Базовые коллекции. Строки

Форматирование строк: format и f-strings

В Python много инструментов, позволяющих изменять строки. Один из них — метод **format**. С его помощью можно вставлять слова или значения в строку, например:

```python
username = input('Введите своё имя')  # Получаем имя пользователя.
print('Hello, {name}!'.format(name=username))  # Вставляем имя в строку при помощи метода format.
```

Здесь `{name}` — это параметр, который заменяется значением переменной `username`.

f-strings — ещё один способ форматирования строк. Он похож на `format`, но работает быстрее и проще. Для использования f-строк нужно поставить перед строкой букву `f`:

```python
username = input('Введите своё имя')
print(f'Hello, {username}!')
```

В этом случае мы не создаём промежуточный параметр `name`, а сразу указываем переменную `username` внутри фигурных скобок.

Форматирование чисел при подстановках в строки

Кроме простой подстановки, можно указать дополнительные параметры для чисел. Например, разделить большое число на разряды или указать количество знаков после запятой для вещественных чисел.

Примеры:

- Разделение большого числа на разряды:

```python
details_num = 500000
print("На складе осталось {:,d}".format(details_num))
# Результат: На складе осталось 500,000.
```

Разберём запись `{:,d}` подробнее:

`:` — объявляет подстановку;

`d` — означает, что мы работаем с целыми числами;

`,` — указывает методу, чем нужно разделять большое целое число.

- Форматирование вещественного числа:

```python
price = 23.594233
print("Каждая деталь стоит {:.2f}".format(price))
# Результат: Каждая деталь стоит 23.59.
```

Здесь в записи `{.2f}`:

`.2` — указывает на количество знаков после точки;

`f` — означает, что мы работаем с вещественными числами.

- Выражение числа в процентах:

```python
percent = 0.05
print("Перевод в проценты числа 0.05 будет выглядеть так
{:.1%}".format(percent))
# Результат: Перевод в проценты числа 0.05 будет выглядеть так 5.0%.
```

Здесь `.1%` означает, что мы показываем один знак после запятой и выражаем число в процентах.

- Экспоненциальная запись большого числа:

```python
details_num = 500000
print("Большие числа иногда удобнее записать в экспоненциальном виде
{:.0e}".format(details_num))
# Результат: Большие числа иногда удобнее записать в экспоненциальном виде
5e+05.
```

Здесь `.0e` означает, что число записывается в экспоненциальном виде без знаков после запятой.

Теперь, используя эти методы, вы сможете легко и удобно форматировать строки.

Методы split и join

Метод split

Метод `split` предназначен для разделения строки на части.

Синтаксис:

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

Здесь:

- `str` — строка, которую хотим разделить;
- `sep` — символы, по которым будет происходить разделение (по умолчанию пробел);
- `maxsplit` — максимальное количество разделений (по умолчанию -1, что означает отсутствие ограничений).

Примеры:

```python
test_for_split = 'Мы хотим получить каждый элемент отдельно!'
print(test_for_split.split())
```

Запустив метод без параметров, мы позволяем Python использовать стандартные параметры метода. По умолчанию разделителем является пробел, а максимальное ограничение равно -1, то есть количество разделений не ограничено. В итоге мы получим список:

```python
['Мы', 'хотим', 'получить', 'каждый', 'элемент', 'отдельно!']
```

Python ищет пробелы, выделяет слова между ними и помещает их в список.

Если использовать другой разделитель, например запятую:

```python
test_for_split = 'Мы,хотим,получить каждый,элемент отдельно!'
print(test_for_split.split(','))
```

Получим:

```python
['Мы', 'хотим', 'получить каждый', 'элемент отдельно!']
```

Если указать ограничение количества разделений:

```python
print(test_for_split.split(',', 1))
```

Получим:

```python
['Мы', 'хотим,получить каждый,элемент отдельно!']
```

Метод join

Метод `join` выполняет обратную задачу — объединяет элементы списка в строку, используя заданный разделитель.

Примеры:

```python
test_for_join = ['Мы', 'хотим', 'получить', 'каждый', 'элемент',
'отдельно!']
print(','.join(test_for_join))
```

В итоге получаем строку:

```python
'Мы,хотим,получить,каждый,элемент,отдельно!'
```

Метод `join` применяется к разделителю, который будет вставлен между элементами списка. При `split` мы обращались к строке и в качестве параметра использовали разделитель. При `join` мы обращаемся к разделителю и в качестве параметра используем список.

Можно использовать любой разделитель (пока он остаётся строкой), например:

```python
test_for_join = ['Мы', 'хотим', 'получить', 'каждый', 'элемент',
'отдельно!']
print('- даже так? - да - '.join(test_for_join))
```

Получим строку:

```python
'Мы - даже так? - да - хотим - даже так? - да - получить - даже так? - да -
каждый - даже так? - да - элемент - даже так? - да - отдельно!'
```

Синтаксис:

```python
str.join(iterable)
```

Здесь:

- `str` — строка-разделитель;
- `iterable` — итерируемый объект (например, список) с элементами-строками.

Теперь вы знаете, как разделять строки на части и соединять их обратно.

Методы startswith и endswith

Эти методы проверяют, начинается или заканчивается ли строка на указанную подстроку:

- `startswith` возвращает `True`, если строка начинается с указанной подстроки, иначе `False`.
- `endswith` возвращает `True`, если строка заканчивается на указанную подстроку, иначе `False`.

Пример:

```python
start_end = 'Начало, конец'
print(start_end.startswith('Начало'))  # Вернёт True.
print(start_end.endswith('конец'))     # Вернёт True.
```

Важно понимать, что такой поиск чувствителен к регистру:

```python
print(start_end.startswith('начало'))  # Вернёт False.
print(start_end.endswith('Конец'))     # Вернёт False.
```

Также вы можете указывать не один вариант, а, например, кортеж вариантов, каждый из которых будет проверен на наличие в вашей строке. Пример подобных приёмов можно найти в документации.

Методы upper и lower

Эти методы меняют регистр всех букв в строке:

- `upper` переводит все буквы в верхний регистр;
- `lower` переводит все буквы в нижний регистр.

Пример:

```python
start_end = 'Начало, конец'
print(start_end.upper())  # НАЧАЛО, КОНЕЦ
print(start_end.lower())  # начало, конец
```

Эти методы полезны при проверке ввода, чтобы игнорировать регистр. Иногда они могут помочь при проверке слов. Например, мы ждём на вход слово «Привет», но пользователь случайно вводит «ПрИвЕт», и наша проверка `if user_input == “Привет”:` уже не сработает.

Однако если мы напишем `if user_input.lower() == ‘привет’:`, то мы сразу учтём все возможные варианты слова «привет» из прописных и строчных букв.

```python
user_input = input("Введите 'Привет': ")
if user_input.lower() == 'привет':
    print("Вы ввели 'Привет'")
```

Возможные ошибки

Метод `join` работает только со строками. Если в списке есть числа или другие типы данных, нужно сначала преобразовать их в строки.

Пример:

```python
test_for_join = [1, '3', 3]
print('-'.join(test_for_join))  # Вызовет ошибку, ведь в списке у нас содержатся числа и строки, а не только строки.
```