Урок 2.1 Строки. Срезы. Методы строк

1. Строки в Python

Строка состоит из последовательности символов:

'Текст в виде строки. Можно использовать цифры и другие символы, но не всякие.'

или так:

"1 + 2 = 3 - my text."

Тип строки str. В языке Python нет отдельного символьного типа. Символ — это просто строка длины 1.

Можно преобразовать число в строку: str(5)

In [1]:
```
# Длина строки
my_str = '1 + 2 = 3 - my text'
len(my_str)
```

Out[1]:
```
19
```

2. Индексация строк

По индексу можно получить соответствующий ему символ строки. Для этого нужно после самой строки написать в квадратных скобках индекс символа.

Обращение к элементу строки my_str[i]

In [2]:
```
word = 'привет'
print(word[0])
print(word[3])
```

```
п
в
```

Если попытаться получить букву, номера которой нет, Python выдаст ошибку

In [3]:
```
print(word[6])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [3], line 1
----> 1 print(word[6])

IndexError: string index out of range
```

В квадратных скобках можно указывать не только фиксированное число, но и считать его с клавиатуры или получить в результате арифметического действия.

In [4]:
```
word = 'привет'
print(word[int(input())])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [4], line 2
      1 word = 'привет'
----> 2 print(word[int(input())])

ValueError: invalid literal for int() with base 10: ''
```

Отрицательные индексы

В Python разрешены отрицательные индексы: word[-1] означает последний символ строки word, word[-2] — предпоследний и т д.

Важно!

Так как в Python строки - это неизменяемый тип данных, изменить отдельный символ строки невозможно:

In [5]:
```
word = 'малоко'
word[1] = 'о'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [5], line 2
      1 word = 'малоко'
----> 2 word[1] = 'о'

TypeError: 'str' object does not support item assignment
```

3. Перебор элементов строки

Оператор **in** позволяет проверить наличие подстроки в строке, а также итерировать по строке.

In [6]:
```
'3.14' in 'Число Пи = 3.1415926'
```

Out[6]:
```
True
```

In [7]:
```
text = 'Лень — главное достоинство программиста. (Larry Wall)'
vowels = 0
for i in text:
    if i in {'a', 'e', 'i', 'o', 'u', 'y'}:
        vowels += 1
print(vowels)
```

```
3
```

Так как символы в строке пронумерован, можно перебирать их с помощью for .. in ..range()

In [8]:
```
text = 'Лень — главное достоинство программиста. (Larry Wall)'
vowels = 0
for i in range(len(text)):
    if text[i] in 'aeiouy':
```

```
        vowels += 1
print(vowels)
```

```
3
```

4. Юникод. Хранение текстов в памяти компьютера

Поскольку компьютер умеет хранить только двоичные числа, для записи нечисловой информации используются кодовые таблицы - ASCII, Unicod и др.

Изначально в таблице каждому символу был поставлен в соответствие 7-битный код, что позволяло идентифицировать 128 различных символов.

Этого хватало на латинские буквы обоих регистров, знаки препинания и спецсимволы — например, перевод строки или разрыв страницы. Позже код расширили до 1 байта, что позволяло хранить уже 256 различных значений: в таблицу помещались буквы второго алфавита (например, кириллица) и дополнительные графические элементы (псевдографика).

В некоторых относительно низкоуровневых языках (например, в C) можно в любой момент перейти от представления строки в памяти к последовательности байтов, начинающейся по какому-либо адресу.

Сейчас однобайтные кодировки отошли на второй план, уступив место **Юникоду**.

Юникод содержит соответствия между числом и каким-либо знаком, причем количество знаков может быть любым. Это позволяет одновременно использовать любые символы любых алфавитов и дополнительные графические элементы. Кроме того, в Юникоде каждый символ, помимо кода, имеет некоторые свойства: например, буква это или цифра. Это позволяет более гибко работать с текстами.

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, такой:)

In [9]:
```
print('\u2603')
```

```
☃
```

Функции ord и chr

- **ord** - возвращает код символа(от order — «порядок»).
- **chr** - возвращает по коду соответствующий ему символ.

