МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Ф. УТКИНА

Python. Обработка текстовой информации.

Методические указания к лабораторной работе

Рязань 2021

УДК 004.432

Python. Обработка текстовой информации: методические указания к лабораторной работе №16 / Рязан. гос. радиотехн. ун-т.; сост.: А.В.Климухина, А.Н.Пылькин, Ю.С.Соколова, Е.С. Щенёв, М.Г. Щетинин. Рязань, 2021. – 24 с.

Рассмотрены основные правила работы со строковыми данными. Приведены сведения о составе таблицы ASCII и Unicode, которые определяют существующие стандарты кодирования символов, используемых в строковых данных. Рассмотрены особенности использования псевдографики. Приведены типовые задачи обработки символьных данных. Приведена информация по регулярным выражениям и использование этих объектов в Python.

В качестве практических заданий предлагается реализовать обработку строковых данных.

Предназначены для студентов очной и заочной формы обучения всех направлений подготовок и специальностей.

Табл. 7

Строки, таблицы ASCII, Unicode, обработка строковых данных, регулярные выражения.

Печатается по решению Научно-методического совета Рязанского государственного радиотехнического университета имени В.Ф. Уткина.

Рецензент: кафедра информатики, информационных технологий и защиты информации ФГБОУ ВО «Липецкий государственный педагогический университет им. П.П. Семенова-Тян-Шанского» (зав. каф., к.т.н., доц. Скуднев Д.М.).

Python. Обработка текстовой информации.

Составители: К л и м у х и н а Анастасия Витальевна

П ы л ь к и н Александр Николаевич

С о к о л о в а Юлия Сергеевна

Щ е н ё в Евгений Сергеевич

Щ е т и н и н Максим Геннадьевич

Обработка текстовой информации
Понятие строкового типа

В языке Python строка определяется как упорядоченная последовательность символов. Такая последовательность используется для хранения и представления текстовой информации, поэтому с помощью строк можно обрабатывать любую информацию и любые данные, которые могут быть представлены в текстовой форме. Отличительной особенностью строки служит заключение последовательности символов в апострофы или кавычки. Наличие двух вариантов позволяет вставлять в строки символы кавычек или апострофов, не используя специальных действий (называемыми экранированием).

В общем случае символы (литеры) используются для представления алфавитно-цифровой (символьной информации. Допустимое множество возможных символов определяется стандартом, который поддерживается тем или иным транслятором. Наиболее часто используется стандарт ASCII (American Standard Code for Information Interchange – американский стандартный код для обмена информацией), определенный в стандарте ISO/IEC 646:1991 «Информационные технологии. Семибитный набор кодированных символов ISO для обмена информацией». Код ASCII семиразрядный, т.е. стандартной кодировке подлежат 128 символов, включая прописные и строчные латинские буквы, арабские цифры от 0 до 9 и ряд других символов, причем латинские буквы и цифры в таблице ASCII (табл. 16.1) упорядочены и непрерывны, т.е. соблюдаются следующие отношения:

```
'A' < 'B' < 'C' < … < 'X' < 'Y' < 'Z'

'a' < 'b' < 'c' < … < 'x' < 'y' < 'z'

'0' < '1' < '2' < … < '7' < '8' < '9'
```

Первые 32 символа этого кода являются служебными/управляющими (например, символы «перевод строки» и «возврат каретки»), служат для управления передачей данных и не имеют символьного эквивалента. Восьмой бит определяет расширенный набор 128 символов свободной кодировки и может быть использован для национальных стандартов, например для алфавита языка, национальных мер, графических символов. Для кодировки

символов с номерами от 128 до 255 существует несколько различных стандартов: в MS DOS, Windows, Macintosh, ISO.

