Лекция №8. Часть 1.

Тема: **Строковые выражения в языке Python.**

Тип str. Инициализация переменных типа str. Операции над строками. Стандартные методы по работе со строками.

Ключевые слова: строковый тип данных, строковый литерал, стандарт Unicode, стандарт ASCII, операции над строками, методы по работе со строками.

Keywords: str, string literal, Unicode, ASCII, string operators, string methods.

1 Тип данных str

В языке Python для представления строк вводится тип данных str. Особенности типа данных str:

- строка представляет собой неизменяемую (immutable) последовательность Unicode-символов (см. https://en.wikipedia.org/wiki/Unicode);
- строка представляет собой последовательность с произвольным доступом, т.е. допустимо обращение к отдельным символам строки. Пример

```
>>> s = "Hello world"
>>> for i in range(0, len(s)):
     print(s[i])
```

- тип данных str может вызываться как функция для создания строковых объектов. Примеры:

```
>>> # str() без аргументов – возвращает пустую строку
>>> str()
''
```

```
>>> # str() с аргументом не строкового типа –
>>> # возвращает строковое представление аргумента
>>> str(2.5)
'2.5'
```

```
>>> # str() с аргументом строкового типа –
>>> # возвращает копию аргумента
>>> str("Hello world!")
'Hello world!'
```

2 Инициализация переменных типа str

Инициализация переменной строкового типа может быть выполнена несколькими способами:

1. Инициализация с помощью литерала. Примеры:

```
>>> # Литералы строк заключаются в одинарные кавычки
>>> s = 'Hello'
>>> s
'Hello'
```

```
>>> # Литералы строк заключаются в двойные кавычки
>>> s = "Hello"
>>> s
'Hello'
```

```
>>> # Литералы строк заключаются в тройные кавычки
>>> s = '''Hello'''
>>> s
'Hello'
>>> s = """Hello"""
>>> s
'Hello'
```

```
>>> # Литералы строк можно разбивать на несколько строк
>>> s = 'Hello \
World'
>>> s
'Hello World'
```

```
>>> # В строковых литералах кавычки одного типа могут быть вложены в кавычки другого типа
>>> s = "'1', '2', '3'"
>>> s
"'1', '2', '3'"
>>> s = '"1", "2", "3"'
>>> s
'"1", "2", "3"'
```

2. Инициализация с помощью пустой строки. Пример:

```
>>> s = str()
>>> s
''
>>> s = ''
>>> s
''
```

3. Инициализация с помощью данных, полученных из потока ввода. Пример

```
>>> s = input()
Hello World!
>>> s
'Hello World!'
```

4. Инициализация с помощью выражения строкового типа

```
>>> s = 'Hello ' + 'World!'
>>> s
'Hello World!'
```

5. Инициализация с помощью предопределенной константы. В модуле string определены следующие строковые константы:

- ASCII-символы, соответствующие буквам латинского алфавита

```
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

- ASCII-символы, соответствующие буквам латинского алфавита в нижнем регистре

```
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

- ASCII-символы, соответствующие буквам латинского алфавита в верхнем регистре

```
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

- ASCII-символы, соответствующие цифрам десятичной системы счисления

```
>>> string.digits
'0123456789'
```

- ASCII-символы, соответствующие цифрам шестнадцатеричной системы счисления

```
>>> string.hexdigits
'0123456789abcdefABCDEF'
```

- ASCII-символы, соответствующие цифрам осьмеричной системы счисления

```
>>> string.octdigits
'01234567'
```

- ASCII-символы, соответствующие знакам пунктуации

```
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

- печатаемые ASCII- символы

```
>>> string.printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"
#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

- ASCII-символы, рассматриваемые как пробелы

