

Урок 11.

Методы строк

Методы строк	2
Задания для закрепления	6
Поиск подстроки в строке	7
Задания для закрепления	11
Методы изменения регистра	12
Методы <code>strip</code> , <code>lstrip</code> , <code>rstrip</code>	19
Задания для закрепления	22
Практические задания	23

Методы строк

Для работы со строками доступен широкий набор встроенных методов. Эти методы позволяют выполнять различные операции: изменение регистра, поиск, замена подстрок, разбиение строки и многое другое.

Методы проверки предиката



Методы проверки предиката (или булевые методы) — это методы, которые возвращают True или False в зависимости от выполнения определённого условия для строки.

Эти методы используются для проверки, соответствует ли строка определённым критериям, таким как наличие только букв, только цифр, наличие пробелов и т.д.

Таблица с предикатными методами

Метод	Описание
isalpha()	Проверяет, состоит ли строка только из букв.
isdigit()	Проверяет, состоит ли строка только из цифр.
isalnum()	Проверяет, состоит ли строка только из букв и цифр.
isspace()	Проверяет, состоит ли строка только из пробелов.
islower()	Проверяет, все ли символы строки в нижнем регистре.
isupper()	Проверяет, все ли символы строки в верхнем регистре.
istitle()	Проверяет, начинается ли каждое слово в строке с заглавной буквы.
startswith()	Проверяет, начинается ли строка с указанной подстроки.
endswith()	Проверяет, заканчивается ли строка на указанную подстроку.
isdecimal()	Проверяет, состоит ли строка только из десятичных цифр.
isnumeric()	Проверяет, состоит ли строка только из числовых символов.
isascii()	Проверяет, состоит ли строка только из символов ASCII.

isprintable()	Проверяет, все ли символы строки являются печатаемыми.
isidentifier()	Проверяет, является ли строка допустимым идентификатором в Python.

Рассмотрим основные методы проверки предиката, которые используются чаще.

Примеры:

Python

```
# проверяет, состоит ли строка только из букв (без пробелов и других символов)
print("Hello".isalpha())

# проверяет, состоит ли строка только из цифр
print("12345.1".isdigit())

# проверяет, состоит ли строка только из букв и цифр (без пробелов и специальных символов)
print("Hello123".isalnum())

# проверяет, состоит ли строка только из пробельных символов (пробелы, табуляция и т.д.)
print(" ".isspace())

# проверяет, находятся ли все символы строки в нижнем регистре
print("Hello".islower())

# проверяет, находятся ли все символы строки в верхнем регистре
print("HELLO".isupper())
```

```
# проверяет, начинается ли каждое слово в строке с заглавной буквы
print("Hello world".istitle())

# проверяет, начинается ли строка с указанной подстроки
print("Hello world".startswith("Hello"))

# проверяет, заканчивается ли строка указанной подстрокой
print("Hello world!".endswith("world"))
```

Также есть и более специфичные методы, такие как проверка на печатаемые символы, десятичные числа или допустимые идентификаторы.

Примеры:

```
Python
# проверяет, состоит ли строка только из десятичных цифр, без символов вроде
"2"

print("123452".isdecimal())

# проверяет, состоит ли строка только из числовых символов, включая дробные
# числа или надстрочные символы

print("½".isnumeric())

# проверяет, состоит ли строка только из символов ASCII
```

```
print("Привет".isascii())

# проверяет, все ли символы в строке являются печатаемыми, например без символа
перевода строки

print("Hello\nWorld".isprintable())

# проверяет, является ли строка допустимым идентификатором (названием
переменной)

print("1variable".isidentifier())
```

•☆• Задания для закрепления

1. Какой результат будет выведен при выполнении следующего кода?

Python

```
text = "Python3"  
  
print(text.isalpha())
```

- a. True
- b. False
- c. Ошибка
- d. None

2. Какой результат будет выведен при выполнении следующего кода?

Python

```
text = "Hello World"  
  
print(text.istitle())
```

- a. True
- b. False
- c. Ошибка
- d. None

Поиск подстроки в строке

Python предоставляет несколько способов для поиска подстроки в строке. Эти методы позволяют определить, содержится ли подстрока в строке, и найти её местоположение, если она присутствует, а также подсчитать количество вхождений подстроки.

Таблица методов поиска подстроки в строке

Метод	Описание
find()	Возвращает индекс первого вхождения подстроки. Если подстрока не найдена, возвращает -1.
rfind()	Возвращает индекс последнего вхождения подстроки. Если подстрока не найдена, возвращает -1.
index()	Возвращает индекс первого вхождения подстроки. Если подстрока не найдена, вызывает ValueError.
rindex()	Возвращает индекс последнего вхождения подстроки. Если подстрока не найдена, вызывает ValueError.
count()	Возвращает количество вхождений подстроки в строке.