Таблица 16.1. Символы ASCII

Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.
0x00	NUL	0x20	space	0x40	@	0x60	`
0x01	SOH	0x21	!	0x41	A	0x61	a
0x02	STX	0x22	“	0x42	B	0x62	b
0x03	ETX	0x23	#	0x43	C	0x63	c
0x04	EOT	0x24	$	0x44	D	0x64	d
0x05	ENQ	0x25	%	0x45	E	0x65	e
0x06	ACK	0x26	&	0x46	F	0x66	f
0x07	BEL	0x27	‘	0x47	G	0x67	g
0x08	BS	0x28	(	0x48	H	0x68	h
0x09	HT	0x29	)	0x49	I	0x69	i
0x0A	LF	0x2A	*	0x4A	J	0x6A	j
0x0B	VT	0x2B	+	0x4B	K	0x6B	k
0x0C	FF	0x2C	,	0x4C	L	0x6C	l
0x0D	CR	0x2D	-	0x4D	M	0x6D	m
0x0E	SO	0x2E	.	0x4E	N	0x6E	n
0x0F	SI	0x2F	/	0x4F	O	0x6F	o
0x10	DLE	0x30	0	0x50	P	0x70	p
0x11	DC1	0x32	1	0x51	Q	0x71	q
0x12	DC2	0x32	2	0x52	R	0x72	r
0x13	DC3	0x33	3	0x53	S	0x73	s
0x14	DC4	0x34	4	0x54	T	0x74	t
0x15	NAK	0x35	5	0x55	U	0x75	u
0x16	SYN	0x36	6	0x56	V	0x76	v
0x17	ETB	0x37	7	0x57	W	0x77	w
0x18	CAN	0x38	8	0x58	X	0x78	x
0x19	EM	0x39	9	0x59	Y	0x79	y
0x1A	SUB	0x3A	:	0x5A	Z	0x7A	z
0x1B	ESC	0x3B	;	0x5B	[	0x7B	{
0x1C	FS	0x3C	<	0x5C	\	0x7C	|
0x1D	GS	0x3D	=	0x5D	]	0x7D	}
0x1E	RS	0x3E	>	0x5E	^	0x7E	~
0x1F	US	0x3F	?	0x5F	_	0x7F	DEL

Для создания универсального набора символов предложен двухбайтовый универсальный способ кодирования символьной информации в виде Unicode, описанный в стандарте ISO/IEC 10646:2003 «Информационные технологии. Универсальный многооктетный комплекс закодированных знаков (USC)», определяющий шестнадцатеричные коды символов различных языков и нотаций.

Стандарт Unicode (Юникод, иногда – Уникод) позволяет закодировать очень большое число символов из разных письменностей. В документах Юникода могут использоваться китайские, японские и корейские иероглифы, математические символы, буквы греческого алфавита, латиницы, кириллицы и др. При этом отсутствует необходимость переключения кодовых страниц. Юникод состоит из двух основных разделов:

- универсальны набор символов (universal character set, UCS);
- семейство кодировок (Unicode transformation format, UTF).

Коды в стандарте Юникод разделены на несколько областей. Область с кодами от U+0000 до U+007F содержит символы набора ASCII с соответствующими кодами. Далее расположены знаки различных письменностей, знаки пунктуации и технические символы. Часть кодов зарезервированы для будущего использования. Под символы кириллицы выделены области с кодами:

```
U+0400 – U+052F

U+2DE0 – U+2DFF

U+A640 до U+A69F
```

Известны различные версии стандарта Юникод. Первая версия использует кодировку с фиксированным размером символа в 16 бит, т.е. общее число кодов было $2^{16} = 65536$. Поэтому символы обозначают четырьмя шестнадцатеричными цифрами (например, U+052F). Стандарт Юникод постоянно обновляется: например, в стандарте версии 2.1 (май 1998) добавили символ евро, в стандарте версии 3.0 (сентябрь 1999) – символы шифра Брайля алфавит для незрячих людей), в стандарте версии 7.0 (июнь 2014) – символ рубля (код U+20BD). В настоящее время количество символов в стандарте Юникод превышает число сто тысяч.

Таким образом, используемый при записи строк Юникод является универсальным кодом для любого символа, независимо от платформы, программы или языка.

Часть символов сложно ввести с клавиатуры, поэтому для них обозначения используют экранированные последовательности. В табл. 16.2 приведены экранированые последовательности, которые позволяют ввести с клавиатуры и вставить в строку служебные символы.