```
>>> string.whitespace
' \t\n\r\x0b\x0c'
```

3 Операции над строками

Рассмотрим особенности операций над строками в языке Python (табл. 1).

Таблица 1 – Операции над строками

Оператор	Описание	Пример

=	Оператор присваивания	`>>> s1 = 'Hello'`
		`>>> s2 = s1`
		`>>> print(s1, s2)`
		`Hello Hello`

<, <=, >, >=, ==, !=	Операторы отношения (см. примечание 1)	`>>> 'Hello' == 'Hello'`
		`True`
		`>>> 'Hello World' > 'Hello'`
		`True`
		`>>> 'Hello World' > 'Zzzz'`
		`False`

+	Оператор конкатенации строк	`>>> s = 'Hello' + ' World!'`
		`>>> s`
		`'Hello World!'`

+=	Составной оператор конкатенации строк	`>>> s = 'Hello'`
		`>>> s += ' World'`
		`>>> s`
		`'Hello World'`

*	Оператор дублирования	`>>> s = 'Hello'`
		`>>> s = s * 3`
		`>>> s`
		`'HelloHelloHello'`

*=	Составной оператор дублирования	`>>> s = 'Hello'`
		`>>> s *= 3`
		`>>> s`
		`'HelloHelloHello'`

[]	Оператор доступа к элементам строки (см. примечание 2).	`>>> s = "Hello world"`
		`>>> for i in range(0, len(s)):`
		`print(s[i], end = '*')`
		`H*e*l*l*o* *w*o*r*l*d*`
		`>>> for i in range(-1, -1 * len(s) - 1, -1):`
		`print(s[i], end = '*')`
		`d*l*r*o*w* *o*l*l*e*H*`

[::]	Оператор извлечения среза (см. примечание 3).	`>>> word = 'Python'`
		`>>> word[:]`
		`'Python'`
		`>>> word[::]`
		`'Python'`
		`>>> word[2:]`
		`'thon'`
		`>>> word[:2]`
		`'Py'`
		`>>> word[2:5]`
		`'tho'`
		`>>> word[1:4:2]`
		`'yh'`

in	оператор проверки на вхождение – возвращает True, если подстрока присутствует в строке; в противном случае возвращает False.	`>>> 'Py' in 'Python'`
		`True`
		`>>> 'Zzz' in 'Python'`
		`False`

not in	оператор проверки на вхождение – возвращает True, если подстрока не присутствует в строке; в противном случае возвращает False.	`>>> 'Hello' not in 'Hello'`
		`False`
		`>>> 'Zzzz' not in 'Hello World'`
		`True`

is	оператор идентичности – возвращает True, если строки хранятся в памяти по одному и тому же адресу; в противном случае возвращает False.	`>>> s1 = 'Hello'`
		`>>> s2 = 'Hello'`
		`>>> s1 is s2`
		`True`
		`>>> s1 = 'Hello'`
		`>>> s2 = 'World'`
		`>>> s1 is s2`
		`False`

is not	оператор идентичности – возвращает True, если строки хранятся в памяти по разным адресам; в противном случае возвращает False.	`>>> s1 = 'Hello'`
		`>>> s2 = 'World'`
		`>>> s1 is not s2`
		`True`
		`>>> s1 = 'Hello'`
		`>>> s2 = 'Hello'`
		`>>> s1 is not s2`
		`False`

len(s)	возвращает длину строки	`>>> len('Hello')`
		`5`
		`>>> len('Hello' + 'World')`
		`10`

min(s)	возвращает символ строки с минимальным значением.	`>>> min('Hello')` `'H'`
max(s)	возвращает символ строки с максимальным значением	`>>> max('Hello')` `'o'`

Примечание 1. Строки можно сравнивать друг с другом с помощью операторов отношения. При сравнении строки рассматриваются посимвольно слева направо, при этом сравниваются коды соответствующих пар символов. Строки равны, если они имеют одинаковую длину и посимвольно эквивалентны. В строках разной длины существующий символ всегда больше соответствующего ему отсутствующего символа. Меньшей будет та строка, у которой меньше код первого несовпадающего символа (вне зависимости от максимальных и текущих длин сравниваемых строк).

Примечание 2. В языке Python нумерация символов строки начинается с нуля. Допускается использовать отрицательные индексы для доступа к элементам строки (-1 соответствует последнему символу) – в этом случае отсчет начинается с последнего символа и ведется справа налево. На рис. 1 показан пример нумерации позиции символов в строке.

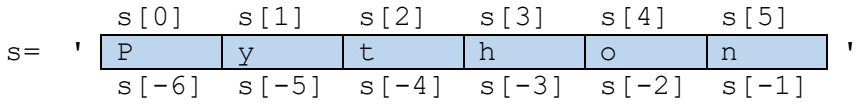

Рис. 1. Пример нумерации символов строки

Примечание 3. Оператор извлечения среза (slice) возвращает подстроку заданной строки str начиная с элемента с индексом start, заканчивая элементом с индексом stop (не включая его) и с шагом step.

Форма записи оператора

```
str[[start]: [stop] [:step]]
```

Параметры start, stop, step являются необязательными. Если опущен параметр start – то по умолчанию используется значение 0, если опущен параметр stop – то по умолчанию используется значение, равное длине исходной строки, если опущен параметр step – то по умолчанию используется значение 1. Рассмотрим примеры использования данного оператора.