Методы find() и rfind()

find():

- Ищет первое вхождение подстроки и возвращает индекс её начала (первого элемента подстроки).
- Если подстрока не найдена, возвращает -1, что делает метод безопасным для использования без вызова исключений.

rfind():

- Ищет последнее вхождение подстроки и возвращает индекс её начала.
- Если подстрока не найдена, также возвращает -1.

Пример использования:

Python

```
text = "Python is awesome. Python is dynamic."  
  
# Поиск первого вхождения подстроки  
  
print(text.find("Python"))  
  
  
# Поиск последнего вхождения подстроки  
  
print(text.rfind("Python"))  
  
  
# Подстрока не найдена  
  
print(text.find("Java"))
```

Методы index() и rindex()

index():

- Работает аналогично find(), возвращает индекс первого вхождения подстроки.
- Если подстрока не найдена, вызывает исключение ValueError.

rindex():

- Работает аналогично rfind(), возвращает индекс последнего вхождения подстроки.
- Если подстрока не найдена, также вызывает исключение ValueError.

Пример использования:

Python

```
text = "Python is awesome. Python is dynamic."  
  
# Поиск первого вхождения подстроки  
  
print(text.index("Python"))  
  
  
# Поиск последнего вхождения подстроки  
  
print(text.rindex("Python"))  
  
  
# Попытка найти отсутствующую подстроку вызывает ошибку  
  
print(text.index("Java")) # ValueError
```

Метод count()

count():

- Возвращает количество вхождений подстроки в строке, т.к. подсчитывает сколько раз подстрока найдена в строке. Этот метод всегда безопасен и не вызывает исключений.

Пример использования:

Python

```
text = "Python is awesome. Python is dynamic."
```

```
# Подсчёт количества вхождений подстроки

print(text.count("Python"))

# Если подстрока не найдена, возвращает 0

print(text.count("Java"))

# Следующую подстроку ищет после конца предыдущей, т.е. без пересечений

text = "hahahahahaha"

print(text.count("ha"))

print(text.count("haha"))
```

 Задания для закрепления

1. Какой результат будет выведен при выполнении следующего кода?

Python

```
text = "Python is awesome. Python is dynamic."  
  
print(text.find("is"))
```

- a. 23
- b. 7
- c. 8
- d. -1

2. Какой результат будет выведен при выполнении следующего кода?

Python

```
text = "33333"  
  
print(text.count("33"))
```

- a. 1
- b. 2
- c. 3
- d. 5

Методы изменения регистра

В Python существуют несколько методов для работы с регистром строк. Эти методы позволяют преобразовывать символы строки в верхний или нижний регистр, а также обеспечивать правильное форматирование с заглавными буквами в начале каждого слова или строки.

При этом исходная строка остаётся неизменной, так как строки в Python являются неизменяемыми (immutable), и каждый метод возвращает новую строку с изменённым регистром.

Таблица методов изменения регистра

Метод	Описание
upper()	Преобразует все символы строки в верхний регистр.
lower()	Преобразует все символы строки в нижний регистр.
capitalize()	Преобразует первую букву строки в заглавную, остальные символы делает строчными.
title()	Преобразует первую букву каждого слова строки в заглавную.
swapcase()	Меняет регистр всех символов строки на противоположный.
casifold()	Преобразует строку в нижний регистр с учётом языковых особенностей для более точного сравнения.

Примеры:

```
Python
text = "Hello, wOrld!"

# Преобразование строки в верхний регистр

print(text.upper())
```

```
# Преобразование строки в нижний регистр

print(text.lower())

# Первая буква строки становится заглавной, остальные строчными

print(text.capitalize())

# Первая буква каждого слова становится заглавной

print(text.title())

# Меняет регистр всех символов на противоположный

print(text.swapcase())

# Приводит строку к нижнему регистру с учётом особенностей языка

print("Straßе".casfold())
```

Метод `replace`



Метод `replace()` используется для замены всех вхождений одной подстроки на другую в строке.

Он возвращает новую строку, оставляя исходную неизменной, поскольку строки в Python неизменяемы.

Синтаксис:

```
Python
string.replace(old, new, count=-1)
```

- **old** — подстрока, которую нужно заменить.
- **new** — подстрока, на которую будет заменена старая подстрока.
- **count (необязательный)** — количество замен. Если не указано, заменяются все вхождения.

Примеры использования:

```
Python
# Пример замены всех вхождений

text = "I love Python, Python is great"

new_text = text.replace("Python", "Java")

print(new_text)

# Пример с ограничением замен

text = "apple apple apple"

new_text = text.replace("apple", "orange", 1)

print(new_text)
```

Методы split и join

 Методы **split()** и **join()** используются для работы со строками, позволяя легко разделять строку на части и объединять список строк в одну строку с помощью разделителя.