Таблица 16.2. Экранированные последовательности

Экранированная последовательность	**Назначение**
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhh	16-битовы символ Юникода в 16-ричном представлении
\Uhhhhhhhh	32-битовый символ Юникода в 16-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

Использование в экранированной последовательности символа \ (обратный слэш) может привести к неоднозначности. Например, при обозначении пути в дереве каталогов используется слеш. Поэтому, если символ \ не имеет смысла экранированной последовательности, применяют так называемые «сырые» строки, в которых перед открывающей кавычкой стоит символ ‘r’ (в любом регистре), например:

```
S = r'C:\newt.txt'
```

«Сырая» строка не может заканчиваться символом обратного слэша, поэтому применяют искусственные решения:

```
S = r'\n\n\\'[:-1]
S = r'\n\n' + '\\'
S = '\\n\\n\\'
```

Строки Юникода (Unicode)

Для обеспечения обработки всех символов, которые используются в современных алфавитах и древних текстах, в Python имеется возможность использовать стоки Юникода. Использование строк Юникода (вместо стандарта ASCII) позволяет эффективно решать проблемы интернационализации программного обеспечения. Эта проблема решается определением единого стандарта Unicode для всех возможных символов.

Создание строки Юникода характеризуется наличием символа ‘u’ перед кавычками. Например, строка:

```
u 'Hello Python!'
```

реализует универсальную международную кодировку строки символов.

В строку можно включить специальные символы с помощью применения управляющих последовательностей. Строка

```
u 'Hello\u0020Python!'
```

соответствует строке

```
u 'Hello Python!'
```

В рассмотренном примере управляющая последовательность \u0020 указывает, что необходимо вставить символ из таблицы Unicode с порядковым номером в шестнадцатеричной системе исчисления U+0020 (пробел).

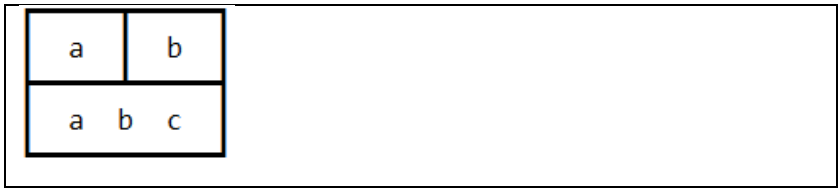
Псевдографика

Для вывода табличной информации в текстовый консоли можно воспользоваться специальными символами псевдографики, предназначенных для рисования границ таблицы. Коды данных символов по Юникоду представлены в таблице 16.3.

Таблица 16.3. Символы псевдографики

Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.
\u250C	┌	\u2550	═	\u2558	╘	\u256B	╫
\u2500	─	\u2557	╗	\u255B	╛	\u2562	╢
\u2510	┐	\u255A	╚	\u255E	╞	\u2568	╨
\u2514	└	\u255D	╝	\u256A	╪	\u2565	╥
\u2518	┘	\u2560	╠	\u2561	╡	\u2592	▒
\u251C	├	\u256C	╬	\u2567	╧	\u2593	▓
\u253C	┼	\u2563	╣	\u2564	╤	\u2588	█
\u2524	┤	\u2551	║	\u2553	╓	\u2584	▄
\u2502	│	\u2569	╩	\u2556	╖	\u258C	▌
\u2534	┴	\u2566	╦	\u2559	╙	\u2590	▐
\u252C	┬	\u2552	╒	\u255C	╜	\u2580	▀
\u2554	╔	\u2555	╕	\u255F	╟		

С помощью символов псевдографики можно вывести в консоль или в текстовый файл, например, такую таблицу:



Замечание. Для того чтобы в тексте программы символы псевдографики были представлены наглядно, а не кодами, их можно копировать и вставлять в текст программы. Например, нужные символы можно вывести с помощью их кодов в консоль Python, в затем оттуда скопировать сами символы.

Обработка строк

Основные операции, функции и методы для обработки строк представлены в таблице 16.4. Для некоторых из них также показаны примеры использования.