```
>>> # Получение копии строки.
>>> s = 'Python'
>>> s[:]
'Python'
>>> s[::]
'Python'
```

```
>>> # Получить все элементы строки, начиная с позиции 2.
>>> s[2:]
'thon'
```

```
>>> # Получить все элементы строки, заканчивая позицией 4.
>>> s[:4]
'Pyth'
```

```
>>> # Получить последние 4 элемента.
>>> s[-4:]
'thon'
```

```
>>> # Отбросить первый и последний элемент.
>>> s[1: -1]
'ytho'
```

```
>>> # Выбрать все элементы с четными номерами.
>>> s[::2]
'Pto'
```

```
>>> # Выбрать все элементы с нечетными номерами.
>>> s[1 : : 2]
'yhn'
```

Правила обработки срезов с неверными параметрами:

- если stop больше длины строки, то stop уменьшается до длины строки;

- если start больше stop / длины строки, то возвращается пустая строка.

4 Встроенные методы по работе со строками

Рассмотрим основные методы по работе со строками (табл. 2).

Таблица 2 – Встроенные методы типа данных str

Метод	Описание	Пример

str.capitalize()	Возвращает копию строки str с первым символом в верхнем регистре.	`>>> s = 'hELLO'`
		`>>> s.capitalize()`
		`'Hello'`

str.center(width[, fillchar])	Возвращает строку str, отцентрированную в строке длиной width. Недостающие символы заполняются в соответствии с необязательным параметром fillchar (по умолчанию - пробел).	`>>> s = 'Hello'`
		`>>> s.center(10)`
		`'  Hello   '`
		`>>> s.center(10, '*')`
		`'**Hello***'`

str.ljust(width[, fillchar])	Возвращает строку str, выровненную по левому краю в строке длиной width. Недостающие символы заполняются в соответствии с необязательным параметром fillchar (по умолчанию - пробел).	`>>> s = 'Hello'`
		`>>> s.ljust(10, '*')`
		`'Hello*****'`

str.rjust(width[, fillchar])	Возвращает строку str, выровненную по правому краю в строке длиной width. Недостающие символы заполняются в соответствии с необязательным параметром fillchar (по умолчанию - пробел).	`>>> s = 'Hello'`
		`>>> s.rjust(10, '*')`
		`'*****Hello'`

str.count(sub[, start[, end]])	Возвращает число вхождений подстроки sub в строку str или в срез строки str [start:end].	`>>> s = 'Hello'`
		`>>> s.count('l')`
		`2`

str.endswith(suffix[, start[, end]])	Возвращает True, если строка str или срез строки str оканчивается подстрокой suffix; в противном случае возвращает False.	`>>> s = 'Hello'`
		`>>> s.endswith('lo')`
		`True`
		`>>> s.endswith('he')`
		`False`

str.find(sub[, start[, end]])	Возвращает позицию первого (крайнего слева) вхождения подстроки sub в строку str или в срез строки str. Если подстрока не найдена, возвращается -1.	`>>> s = 'Hello world!'`
		`>>> s.find('o')`
		`4`
		`>>> s.find('z')`
		`-1`

str.index(sub[, start[, end]])	Возвращает позицию первого (крайнего слева) вхождения подстроки sub в строку str или в срез строки str. Если подстрока не найдена, возбуждается исключение ValueError.	`>>> s = 'Hello world!'`
		`>>> s.index('o')`
		`4`
		`>>> s.index('z')`
		`Traceback (most recent call last):`
		`  File "<pyshell#22>", line 1, in <module>`
		`  s.index('z')`
		`ValueError: substring not found`

str.rfind(sub[, start[, end]])	Возвращает позицию последнего (крайнего справа) вхождения подстроки sub в строку str или в срез строки str. Если подстрока не найдена, возвращается -1.	`>>> s = 'Hello world!'`
		`>>> s.rfind('o')`
		`7`

str.rindex(sub[, start[, end]])	Возвращает позицию последнего (крайнего справа) вхождения подстроки sub в строку str или в срез строки str. Если подстрока не найдена, возбуждается исключение ValueError.	`>>> s = 'Hello world!'`
		`>>> s.rindex('o')`
		`7`

str.format(*args, **kwargs)	Возвращает копию строки str, отформатированную в соответствии с заданными аргументами.	`>>> s = '1 + 2 is {0}'`
		`>>> s.format(1+2)`
		`'1 + 2 is 3'`

