Обработка строк к python

Строка представляет последовательность символов в кодировке Unicode, заключенных в кавычки. Причем для определения строк Python позволяет использовать как одинарные, так и двойные кавычки:

```
message = "Hello World!"
print(message)  # Hello World!

name = 'Tom'
print(name)  # Tom
```

Если строка длинная, ее можно разбить на части и разместить их на разных строках кода. В этом случае вся строка заключается в круглые скобки, а ее отдельные части - в кавычки:

```
text = ("Laudate omnes gentes laudate "
        "Magnificat in secula ")
print(text)
```

Управляющие последовательности в строке

Строка может содержать ряд специальных символов - управляющих последовательностей или escape-последовательности. Некоторые из них:

- \: позволяет добавить внутрь строки слеш
- \': позволяет добавить внутрь строки одинарную кавычку
- \": позволяет добавить внутрь строки двойную кавычку
- **\n**: осуществляет переход на новую строку
- **\t**: добавляет табуляцию (4 отступа)

Вставка значений в строку

Python позволяет встравивать в строку значения других переменных. Для этого внутри строки переменные размещаются в фигурных скобках {}, а перед всей строкой ставится символ **f**:

```
userName = "Tom"
userAge = 37
user = f"name: {userName}  age: {userAge}"
print(user)   # name: Tom  age: 37
```

Обращение к символам строки

И мы можем обратиться к отдельным символам строки по индексу в квадратных скобках:

```
string = "hello world"
c0 = string[0]  # h
print(c0)
c6 = string[6]  # w
print(c6)
```

Перебор строки

С помощью цикла **for** можно перебрать все символы строки:

```
string = "hello world"
for char in string:
    print(char)
```

Объединение строк

Одной из самых распространенных операций со строками является их объединение или конкатенация. Для объединения строк применяется операция сложения:

```
name = "Tom"
surname = "Smith"
fullname = name + " " + surname
print(fullname)  # Tom Smith
```

Повторение строки

Для повторения строки определенное количество раз применяется операция умножения:

```
print("a" * 3)  # aaa
```

Сравнение строк

Особо следует сказать о сравнении строк. При сравнении строк принимается во внимание символы и их регистр. Так, цифровой символ условно меньше, чем любой алфавитный символ. Алфавитный символ в верхнем регистре условно меньше, чем алфавитные символы в нижнем регистре. Например:

```
str1 = "1a"
str2 = "aa"
str3 = "Aa"
print(str1 > str2)  # False, так как первый символ в str1 - цифра
print(str2 > str3)  # True, так как первый символ в str2 - в нижнем регистре
```

Чтобы сменить регистр необходимо использовать:

Функция **lower()** приводит строку к нижнему регистру, а функция **upper()** - к верхнему.

```
str1 = "Tom"
str2 = "tom"
print(str1 == str2)  # False - строки не равны

print(str1.lower() == str2.lower())  # True
```

Поиск в строке

С помощью выражения term in string можно найти подстроку term в строке string. Если подстрока найдена, то выражение вернет значение True, иначе возвращается значение False:

```
string = "hello world"
exist = "hello" in string
print(exist)    # True

exist = "sword" in string
print(exist)    # False
```

Основные методы строк:

- **isalpha()**: возвращает True, если строка состоит только из алфавитных символов
- **islower()**: возвращает True, если строка состоит только из символов в нижнем регистре
- **isupper()**: возвращает True, если все символы строки в верхнем регистре
- **isdigit()**: возвращает True, если все символы строки - цифры
- **isnumeric()**: возвращает True, если строка представляет собой число
- **lower()**: переводит строку в нижний регистр
- **upper()**: переводит строку в вехний регистр
- **find(str[, start [, end])**: возвращает индекс подстроки в строке. Если подстрока не найдена, возвращается число -1
- **replace(old, new[, num])**: заменяет в строке одну подстроку на другую
- **split([delimeter[, num]])**: разбивает строку на подстроки в зависимости от разделителя
- **join(strs)**: объединяет строки в одну строку, вставляя между ними определенный разделитель

Пример поиска в строке:

Для поиска подстроки в строке в Python применяется метод **find(),** который возвращает индекс первого вхождения подстроки в строку и имеет три формы:

- **find(str)**: поиск подстроки str ведется с начала строки до ее конца
- **find(str, start):** параметр start задает начальный индекс, с которого будет производиться поиск
- **find(str, start, end):** параметр end задает конечный индекс, до которого будет идти поиск

  Если подстрока не найдена, метод возвращает -1:

```
welcome = "Hello world! Goodbye world!"
index = welcome.find("wor")
print(index)        # 6

# поиск с 10-го индекса
index = welcome.find("wor",10)
print(index)        # 21

# поиск с 10 по 15 индекс
index = welcome.find("wor",10,15)
print(index)        # -1
```

Разделение на подстроки

Метод **split()** разбивает строку на список подстрок в зависимости от разделителя. В качестве разделителя может выступать любой символ или последовательность символов. Данный метод имеет следующие формы:

- split(): в качестве разделителя используется пробел
- split(delimeter): в качестве разделителя используется delimeter
- split(delimeter, num): параметр num указывает, сколько вхождений delimeter используется для разделения. Оставшаяся часть строки добавляется в список без разделения на подстроки

```
text = "Это был огромный, в два обхвата дуб, с обломанными ветвями и с обломанной корой"
# разделение по пробелам
splitted_text = text.split()
print(splitted_text)
print(splitted_text[6])     # дуб,

# разбиение по запятым
splitted_text = text.split(",")
print(splitted_text)
print(splitted_text[1])     # в два обхвата дуб

# разбиение по первым пяти пробелам
splitted_text = text.split(" ", 5)
print(splitted_text)
print(splitted_text[5])     # обхвата дуб, с обломанными ветвями и с обломанной корой
```

Соединение строк

При рассмотрении простейших операций со строками было показано, как объединять строки с помощью операции сложения. Другую возможность для соединения строк представляет метод **join()**: он объединяет список строк. Причем текущая строка, у которой вызывается данный метод, используется в качестве разделителя:

```
words = ["Let", "me", "speak", "from", "my", "heart", "in", "English"]

# разделитель - пробел
sentence = " ".join(words)
print(sentence)  # Let me speak from my heart in English

# разделитель - вертикальная черта
sentence = " | ".join(words)
print(sentence)  # Let | me | speak | from | my | heart | in | English
```