Таблица 16.4. Операции, функции и методы для обработки строк

Операция, функция или метод	Описание
`S1 + S2`	Конкатенация (сложение); `S1, S2` - строки S = 'abc' + 'def' S == 'abcdef'
`S1 * n`	Повтор строки; `S1` - строка; `n` - целое число S = 'ab' * 3 S == 'ababab'
`S[i]`	Доступ по индексу; `S` - строка; `i` - целое число S = 'abc' S[0] == 'a'
`S[i:j:step]`	Извлечение среза; `S` - строка; `i,j,step` – целые числа (начальный индекс, конечный индекс и шаг) S = 'abcd' S[:] == 'abcd' S[1:2] == 'bc' S[:-1] == 'abc' S[1:] == 'bcd' S[::-1] == 'dcba'
`len(S)`	Длина стоки len(‘abc’) == 3
`S.find(str,`	Поиск подстроки в строке.

`[start],[end])`	Возвращает номер первого вхождения или -1
`S.rfind(str, [start],[end])`	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
`S.index(str, [start],[end])`	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError

Продолжение таблицы 16.4

Операция, функция или метод	**Описание**
`S.rindex(str, [start],[end])`	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
`S.replace(шаблон, замена)`	Замена шаблона
`S.split(символ)`	Разбиение строки по разделителю
`S.isdigit()`	Состоит ли строка из цифр
`S.isalpha()`	Состоит ли строка из букв
`S.isalnum()`	Состоит ли строка из цифр или букв
`S.islower()`	Состоит ли строка из символов в нижнем регистре
`S.isupper()`	Состоит ли строка из символов в верхнем регистре
`S.isspace()`	Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы (‘\f’), «новая строка» (‘\n’), «перевод каретки» (‘\r’), «горизонтальная табуляция» (‘\t’) и «вертикальная табуляция» (‘\v’))
`S.istitle()`	Начинаются ли слова в строке с заглавной буквы
`S.upper()`	Преобразовать строки к верхнему регистру
`S.lower()`	Преобразовать строки к нижнему регистру

`S.startwith(str)`	Начинается ли строка S с шаблона str
`S.endwith(str)`	Заканчивается ли строка S шаблоном str
`S.join()`	Сборка строки из списка с разделителем S
`ord()`	Символ в его код ASCII
`chr()`	Код ASCII в символ
`S.capitalize()`	Переводит первый символ строки в верхний регистр, а все остальные в нижний

Продолжение таблицы 16.4

Операция, функция или метод	Описание
`S.center(width, [fill])`	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)
`S.count(str, [start], [end])`	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
`S.expandtabs([tabsize])`	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам
`S.lstrip([chars])`	Удаление пробельных символов в начале строки
`S.rstrip([chars])`	Удаление пробельных символов в конце строки
`S.strip([chars])`	Удаление пробельных символов в начале и конце строки
`S.partition(шаблон)`	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращает кортеж, содержащий саму строку, а затем две пустые строки
`S.rpartition(sep)`	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращает кортеж, содержащий саму две пустые строки, а затем саму строку
`S.swapcase()`	Переводит символы нижнего регистра в верхний, а верхнего – в нижний

Окончание таблицы 16.4

Операция, функция или метод	Описание
`S.title()`	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
`S.zfill(width)`	Делает длину строки не меньшей width по необходимости заполняя первые символы нулями
`S.ljust(width, fillchar= " ")`	Делает длину строки не меньшей width по необходимости заполняя последние символы символом fillchar
`S.rjust(width, fillchar= " ")`	Делает длину строки не меньшей width по необходимости заполняя первые символы символом fillchar

Пример 1. Вывести строку в перевернутом виде и по отдельным словам.

Для вывода строки в перевернутом виде необходимо вывести все ее символы в обратном порядке, начиная с последнего. Это удобно сделать с помощью операции извлечения среза: s[::-1]. В качестве параметров здесь указывается только отрицательный шаг, равный -1.

С выводом строки, по отдельным словам, несколько сложнее. Алгоритм состоит из цикла по всем символами строки, в котором текущий символ сравнивается с пробелом. Если встречен пробел, то вместо него на экран выводится перенос стоки (символ ‘\n’ является значением по умолчанию для параметров end в команде print). Если же символ отличен от пробела, то он выводится на экран без переноса строки.