str.join(iterable)	Объединяет все элементы последовательности iterable, вставляя между ними строку str (может быть пустой).	`>>> s = '*'`
		`>>>  s.join(('Hello', 'World!'))`
		`'Hello*World!'`

str.lower()	Возвращает копию строки str, в которой все символы приведены к нижнему регистру.	`>>> s = 'HELLO'`
		`>>> s.lower()`
		`'hello'`

str.upper()	Возвращает копию строки str, в которой все символы приведены к верхнему регистру.	`>>> s = 'hello'`
		`>>> s.upper()`
		`'HELLO'`

str.isalnum()	Возвращает True, если строка str не пустая и содержит только алфавитно-цифровые символы; в противном случае возвращает False.	`>>> s = 'Hello'`
		`>>> s.isalnum()`
		`True`
		`>>> s = 'Hello!'`
		`>>> s.isalnum()`
		`False`

str.isalpha()	Возвращает True, если строка str не пустая и содержит только алфавитные символы; в противном случае возвращает False.	`>>> s = 'Hello'`
		`>>> s.isalpha()`
		`True`
		`>>> s = '123'`
		`>>> s.isalpha()`
		`False`

str.isdigit()	Возвращает True, если строка str не пустая и содержит только цифры; в противном случае возвращает False.	`>>> s = '1234567890'`
		`>>> s.isdigit()`
		`True`
		`>>> s = '3.2'`
		`>>> s.isdigit()`
		`False`

str.isdecimal ()	Возвращает True, если строка str не пустая и содержит Unicode-символы категории Nd (“Number, Decimal Digit”); в противном случае возвращает False.	`>>> s = '\u00B2'`
		`>>> s`
		`'²'`
		`>>> s.isdecimal()`
		`False`

str.isnumeric()	Возвращает True, если строка str не пустая и содержит символы для обозначения чисел; в противном случае возвращает False.	`>>> s = '\u2155'`
		`>>> print(s)`
		`⅕`
		`>>> s.isnumeric()`
		`True`

str.isspace()	Возвращает True, если строка str не пустая и содержит только пробелы; в противном случае возвращает False.	`>>> s = 'Hello'`
		`>>> s.isspace()`
		`False`
		`>>> s = '  '`
		`>>> s.isspace()`
		`True`

str.lstrip([chars])	Возвращает строку str, с удаленными ведущими символами. Необязательный параметр chars задает последовательность символов для удаления (по умолчанию - пробел).	`>>> s = '  Hello'`
		`>>> s.lstrip()`
		`'Hello'`
		`>>> s = '***Hello'`
		`>>> s.lstrip('*')`
		`'Hello'`

str.replace(old, new[, count])	Возвращает строку str, в которой каждое вхождение подстроки old (но не более count, если этот параметр определен) заменяется подстрокой new.	`>>> s = 'aaa bbb aaa bbb ccc aaa'`
		`>>> s.replace('aa', 'z', 2)`
		`'za bbb za bbb ccc aaa'`

str.split(sep=None, maxsplit=-1)	Возвращает список строк, выполняя разбиение строки str не более чем maxsplit раз по подстроке sep. Если число maxsplit не задано, разбиение выполняется по всем найденным подстрокам sep. Если подстрока sep не задана, разбиение выполняется по пробельным символам.	`>>> s = 'Hello world!'`
		`>>> s.split()`
		`['Hello', 'world!']`

str.strip([chars])	Возвращает строку str, из которой удалены начальные и завершающие символы, входящие в подстроку chars. Если подстрока chars не задана, то удаляются пробельные символы.	`>>> s = '  Hello  '`
		`>>> s.strip()`
		`'Hello'`
		`>>> s = '***Hello***'`
		`>>> s.strip('*')`
		`'Hello'`

Рассмотрим простые примеры программ, использующих стандартные операторы и методы по работе со строками.

Пример №1. Написать программу, которая вставляет в заданную позицию одной строки другую строку.

```
>>> s1 = input('Первая строка: ')
Первая строка: I learn Pascal
>>> s2 = input('вторая строка: ')
вторая строка: Python
>>> n = int(input('Позиция: '))
Позиция: 8
>>> s1 = s1[: 8] + s2
>>> s1
'I learn Python'
```

Пример №2. Написать программу, которая удаляет из строки её часть с заданной позиции и заданной длины.

```
>>> s = input('Введите строку: ')
Введите строку: Hello World!
```

```
>>> n = int(input('Позиция: '))
Позиция: 5
>>> length = int(input('Длина: '))
Длина: 6
>>> s = s[0 : n] + s[n + length:]
>>> s
'Hello!'
```