Тема 2.5 СТРОКИ

Строки в Python (str) - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации.

Строки — это массивы байтов, которые представляют символы Unicode. Они состоят из одного или нескольких символов, заключенных в кавычки.

Строка может заключаться как в апострофы (одинарные кавычки), так и в двойные кавычки. Чтобы создать строку, нужно присвоить переменной последовательность символов, заключённую в кавычки. Например:

S = 'spams'

S = "spams"

Строки можно писать также в тройных кавычках. Используется тогда, когда внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трёх кавычек подряд, а также можно использовать для записи многострочных блоков текста.

S = ''' Строка в "Python" состоит из последовательности символов'''

В Python нет отдельное типа для представления одного символа, символ — это просто единичная строка (*S = 'a'*).

Строка считывается со стандартного ввода функцией **input()**.

Строки могут выводиться на экран с использованием функции вывода. Например:

print("привет")

Любой другой объект в Python можно перевести к строке, которая ему соответствует. Для этого нужно вызвать функцию **str()**, передав ей в качестве параметра объект, переводимый в строку.

Символы хранятся в памяти в виде кодов, базирующихся на Юникоде. ***Юникод*** – таблица, которая содержит соответствия между числом и каким-либо знаком, причем количество знаков может быть любым.

Код	Символ	Код	Символ	Код	Символ	Код	Символ
32	пробел	56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	"	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	'	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v

47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	|
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	□

Для того чтобы узнать код некоторого символа, существует функция **ord()**, например, ord('s') даст результат 115.

Зная код, всегда можно получить соответствующий ему символ. Для этого существует функция **chr()**, например chr(100) даст результат 'd'.

В Юникод все время добавляются новые элементы, а сам размер этой таблицы не ограничен и будет только расти, поэтому сейчас при хранении в памяти одного юникод-символа может потребоваться от 1 до 8 байт. Отсутствие ограничений привело к тому, что стали появляться символы на все случаи жизни. Например, есть несколько снеговиков.

Этого вы можете увидеть, если наберете:

```
print('\u2603')
```

Строку можно рассматривать как массив символов и соответственно обращаться к каждому символу по отдельности.

Срез (slice) – извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Есть три формы срезов:

S[i] - срез, состоящий из одного символа, который имеет номер i. При этом считается, что нумерация начинается с 0.

S[a:b] - срез с двумя параметрами возвращает подстроку из **b - a** символов, начиная с символа с индексом **a** до символа с индексом **b**, не включая его.

S[a:b:d]- срез с тремя параметрами, в котором третий параметр задает шаг.

Примеры:

```
>str = 'string'
 >print str[0]  # Вывод первого (нулевого) символа
 >s
 >print str[2]  # Вывод 3-го символа (т.к счет идет от 0).
 >r
 >print str[-1]  # Данным способом мы выводим последний символ строки.
 >g
```

>print str[0:3] # Выводим диапазон символов от 0 до 3. Все 4 символа. Можно записать так [:3]

>stri

>print str S[::-1] # Данным способом можно перевернуть строку

>gnirts

>print str[0:5:2] # Выводим символы от 0 до 5 с шагом в 2 символа (т.е.перескакиваем через 1 символ).

>srn

>print str[:] # Выводим всю строку. Поскольку нули можно не указывать. Эквивалентно [0:0]

>string

Со строками можно выполнять ***математические операции***, которые позволяют быстро преобразовывать любые последовательности символов:

1) Конкатенация (сложение)

>s1 = 'привет '

>s2 = 'мир '

>print s1+s2 # Конкатенация строк (добавление)

>привет мир

2) Дублирование строки (умножение)

*>print s1*3 # Умножение строки*

>приветприветпривет

Python включает **форматирование строк**, это подразумевает подстановку какого-либо шаблона в определенное место (или в определенные позиции) текста. Для этого в коде на Python используют метод **format**

Например:

1) Одна подстановка:

'Hello, {}!'.format('Vasya') # 'Hello, Vasya!'

Аргументом метода является текст-подстановка, который при исполнении программы подставляется на место фигурных скобок.

2) Несколько подстановок:

'{0}{1}{0}'.format('abra', 'cad') # 'abracadabra'

Подстановки нумеруются, аргументы метода format заполняют позиции для подстановок согласно их порядковым номерам, указанным в фигурных скобках.