Код программы:

```
print("Введите строку: ", end='')
s = input()
print("Перевернутая строка: ", end='')
print(s[::-1])
print("Отдельные слова")
for c in s:
    if c == ' ':
```

```
        print("")
    else:
        print(c, end='')
```

Результат выполнения программы:

```
Введите строку: 123 456
Перевернутая строка: 654 321
Отдельные слова
123
456
```

Пример 2. Проверка баланса скобок в скобочном выражении.

Скобочное выражение – строка, содержащая в себе открывающие и закрывающие скобки, а также другие символы между этими скобками. Для простоты рассмотрим случай с выражением, содержащим только круглые скобки (скобки одного вида). В скобочном выражении открывающаяся скобка должна идти всегда до закрывающей, а количества открывающих и закрывающих скобок должны совпадать. Иными словами, все скобки должны образовывать пары из открывающей и закрывающей скобок.

Алгоритм программы состоит в ведении счета открытых пар скобок. Если при проходе по символам строки встречается открывающая скобка, то она увеличивает счетчик открытых пар. Закрывающая же скобка уменьшит этот счетчик. В правильном случае счетчик должен быть всегда неотрицательный, а в конце строки – равен нулю. Если к концу строки счетчик больше нудя, то в строке не хватает закрывающих скобок. Если же в процессе анализа строки счетчик стал отрицательным, то в момент была найдена лишняя закрывающая скобка, не имеющая перед собой парной открывающей.

Код программы:

```
print("Введите скобочное выражение: ", end='')
s = input()
k = 1
i = 0
for c in s:
    if c == '(':
        i += 1
    elif c ==')':
```

```
        i -= 1
    if i < 0:
        break
if i == 0:
    print('Скобки расставлены правильно')
else:
    print('Скобки расставлены неправильно')
```

Пример 3. Поменять в строке одно сочетание букв на другое.

Для замены подстроки в строке используется метод replace, вызываемый у строки. Входными параметрами передаются искомая подстрока и замена. Метод не меняет строку, от которой он вызван, но возвращается новое значение. При этом по умолчанию заменяются все найденные подстроки. Также можно третьим параметром в метод replace передать максимальное количество замен подстроки.

Код программы:

```
print("Введите строку: ", end='')
s = str(input())
print("Исходное сочетание букв: ", end='')
oldstr = input()
print("Замена: ", end='')
newstr = input()
replace = s.replace(oldstr,newstr)
print(replace)
```

Результат выполнения программы:

```
Введите строку: проверка программы
Исходное сочетание букв: ро
Замена: 123
п123верка п123граммы
```

Пример 4. Определить в строке количество слов длиннее 4 символов. Слова разделены пробелом.

Данная задача решается путем последовательного поиска всех пробелов в строке и вычисления количества символов между соседними пробелами. Для этого хорошо подходит строковый метод

find. Первым параметром в него передается искомая подстрока. Дополнительно можно указать в качестве второго параметра начальную позицию поиска. Таким образом, будем искать первое появление в строке пробела, запоминать его позицию, а затем продолжать поиск со следующего за найденным пробелом символа. При этом нужно не забывать про основную задачу – подсчет найденных слов длине 4 символов. Для этого будет вычислять разницу позиций каждого найденного пробела и предыдущего.

Код программы:

```
print("Исходная строка: ", end='')
s = str(input())
currentsp = 0
nextsp = 0
counter = 0
while currentsp < len(s):
    nextsp = s.find(' ', currentsp + 1)
    if nextsp == -1:
        nextsp = len(s)
    if (nextsp - currentsp) > 5:
        counter += 1
    currentsp = nextsp
print("Количество слов длиннее 4 символов: %d" %
\
counter)
```

Результат выполнения программы:

```
Исходная строка: раз два три четыре пять шесть семь
Количество слов длиннее 4 символов: 2
```

Пример 5. Удалить «лишние» пробелы в строке, содержащей слова.

