцы', 'Баклажаны', 'Перцы', 'Кабачки']
# Разделитель - пробел.
print(str.join(' ', veggie_list))

# Разделитель - пробел, смайлик и пробел.
print(str.join(' ¯\_(ツ)_/¯ ', veggie_list))

# Вывод в терминал:
# Помидоры Огурцы Баклажаны Перцы Кабачки
# Помидоры ¯\_(ツ)_/¯ Огурцы ¯\_(ツ)_/¯ Баклажаны ¯\_(ツ)_/¯ Перцы ¯\_
(ツ)_/¯ Кабачки
```

Конкатенация

Операция объединения строк с переменными других типов с помощью оператора `+`.

```
tomatoes = 250
print('Вы собрали ' + str(tomatoes) + ' кг помидоров.')

# Вывод в терминал: Вы собрали 250 кг помидоров.
```

Функция `str.format(<str>, *vars)`

Позволяет подставлять в строку значения, перечисленные в `*vars` , — таких значений может быть сколько угодно. Значение вставляется на место фигурных скобок `{N}` в строке.

```
print(str.format('{0} имеют {1} {2}.', 'огурцы', 'неповторимый', 'вкус'))
# Вывод в терминал: огурцы имеют неповторимый вкус.
```

f-строки

f-строка — отформатированная строка. Чтобы её составить, нужно вставить имя переменной в фигурных скобках в f-строку — на это место подставится значение переменной:

```
tomatoes_yield = 10.564
fstring = f'За день вы собрали {tomatoes_yield} кг помидоров'
# Вывод в терминал: За день вы собрали 10.564 кг помидоров
```

Шпаргалка — мост, ведущий
мудрого над безднами забвения к
вершине знания.

Дге Тибетский