Шпаргалка: операции со строками

Кратко

Что	Как	О чём помнить

Длина строки	Функция `len()`
	Схема:
	`len(<строка>)`
	Примеры:
	`len(s)`; `len('строка')`
		В функцию можно передать и переменную, и саму строку.
		Не забывай закавычивать строки и подстроки.

Вызов символа строки по индексу	**Схема:**
	`<строка>[<индекс символа>]`
	Пример:
	`print('строка'[0])`
	Будет напечатано: с
		Индекс **первого** символа строки равен `0`.
		Индекс **последнего** символа равен длине строки минус один — `len(<строка или имя переменной>) - 1`.
		Можно использовать отрицательные индексы.

Нахождение индекса символа через длину	**Схема:** `len(<строка>) - x`
	Пример:
	`last_index = len('строка') - 1`
	`print('строка'[last_index])`
	Напечатает: а	

Получение подстроки с помощью среза	**Схема:** `<строка>[<индекс начала>:<индекс конца>:<величина шага>]` **Пример:** `print('строка'[0:3])` Напечатает: стр	**Подстрока** — это любой фрагмент строки. Её можно получить с помощью среза. Срез строки не отличается от среза списка.

Проверить вхождение подстроки	Оператор `in`
	Схема: `print(<символ> in <строка>)`
	Пример:
	`s = 'строка'`
	`print('стр' in s)`
	Напечатает: True	

Разбить строку	Метод `split()` **Схема:** `<строка>.split(<разделитель>)` **Примеры:** `print('строка'.split('р'))` Напечатает: ['ст', 'ока'] `print('спи моя радость'.split())` Напечатает: ['спи', 'моя', 'радость']	Разделитель — символ или комбинация символов, которые есть в строке. В результате разбиения получится список. Подстроки станут его значениями. Разделитель в список **не сохраняется**. Разделитель по умолчанию — одиночный пробел.

Заменить одну подстроку на другую	Метод `replace()`
	Схема: `<строка>.replace(<подстрока>, <её замена>)`
	Пример:
	`new_line ='строка'.replace('ст', '').replace('о', 'у')`
	`print(new_line)`
	Напечатает: рука
		Чтобы убрать из строки подстроку, замени её на пустую строку `''`.
		Можно использовать с одной строкой сразу несколько раз.

Подробно

У строк много общего со списками. Если список — это последовательность элементов, то строка — последовательность символов.

Длина строки

Чтобы определить длину строки, используют функцию `len()`. В функцию можно передать не только переменную, но и саму строку.

Пример:

```
s = 'строка'

print(len(s)) # Будет напечатано: 6

print(len('строка')) # Будет напечатано: 6

print(len('фокус-покус')) # Будет напечатано: 11
```

Со списком тоже получится так сделать. Например, `print(len([1, 2]))` напечатает `2`. Но такую запись используют редко. Обычно строки и списки хранятся в переменных.

Индексы символов

У символов в строке, как и у элементов в списке, есть индексы. Здесь всё устроено точно так же:

- Индекс определяет, какое место символ занимает в общем ряду.
- Индекс первого символа — `0`.
- Индекс последнего символа равен длине строки минус один — `len(<строка или имя переменной>) - 1`. Например, у строки с длиной `5` индекс последнего символа равен `4`.
- Отрицательные индексы тоже работают. При обратном переборе индекс последнего символа — `-1`. А индекс первого символа равен длине строки со знаком минус. Например, `-5`. Это можно записать так: `-len(<строка или имя переменной>)`.

Как получить символ по индексу

Любой символ строки можно получить по индексу. Для этого в квадратных скобках после строки указывают индекс нужного символа.

Пример:

```
s = 'строка'

print(s[0])  # с
print(s[1])  # т

print(s[-2]) # к
print(s[-len(s)]) # с

print(s[len(s)])  # Будет выведена ошибка IndexError, так как индекс за пределами строки
# А вот так ошибки не будет:
print(s[len(s) - 1])  # а
```

Обрати внимание на разные формы записи. В первых трёх примерах индекс указан, а во вторых двух — найден через длину.

Чем отличаются строки и списки

В отличие от списков, строки — неизменяемые объекты. Поэтому нельзя заменить символ в строке по его индексу. Это приведёт к ошибке:

```
a = 'торт'

a[-1] = 'г'
# TypeError: 'str' object does not support item assignment
# Ошибка типа: объект строкового типа не поддерживает присвоение элемента

b = ['т', 'о', 'р', 'т']
b[-1] = 'г'
print(b) # ['т', 'о', 'р', 'г']
```

Что такое подстроки

В строках можно выделять подстроки. **Подстрока** — это любой фрагмент строки. Например, `'снег'` и `'ад'` в строке `'снегопад'`.

Когда получаешь символ строки по индексу, ты получаешь подстроку. Пусть она и состоит из одного символа.