Лишние пробелы в строке можно разделить на два типа: пробелы в начале/конце строки и серии пробелов между словами. Пробелы по краям строки можно легко удалить методом `strip`. А вот с сериями пробелов в середине строки сложнее. Каждую серию нужно заменить на одиночный пробел. В этом нам поможет упомянутый ранее метод `replace`, меняющий два пробела на один. Однако такая

замена будет плохо работать на больших сериях пробелов, поэтому ее необходимо будет обернуть в цикл с предусловие, который завершиться, когда в строке не будет найдено ни одного вхождения двух пробелов подряд.

Код программы:

```
print("Исходная строка: ", end='')
s = str(input())
result = s.strip()
while result.find('  ') != -1:
    result = result.replace('  ',' ')
print("Строка без лишних пробелов: " + result,
end='')
```

Результат выполнения программы:

```
Исходная строка:     тест    тест     тест
Строка без лишних пробелов: тест тест тест
```

Регулярные выражения

Регулярное выражение (regular expression, regex, «регулярка») – это строка, определяющая шаблон для поиска подстрок в строке. Регулярные выражения представляют собой очень мощный, но при этом не менее сложный инструмент для обработки строковых данных. Многие примеры, рассмотренные ранее, можно решить более просто и изящно с помощью них (об этом речь пойдет несколько позже).

В большинстве языков программирования, в том числе и в Python, присутствуют встроенные средства для работы с регулярными выражениями. При этом существуют различные диалекты синтаксиса регулярных выражений, что вызывает дополнительные сложности при попытке переноса таких выражений между различными проектами.

Также важно понимать, что регулярные выражения не являются панацеей от всех проблем и единственным способом работы со строками. Хоть сколько-то сложное выражение становится трудночитаемым, его правка часто сводится к написанию нового выражения с нуля. Плохо составленное (или неуместное в контексте алгоритма) регулярное выражение может существенно замедлить работу программы.

«Антипримером» использования регулярных выражений можно привести следующий шаблон для проверки правильности ввода email-адреса:

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-
9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-
\x08\x0b\x0c\x0e\x1f\x21\x23-\x5b\x5d-\x7f}|\\[\x01-\
x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9]*[a-z0-
9])?\.)+ [a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[?:(:25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9] :(?:[
\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53\-\x7f]|\\[
\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

Как видите, данный код является абсолютно нечитаемым. В случае необходимости его модификации или устранения ошибки единственным вариантом будет полностью переделать все с нуля. Именно поэтому регулярными выражениями нельзя злоупотреблять. *Впрочем, подобные претензии уместны только для регулярных выражений, написанных в своих проектах. Конкретно этот пример регулярного выражения для проверки email-адреса взят с сайта https://emailregex.com/, является отлаженным и часто используемым, поэтому его использование приемлемо в некоторых случаях. Но пытаться изменить такие примеры для своих нужд не стоит, это в большинстве случаев лишь породит множество труднонаходимых ошибок.*

Далее рассмотрим основы синтаксиса регулярных выражений.

Любая строка сама по себе является регулярным выражением, то есть описывает определенный шаблон для поиска подстрок. Также в регулярных выражениях есть свои символы:

```
.^$*+?{}[]\|()
```

Перечисленные выше символы являются служебными, для их употребления в строке в качестве обычных символов необходимо использовать экранирование символом \.

В табл. 16.5 представлены шаблоны, описывающие один символ. Регулярное выражение обычно состоит из нескольких шаблонов.

Таблица 16.5. Шаблоны, описывающие один символ

Шаблон	Описание	Пример
.	Любой символ, кроме \n	Шаблон: т.ст Проверка: тост тест протест
\d	Цифра	Шаблон: \d - й Проверка: 1-й 2-й 3-я
\D	Любой символ, кроме цифры	Шаблон: 12\ D45 Проверка: 12345 012a45
\s	Любой пробельный символ (пробел, табуляция, конец строки и т.п.)	Шаблон: ер\s Проверка: пример проверки
\S	Любой непробельный символ	Шаблон: ер\ S Проверка: пример проверки
\w	Любая буква, цифра или знак нижнее подчеркивание «_»	Шаблон: тест\ w Проверка: тест! протест тесто
\W	Любой символ, кроме букв, цифр и знака нижнее подчеркивание «_»	Шаблон: тест\ W Проверка: тест! протест тесто
[..]	Любой из символов, перечисленных внутри скобок (возможно указание в виде диапазона)	Шаблон: [0-9] [АБВГ] Проверка: 1А 4Б 7Р 22
[^..]	Любой символ, кроме перечисленных	Шаблон: 1[^2]3 Проверка: 123 133 143
\b	Начало или конец слова (позиция между символами)	Шаблон: \bтест Проверка: тест! протест тесто
\B	Не начало и не конец слова	Шаблон: \Bтест Проверка: тест! протест тесто