Таблица методов split() и join():

Метод	Описание
-------	----------

split(sep, maxsplit=-1)	Разделяет строку по указанному разделителю и возвращает список строк. Можно ограничить количество разбиений.
join(iterable)	Объединяет элементы последовательности в строку, используя указанный разделитель.

Метод split()



Метод split() разделяет строку на части по указанному разделителю и возвращает список этих частей.

Синтаксис:

```
Python
string.split(sep, maxsplit=-1)
```

- **sep (необязательный)** — символ или подстрока, по которой происходит разделение. Если не указано, разделение происходит по пробелам.
- **maxsplit (необязательный)** — максимальное количество разбиений. Если не указано, строка разбивается на все возможные части.

Примеры использования:

```
Python
# Пример 1: Разделение строки по пробелам

text = "apple banana cherry"

fruits = text.split()

print(fruits)

# Пример 2: Разделение строки по запятой

text = "apple,banana,cherry"

fruits = text.split(",")
```

```
print(fruits)

# Пример 3: Разделение строки с ограничением количества разбиений

text = "apple---banana---cherry"

fruits = text.split("---", 1)

print(fruits)
```

Разница между `text.split()` и `text.split(" ")`



Метод `split()` работает по-разному в зависимости от того, указан пробел как разделитель или нет.

`string.split()`:

- Если не указать разделитель, метод автоматически разделяет строку по любым пробельным символам (это могут быть пробелы, табуляция, перенос строки и т.д.).
- Кроме того, несколько подряд идущих пробелов будут восприниматься как один элемент (по которому разбивать).

Пример:

```
Python
text = "apple      banana\t cherry\n"

fruits = text.split()  # Разделяет по любым пробельным символам

print(fruits)
```

string.split(" "):

- В этом случае метод разделяет строку только по пробелам.
- Если между словами несколько пробелов, они не будут восприниматься как один элемент, и это в результате создаст пустые строки.

Пример:

```
Python
text = "apple      banana\t cherry\n"

fruits = text.split(" ") # Разделяет только по пробелам

print(fruits)
```

Метод join()

Метод **join()** объединяет элементы последовательности (списка, кортежа и т.д.) в одну строку, используя указанный разделитель.

Синтаксис:

```
Python
separator.join(iterable)
```

- **separator** — строка, которая будет использоваться в качестве разделителя между элементами.
- **iterable** — последовательность (список, кортеж), элементы которой нужно объединить в строку.

Примеры использования:

```
Python
# Пример 1: Объединение списка строк с пробелом

fruits = ['apple', 'banana', 'cherry']
```

```
text = " ".join(fruits)

print(text)

# Пример 2: Объединение списка строк с запятой и пробелом

fruits = ['apple', 'banana', 'cherry']

text = ", ".join(fruits)

print(text)

# Пример 2: Объединение списка строк по пустому символу

litters = ['a', 'b', 'c', 'd', 'e']

text = "".join(litters)

print(text)
```

Методы strip, lstrip, rstrip



Методы strip(), lstrip() и rstrip() используются для удаления пробелов и других ненужных символов из строк.

Эти методы позволяют удалить символы либо с обеих сторон строки, либо только с левой или правой стороны.

Таблица методов

Метод	Описание
strip()	Удаляет пробелы или указанные символы с обеих сторон строки.
lstrip()	Удаляет пробелы или указанные символы только с левой стороны строки.
rstrip()	Удаляет пробелы или указанные символы только с правой стороны строки.

Метод strip()



Метод strip() удаляет пробелы или указанные символы с обоих концов строки (слева и справа).

Синтаксис:

Python

```
string.strip([chars])
```

- **chars (необязательный)** — символы, которые нужно удалить. Если не указано, по умолчанию удаляются пробелы.

Пример использования:

Python

```
text = "\t hello world\n "
```

```
print(text.strip()) # Удаляет все пробельные символы

text_with_chars = "***hello***"

print(text_with_chars.strip("*")) # Удаляет указанный символ

text_with_chars = "*--*hello---**"

print(text_with_chars.strip("-")) # Удаляет все указанные символы
```

Методы lstrip() и rstrip()



Методы `lstrip()` и `rstrip()` удаляют пробелы или указанные символы только с левой или правой стороны строки соответственно.

Примеры использования:

```
Python
text = " hello world "

print(text.lstrip()) # Вывод: "hello world"

text_with_chars = "***hello***"

print(text_with_chars.lstrip("*")) # Вывод: "hello***"
```

```
text = " hello world "

print(text.rstrip()) # Вывод: " hello world"

text_with_chars = "***hello***"

print(text_with_chars.rstrip("*")) # Вывод: "***hello"
```