С помощью индексов можно получать и более крупные подстроки. Такой способ называют **получением подстроки с помощью среза**. В этом случае подстрока — срез оригинальной строки.

Срез строки ничем не отличается от среза списка.

Как получить подстроку с помощью среза

Чтобы получить срез, нужно записать имя строки. После него поставить квадратные скобки и указать в них:

- индекс первого символа среза;
- двоеточие `:`;
- индекс на единицу больше, чем индекс последнего символа среза.

> 📢 Не забывай: последний элемент среза в него не входит. Поэтому нужно увеличивать значение на единицу.

Пример:

```
a = 'Жил-был в норе под землёй хоббит'
```

```
print(a[0:3])  # Жил
print(a[4:7])  # был
```

Можно указать и шаг среза. Например, `a[0:33:2]`. Тогда в подстроку войдёт каждый третий символ. Но на практике это почти не используют. И вот почему:

```
print(a[0:33:2]) # Жлблвнр о елйхби
```

Значения по умолчанию в срезе строки

В срезе строки, как и в срезе списка, есть значения по умолчанию. Если начало или конец среза совпадают с началом или концом строки, не указывай их индексы. Python сам подставит нужные значения на пустые места.

Пример:

```
a = 'Жил-был в норе под землёй хоббит'

print(a[8:])    # в норе под землёй хоббит
print(a[-6:])   # хоббит

print(a[:-7])   # Жил-был в норе под землёй
print(a[:7])    # Жил-был
```

Можно вообще не указывать индексы. Тогда ты получишь копию оригинальной строки:

```
print(a[:])
# Жил-был в норе под землёй хоббит
```

Оператор `in`

Оператор `in` ты тоже знаешь из темы про списки. Он проверяет, есть ли в списке нужный элемент. Точно так же он работает и со строками. Если подстрока содержится в строке, ты получишь `True`, если не содержится — `False`.

Но есть два нюанса:

- **Регистр букв имеет значение.** Для Python `'Баг'` и `'баг'` — это не одно и то же. Записывай подстроку точно, иначе получишь недостоверный результат.

- **Если ищешь в строке число, записывай его как строку.** Не забывай про кавычки. Например, `'7'`. Это нужно для совместимости типов данных.

```
print('40' in 'сорока')
# False

print('Хунга' in 'вулкан Хунга-Тонга-Хунга-Хаапай')
# True
```

Метод `split()`

Часто нужно разбить большую строку на подстроки или отдельные слова. К примеру, тестовый логин и пароль приходят как одна строка — `'example@mail.ru Qwerty1950'`. А тебе нужно отделить их и хранить по отдельности. С этой задачей справится метод `split()`.

Как вызвать метод `split()`

Чтобы его вызвать:

1. Укажи имя строки. Например, `line`.
2. Поставь точку и напиши имя метода — `line.split()`.
3. В круглые скобки передай **разделитель**. Например, `line.split(',')`.

Разделитель — это символ или комбинация символов, которые есть в строке. Он указывает программе, в каких местах нужно разбить последовательность. Если указать в скобках `','`, метод разобьёт строку по запятой.

Пример:

```
line = 'Волшебник никогда не опаздывает, Фродо Бэггинс.'

print(line.split(','))
# Будет напечатано: ['Волшебник никогда не опаздывает', ' Фродо Бэггинс.']

# Метод проверил строку, нашёл одну запятую и разделил по ней последовательность
```

В результате получится список. Подстроки станут его значениями.

Список можно записать в переменную и дальше работать с ней.

Особенности `split()`

1. **Разделитель не сохраняется в список.** Туда входят только подстроки, которые идут до и после него. Учитывай это, когда будешь использовать метод.

```
line = 'Волшебник никогда не опаздывает, Фродо Бэггинс.'

print(line.split('н'))
# Будет выведено: ['Волшеб', 'ик ', 'икогда ', 'е опаздывает, Фродо Бэгги', 'с.']
# В строках списка пропущены все буквы «н»
```

2. **Разделитель можно не указывать.** Если оставить скобки пустыми, строка разобьётся на отдельные слова по одиночному пробелу. Это разделитель по умолчанию. Python подставит такое значение автоматически, если ничего другого не найдёт.

```
line = 'Волшебник никогда не опаздывает, Фродо Бэггинс.'

print(line.split())
# Будет выведено: ['Волшебник', 'никогда', 'не', 'опаздывает,', 'Фродо', 'Бэггинс.']
```

Пример использования `split()` в цикле

Можно провести инвентаризацию и посчитать, сколько Великих Колец было в истории Средиземья. Дана такая строка: `'3:Эльфы;7:Гномы;9:Люди;1:Властелин Мордора'`. В ней хранятся нужные данные.

Следует выделить из неё количество колец с помощью метода `split()` и цикла `for`. А затем сложить их между собой.