Стоит заметить, что символы табуляции (\t) и новой строки (\n) также можно использовать в регулярных выражениях. При указании диапазона символов не стоит забывать, что буква Ё не входит в основной блок кириллических символов. Полный диапазон символов для кириллицы выглядит следующим образом: [А-Яа-яЁё].

В табл. 16.6 представлены квантификаторы, служащие для указания количества повторений символа.

Таблица 16.6. Квантификаторы.

Шаблон	**Описание**	**Пример**
{n}	Ровно n повторений	Шаблон: \d{3} Проверка: 1 12 123 1234 12345
{m,n}	От m до n повторений (включительно)	Шаблон: \ d{2, 4} Проверка: 1 12 123 1234 12345
{m,}	Не менее m повторений	Шаблон: \d{3,} Проверка: 1 12 123 1234 12345
{,n}	Не более n повторений	Шаблон: \ d{,2} Проверка: 1 12 123 1234 12345
?	Ноль или одно повторение символа	Шаблон: тест\w? Проверка: тест! протест тестовый
*	Ноль или более повторений	Шаблон: тест\w* Проверка: тест! протест тестовый
+	Одно или более повторение	Шаблон: тест\w+ Проверка: тест! протест тестовый

По умолчанию все квантификаторы являются жадными, то есть захватывают как можно больше символов. Если в конце квантификатора добавить «?», то он станет ленивым – будет захватывать как можно меньшее количество символов.

Регулярные выражения в Python

В Python представлены свои функции для работы с регулярными выражениями. Расположены они в модуле re, который необходимо подключать в начале программы. Основные функции из этого модуля описаны в табл. 16.7.

Таблица 16.7. Функции для работы с регулярными выражениями

Функция	Описание
`re.search(pattern, string)`	Поиск в строке `string` первой подстроки, подходящей под шаблон `pattern`
`re.fullmatch(pattern, string)`	Проверка, подходит ли строка `string` под шаблон `pattern`
`re.split(pattern, string, maxsplit=0)`	Разделение строки на подстроки; аналог `str.split()`, только разделение происходит по подстрокам, подходящим под шаблон `pattern`
`re.findall(pattern, string)`	Поиск в строке `string` всех непересекающихся подстрок, под ходящих под шаблон `pattern`
`re.sub(pattern, repl, string, count=0)`	Замена в строке `string` всех непересекающихся подстрок по шаблону `pattern` на подстроку `repl`

Благодаря описанным выше функциям можно упростить решение некоторых примеров, рассмотренных ранее. Нумерация примеров сохранена.

Пример 6. Вывести строку в перевернутом виде и по отдельным словам.

В данном примере можно существенно упростить механизм вывода строки по отдельным словам. Для этого потребуется разбить строку на массив слов с помощью функции `re.split()`, шаблоном разделителя в которой будет серия из одного или нескольких пробелов.

Код программы:

```
import re
print("Введите строку: ", end='')
s = input()
print("Перевернутая строка: ", end='')
```

```
print(s[::-1])
print("Отдельные слова: ")
words = re.split(" +", s)
for word in words:
    print(word)
```

Пример 7. Определить в строке количество слов длиннее 4 символов. Слова разделены пробелом.

Здесь можно заменить ручной поиск слов через перебор символов вызовом функции `re.findall()`. В качестве шаблона зададим последовательность из 5 или более букв. Функция вернет список результатов (тип `list`). Нам останется только посмотреть размер этого списка, для чего используем функцию `len()`.

