В языке Python 3 строки — это последовательности символов. Для обозначения строки в программе можно использовать одну из четырёх возможных конструкций:

```
a = 'В этой строке можно использовать символ ", а обычную кавычку нужно экранировать'
b = "Здесь можно писать ', но " нужно экранировать: " "
c = '''Строка на
несколько строк (можно писать ' и " свободно)'''
```

Конечно же, строки можно также получать из других типов приведением к строке (str(s)). Строка считывается со стандартного ввода функцией input(). Для строк определена операция сложения (конкатенации), и операция умножения строки на число.

Узнать количество символов (длину строки) можно при помощи функции len:

```
>>> S = 'Hello'
>>> print(len(S))
5
```

Срез — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Есть несколько вариантов взятия срезов. Самая простая форма среза: взятие одного символа строки, а именно, S[i] — это срез, состоящий из одного символа, который имеет номер i, при этом считая, что нумерация начинается с числа 0. То есть если S='Hello', то S[0]=='H', S[1]=='e', S[2]=='l', S[3]=='l', S[4]=='o'.

Номера символов в строке называются индексами.

Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера -1. То есть S[-1]=='o', S[-2]=='l', S[-3]=='l', S[-4]=='e', S[-5]=='H'.

Если же номер символа в срезе строки S больше либо равен len(S), или меньше, чем -len(S), то при обращении к этому символу строки произойдет ошибка IndexError: string index out of range – выход за границы строки.

Срез с двумя параметрами: S[a:b] возвращает подстроку из b-a символов, начиная с символа с индексом a, то есть до символа с индексом b, не включая его. Например, S[1:4]=='ell', то же самое получится если написать S[-4:-1]. Можно использовать как положительные, так и отрицательные индексы в одном срезе, например, S[1:-1] — это строка без первого и последнего символа (срез начинается с символа с индексом 1 и заканчивается индексом -1, не включая его). При использовании такой формы среза ошибки IndexError никогда не возникает.

Если опустить второй параметр (но поставить двоеточие), то срез берется до конца строки. Например, чтобы удалить из строки первый символ (его индекс равен 0, то есть взять срез, начиная с символа с индексом 1), то можно взять срез S[1:], аналогично если опустить первый параметр, то срез берется от начала строки. То есть удалить из строки

последний символ можно при помощи среза S[:-1]. Срез S[:] совпадает с самой строкой S.

Если задать срез с тремя параметрами S[a:b:d], то третий параметр задает шаг, (аналогично range), будут взяты символы с индексами a, a+d, a+2*d и т.д. При задании значения третьего параметра, равному 2, в срез попадет каждый второй символ, а если взять отрицательное значение среза, символы будут идти в обратном порядке.

Методы строк

Метод — это функция, применяемая к объекту, в данном случае — к строке. Метод вызывается в виде Имя_объекта.Имя_метода(параметры).

Метод count

Подсчитывает количество вхождений одной строки в другую строку. Простейшая форма вызова S.count(T) возвращает число вхождений строки T внутри строки S. При этом подсчитываются только непересекающиеся вхождения, например:

```
>>> ('1' * 100000).count('11')
50000
```

Метод replace

Метод replace заменяет все вхождения одной строки на другую. Формат: S.replace(s1, s2) — заменить в строке S все вхождения подстроки s1 на подстроку s2.

Если методу replace задать еще один параметр: S.replace(s1, s2, count), то заменены будут не все вхождения, а только не больше, чем первые count из них.

```
>>> 'Abrakadabra'.replace('a', 'A', 2)
'AbrAkAdabra'
```

Метод find и rfind

Метод find находит в данной строке (к которой применяется метод) данную подстроку (которая передается в качестве параметра). Функция возвращает индекс первого вхождения искомой подстроки. Если подстрока не найдена, то метод возвращает значение -1. Аналогично, метод rfind возвращает индекс последнего вхождения данной строки (“поиск справа”).

Полный список методов с описаниями можно найти, написав в консоли команду help(str)

Довольно часто возникают ситуации, когда нужно создать строку, подставив в неё некоторые данные, полученные в процессе выполнения программы (пользовательский ввод, данные из файлов и т. д.). В этом случае для подстановки этих значений можно использовать строковый метод format. При этом в шаблоне в места подстановок нужно поставить поля вида {}, а в параметры format передать ровно необходимое количество значений. Однако возможностей у format гораздо больше: кроме непосредственной

подстановки в строку возможно применение форматирования в выводимым данным. Полный список возможностей можно получить, опять же, непосредственно в документации, здесь приведём самые популярные примеры использования этого метода:

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{0}, {1}, {2}'.format('a', 'b', 'c') # Можно явно указать индексы
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c') # Если индексы указаны явно, то они могут идти в любом порядке
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad') # И даже повторяться
'abracadabra'
>>> 'Coordinates: {lat}, {lon}'.format(lat='37.24N', lon='-115.81W') # Можно вместо индексов явно указывать имена
'Coordinates: 37.24N, -115.81W'

# Выравнивания
>>> '{:<30}'.format('left aligned')
'left aligned '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'          centered          '
>>> '{:*^30}'.format('centered') # Используем * для заполнения
'**********centered**********'

# Особый вывод действительных чисел
>>> '{:+f}; {:+f}'.format(3.14, -3.14) # Всегда показывать знак
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14) # Или выводить пробел, если знака плюса нет
' 3.140000; -3.140000'
>>> '{:0.2f}; {:0.3f}; {:8.3f}'.format(3.14, -3.14, 179e-4) # Можно выводить фиксированное кол-во значащих цифр
'3.14; -3.140;  0.018'

# Вывод в двоичной, восьмеричной или шестнадцатиричной системе счисления
>>> "int: {0:d}; hex: {0:x}; oct: {0:o}; bin: {0:b}".format(42)
'int: 42; hex: 2a; oct: 52; bin: 101010'
>>> "int: {0:d}; hex: {0:#x}; oct: {0:#o}; bin: {0:#b}".format(42) # С префиксом 0x, 0o, 0b
'int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010'
```

f-строки

Начиная с Python 3.6 появился ещё один способ получения строк, в которые вставлены значения некоторых переменных. Кроме того, что этот способ очень хорошо читается, он

ещё и самый быстрый. Начнём с примера:

```
>>> x = 10
>>> y = 5
>>> print(f"{x} x {y} / 2 = {x * y / 2}")
>>> 10 x 5 / 2 = 25.0

>>> planets = ["Меркурий", "Венера", "Земля", "Марс"]
>>> print(f"Мы живём на планете {planets[2]}")
>>> Мы живём на планете Земля
```

В целом этот подход очень похож на использование метода .format, только необходимо всегда указывать префикс f перед строкой и имена переменных внутри фигурных скобок. Возможностей того, что можно указывать внутри фигурных скобок, достаточно много (см. PEP-0498).