Получится вот так:

```
count_rings = 0     # переменная для подсчета колец
string_of_rings = '3:Эльфы;7:Гномы;9:Люди;1:Властелин Мордора'

# сначала нужно разбить строку по символу ';'
# результат сохранили в lst
lst = string_of_rings.split(';')
print(lst)
# ['3:Эльфы', '7:Гномы', '9:Люди', '1:Властелин Мордора']

# теперь пройдёмся по списку lst и разобьём каждую строку по символу ':'
# сразу будем добавлять кольца к счётчику
for r in lst:
```

```
    ring = r.split(':')      # для первой итерации получится ['3', 'Эльфы']
    # дальше берём первый элемент в списке с помощью индекса,
    # присваиваем ему числовое значение и прибавляем к счётчику
    count_rings += int(ring[0])

print(count_rings)
# 20 колец
```

В теле цикла `for` две строчки кода. Первая разбивает каждую строку из `lst` ещё на две части и сохраняет результат в переменную `ring`. Вторая строчка кода берёт из списка `ring` первый элемент, преобразовывает его в число и прибавляет к `count_rings`.

Метод `replace()`

Этот метод меняет в строке одну подстроку на другую. Метод `replace()` создаёт копию строки и вносит изменения уже в неё. Исходная строка останется неизменной.

Как вызвать метод `replace()`

Чтобы его использовать:

1. Запиши имя строки. Например, `quote`.
2. Поставь точку и укажи имя метода — `replace()`.
3. В метод передай через запятую:
    - Первым аргументом — подстроку, которую нужно заменить. Например, `'Нельзя'`.
    - Вторым аргументом — подстроку, которая должна её заменить. Например, `'Можно'`.

В результате замены получится новая строка. Её можно сразу сохранить в переменную.

```
quote = 'Нельзя просто так взять и войти в Мордор'

new_quote = quote.replace('Нельзя', 'Можно')

print(new_quote)
# Можно просто так взять и войти в Мордор
```

```
print(quote)
# Нельзя просто так взять и войти в Мордор
```

Иногда указывают ещё третий аргумент — количество совпадений. Но это делать не обязательно. Его используют, только если нужно заменить часть совпадений.

Пример. В строке `quote2` слово `'вывозят'` встречается два раза. Звучит это не слишком красиво. От повтора лучше избавиться:

```
quote2 = 'Из них вывозят и вывозят куда-то мебель, ковры, картины, цветы, растения.'

new_quote2 = quote2.replace('вывозят', 'выносят', 1)

print(new_quote2)
# Из них выносят и вывозят куда-то мебель, ковры, картины, цветы, растения.
# Метод произвёл замену один раз
```

Особенности `replace()`

1. **Метод можно использовать для удаления подстрок.** Для этого в `replace()` вторым аргументом передают `''`. Ненужный фрагмент заменяется на пустую строку.

Пример:

```
quote = 'Можно просто так взять и войти в Мордор'

new_quote = quote.replace('просто так ', '')
# когда удаляешь элементы, не забывай следить за пробелами
# если не убрать лишний, получится задвоение

print(new_quote)
# Можно взять и войти в Мордор
```

2. **Метод можно использовать повторно.** После применения метода `replace()`, ты получишь новую строку. В ней можно сразу сделать ещё одну замену.

Если будешь вносить сразу несколько изменений, синтаксис будет таким: `<строка или имя переменной>.replace(...).replace(...).replace(...)`. Получится как бы многоэтажный дом. На каждом этаже будет создаваться копия строки. Она будет отличаться от предыдущей версии одной подстрокой.

Пример:

```
s = 'http://www.practicum.yandex.com/%20profile/'

# много разных замен, читать неудобно
new_s = s.replace('http', 'https').replace('www.', '').replace('com', 'ru').replace('%20', '')

print(new_s)
# 'https://practicum.yandex.ru/profile/'
```

Если замен много, конструкция становится громоздкой. Лучше так не делать и ограничиться одной или двумя заменами.

Сразу несколько замен делают, когда нужно исключить группу неподходящих символов. Это намного удобнее сделать через цикл `for`. Особенно если строка большая.

Пример использования `replace()` в цикле

Например, есть строка `string_to_replace`. В ней хранятся суммы в разных валютах. Нужно заменить все валюты на рубль — `'₽'`.

Чтобы это сделать, неподходящие знаки валют соберём в отдельном списке. Пусть он называется `replaced_symbols`. Затем напишем цикл `for`. Каждую итерацию будем заменять один валютный символ из `replaced_symbols` на `'₽'`. Так строка будет обновляться каждый круг цикла.

```
string_to_replace = '1000€ 345₽ 7738£ 984$ 345€ 4455¥ 34₽ 668$ 6284$ 145€'

replaced_symbols = ['€', '$', '£', '¥']
rub = '₽'

for s in replaced_symbols:
    string_to_replace = string_to_replace.replace(s, rub)

print(string_to_replace)
# 1000₽ 345₽ 7738₽ 984₽ 345₽ 4455₽ 34₽ 668₽ 6284₽ 145₽
```