Код программы:

```
import re
print("Исходная строка: ", end='')
s = str(input())
bigwords = re.findall("\w{5,}", s)
print("Количество слов длиннее 4-х символов: %d"
%)
len(bigwords)
```

Пример 8. Удалить «лишние» пробелы в строке, содержащей слова.

В данном случае можно избавиться от недостатка функции `str.replace()`, заменив ее на функцию `re.sub()` с шаблоном замены в виде последовательности пробелов от 2 и более. В этом случае не понадобится делать обертку в виде цикла с предусловием, необходимые замены будут сделаны за один заход и за минимальное количество действий.

Код программы:

```
import re
print("Исходная строка: ", end='')
s = str(input())
result = s.strip()
result = re.sub(" {2,}", " ", result)
print("Строка без лишних пробелов: " + result,
end='')
```

Контрольные вопросы

1. Что такое строка в Python?
2. В чем разница между ASCII и Unicode?
3. Зачем нужны экранированные последовательности?
4. Для чего предназначены символы псевдографики?
5. Назовите основные программные средства обработки строк.
6. Что такое регулярные выражения?
7. Зачем нужны квантификаторы в регулярных выражениях?

Задания

Вариант 1. Определить, является ли вводимая последовательность символов идентификатором.

Вариант 2. Подсчитать количество сдвоенных символов «сс», «нн», «лл» во введенном тексте.

Вариант 3. Разбить произвольный текст на строки определенной длины. При переносе слова предусмотреть вывод дефиса.

Вариант 4. Дана символьная строка. Подсчитать, сколько раз в ней встречается подслово ABBA.

Вариант 5. Найти во введённом тексте само длинное и самое короткое слова.

Вариант 6. Из заданной строки исключить все символы, входящие в нее более одного раза.

Вариант 7. Проверить правильно ли в заданном тексте расставлены круглые скобки

Вариант 8. В заданной последовательности символов подсчитать общее количество символов «+», «-», «*» и исключить их из текста.

Вариант 9. Вводится последовательность ключевых слов. Отсортировать их по алфавиту.

Вариант 10. В предложении, содержащем не менее двух слов, поменять местами первое и последнее слово.

Вариант 11. Сформировать строку, состоящую из символов, входящих одновременно в обе заданные строки

Вариант 12. Откорректировать заданный текст, заменив в нем все вхождения одной буквы на другую.

Вариант 13. В заданном тексте перевернуть каждое слово.

Вариант 14. Дана символьная строка. Определить длину самой длинной подстроки из подряд состоящих букв «а».

Вариант 15. Дана строка символов. Определить, сколько в ней слов, начинающихся и кончающихся на одну и ту же букву.

Вариант 16. В заданной строке x заменить все вхождения подстроки y на подстроку z.

Вариант 17. Для заданного символа определить, сколько раз он встречается во введенном тексте.

Вариант 18. Из произвольной последовательности символов удалить лишние пробелы, разделяющие слова.

Вариант 19. Из строки символов исключить однобуквенные слова.

Вариант 20. Из заданной последовательности символов удалить лишние проделы, разделяющие слова.

Вариант 21. Выяснить, верно ли, что среди символов строки произвольной длинны имеются все символы, входящие в слово ДЕНЬ.

Вариант 22. Для каждого из слов заданного предложения указать, сколько раз оно встречается в предложении.

Вариант 23. В заданной строке символов исключить все группы символов вида ABC.

Вариант 24. Определить, можно ли из символов заданной строки составить вашу фамилию.

Вариант 25. В заданной строке символов исключить нелитерные символы.

Вариант 26. Заданы две строки символов. Определить, можно ли из символов первой строки сформировать вторую строку.

Вариант 27. В заданной строке символов удалить все слова, длины которых больше 5.

Вариант 28. Заданы две предложения, состоящие из слов (слова разделены пробелами). Поменять местами слова в двух предложениях, длины которых одинаковы.

Вариант 29. Задано предложение из слов, которые разделены пробелами. Расставить слова 1 предложения по длине слов, начиная с наименьшего слова.

Вариант 30. Задан список слов (английский и русских). Упорядочить слова в алфавитном порядке, вначале русские слова, а затем английские.

Оглавление