3) другой вариант форматирования с множественными подстановками:

'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')

'Coordinates: 37.24N, -115.81W'

Аргументы метода format заполняются согласно указанным именам заполнителей.

Для форматирования строк в языке Python используют служебные символы (таблица 1). Как правило, большинство из них позволяют менять положение каретки для выполнения перевода строки, табуляции или возврата каретки.

Таблица 1 - Служебные символы для форматирования строк

Символ	Назначение
\n	Перевод каретки на новую строку
\b	Возврат каретки на один символ назад

Символ	Назначение
\f	Перевод каретки на новую страницу
\r	Возврат каретки на начало строки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\a	Подача звукового сигнала
\N	Идентификатор базы данных
\u, \U	16-битовый и 32-битовый символ Unicode
\x	Символ в 16-ричной системе исчисления
\o	Символ в 8-ричной системе исчисления
\0	Символ Null

Очень часто используется \n. С помощью него осуществляется в Python перенос строки. Например:

print('first\nsecond')
first
second

Функции и методы строк

Методы - это по сути функции, у которых в качестве первого аргумента выступает сам объект, метод которого вызывается.

объект.Метод()

При вызове методов необходимо помнить, что строки в Python относятся к категории неизменяемых последовательностей, то есть все функции и методы могут лишь создавать новую строку. Основные функции и методы, используемыми в Python для работы со строками представлены в таблице 2.

Таблица 2 - Функции и методы строк

Метод	Назначение	Пример
str()	Преобразует объект к строковому виду	str(25) =='25'
s2 in s	Проверка, что подстрока s2 содержится в s	'm' in 'eam'
s2 notin s	Проверка, что подстрока s2 не содержится в s	'I' notin 'team'
len(s)	Возвращает длину строки	len('abracadabra') == 11
s.find(s2) s.rfind(s2)	Возвращает индекс первого и последнего вхождения подстроки s2 в s (вернёт -1, если s2 notin s)	s = 'abracadabra' s.find('ab') == 0 s.rfind('ab') == 7 s.find('x') == -1
s.count(s2)	Возвращает количество неперекрывающихся вхождений s2 в s	'abracadabra'.count('a') == 5

s.startswith(s2) s.endswith(s2)	Проверка, что s начинается с s2 или оканчивается на s2	'abracadabra'.startswith('abra')
s.join(a)	Склеивает строки из списка a через символ s	'+'.join(['Вася', 'Маша']) == 'Вася+Маша'
s.split(s2)	Список всех слов строки s (подстрок, разделённых строками s2)	'Раз два три!'.split('а') == ['Р', 'з дв', ' три!'] 'Раз два три'.split() == ['Раз', 'два', 'три']
s.replace(s2, s3)	Строка s, в которой все неперекрывающиеся вхождения s2 заменены на s3	'Раз два три!'.replace('а', 'я') =='Ряздвя три!'
s.isdigit() s.isalpha() s.isalnum()	Проверка, что в строке s все символы - цифры, буквы (включая кириллические), цифры или буквы соответственно	'100'.isdigit() 'abc'.isalpha() 'E315'.isalnum()
s.lower() s.upper()	Строка s, в которой все буквы (включая кириллические) приведены к верхнему или нижнему регистру, т.е. заменены на строчные (маленькие) или заглавные (большие)	'Привет!'.lower() == 'привет!' 'Привет!'.upper() == 'ПРИВЕТ!'
s.title()	Переводит первую букву каждого слова в верхний регистр, а все остальные в нижний	'привет мир!'. title() == 'Привет Мир!'
s.capitalize()	Переводит первую букву в верхний регистр, а все остальные в нижний	'ПРИВЕТ!'. capitalize () == 'Привет!'
s.lstrip() s.rstrip() s.strip()	Строка s, у которой удалены символы пустого пространства (пробелы, табуляции) в начале, в конце или с обеих сторон	' Привет! '.strip() == 'Привет!'
s.ljust(k, c) s.rjust(k, c)	Добавляет справа или слева нужное количество символов c, чтобы длина s достигла k	'Привет'.ljust(8, '!') == 'Привет !!'
list(s)	Список символов из строки s	list('Привет') == ['П', 'р', 'и', 'в', 'е', 'т']