In [10]:
```
for i in range(26):
    print(chr(ord('A') + i), end=' ')
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

5. Конвертация типов

К строкам можно применить конвертацию типов. Например, мы можем преобразовать вещественное число в переменную типа str:

In [11]:
```
num = 999.01
num_string = str(num)
print(type(num_string))
num_string
```

```
<class 'str'>
```

Out[11]:
```
'999.01'
```

6. Срезы

Кроме перебора элементов с помощью цикла for ..in ..range() взять отдельный ее кусок, символы с нечетными номерами и т.д. можно с помощью срезов (slice).

Срез строки

В самом простом варианте срез строки — ее кусок от одного индекса включительно и до другого не включительно (как для range). То есть это новая, более короткая строка.

Срез записывается с помощью квадратных скобок, в которых указывается начальный и конечный индекс, разделенные двоеточием.

In [12]:
```
text = 'Hello, world!'
print(text[0:5])
print(text[7:12])
```

```
Hello
world
```

Если не указан начальный индекс, срез берется от начала (от 0). Если не указан конечный индекс, срез берется до конца строки.

In [13]:
```
text = 'Hello, world!'
print(text[:5])
print(text[7:])
```

```
Hello
world!
```

Разрешены отрицательные индексы для отсчета с конца списка.

In [14]:
```
name = 'Иванов И. И.'
name[:-6]
```

Out[14]:
```
'Иванов'
```

Как и для **range**, в параметры среза можно добавить третье число — шаг обхода. Этот параметр не является обязательным и записывается через второе двоеточие.

Шаг может быть и **отрицательным** — для прохода по строке в обратном порядке. Если в этом случае не указать начальный и конечный индекс среза, ими станут последний и первый индексы строки соответственно (а не наоборот, как при положительном шаге):

In [15]:
```
text = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
text[::-1]
```

Out[15]:
```
'ZYXWVUTSRQPONMLKJIHGFEDCBA'
```

Пример 1.

Из строки '1234567890abcdefghij' выделить каждый второй символ, начиная с 3-го (нумерация с 0) и до 17-го (не включая).

In [16]:
```
my_str = '1234567890abcdefghij'
my_str[3:17:2]
```

Out[16]:
```
'4680bdf'
```

Пример 2.

Из строки '1234567890abcdefghij' выделим все символы до 15-го (не включая).

In [17]:
```
my_str[:15]
```

Out[17]:
```
'1234567890abcde'
```

Пример 3.

Из строки '1234567890abcdefghij' выделим все символы, начиная с 15-го.

In [18]:
```
my_str[15:]
```

Out[18]:
```
'fghij'
```

Пример 4.

Из строки '1234567890abcdefghij' выделим все символы с четными номерами (нумерация с 0, считаем 0 четным числом). Приходится явно указывать знаки ':'

In [19]:
```
my_str[::2]
```

Out[19]:
```
'13579acegi'
```

Пример 5.

Выведем на экран через пробел все возможные последовательные 10 символов строки.

In [20]:
```
for i in range(len(my_str) - 9):
    print(my_str[i:i + 10], end=' ')
```

```
1234567890 234567890a 34567890ab 4567890abc 567890abcd 67890abcde 7890abcdef 890ab
cdefg 90abcdefgh 0abcdefghi abcdefghij
```

Методы строк:

https://docs.python.org/3/library/stdtypes.html#string-methods

find

ищет подстроку в строке. В случае нахождения - возвращает первое вхождение, под которым была найдена подстрока. В случае если подстрока не найдена - возвращает -1. Значение подстроки чувствительно к регистру. Можно задать облать поиска внутри строки, передав вторым и третьим аргументом значения начала среза для поиска и конца среза для поиска.

Если нужно искать от некоторой позиции до конца строки, то указывается только второй аргумент!

'This is Python. Python is my favorite language to work with'.find('is', 5, 10) ищет первое вхождение строки is в строке Python is my favorite language to work with начиная с пятого символа и до десятого, не включая десятый, возвращает номер первого символа строки is в строке, т.е. 5. Если не задавать область поиска, то получится 2. Существует и брат близнец метода find - rfind. Работает он так же, но с одним отличаем - он возвращает последнее вхождение под которым была найдена подстрока. 'This is Python. Python is my favorite language to work with'.rfind('is'). Возвращает 23.

replace

заменяет одну строку на другую. Первым параметром - то что заменяем, а вторым то на что заменяем, исходная строка не изменяется.

Если третьим аргументом передано число, то выполняется только такое число замен.

Например,

'This is Python. Python is my favorite language to work with'.replace('is', '0') возвращает 'Th0 0 Python. Python 0 my favorite language to work with', а если указать, что нужно выполнить только одну замену, т.е. 'This is Python. Python is my favorite language to work with'.replace('is', '0', 1), то получим 'Th0 is Python. Python is my favorite language to work with'.

split

разбирает строку, по указанному разделителю sep (необязательный аргумент метода split, по умолчанию пробел) и возвращает результат списком. Можно задать максимальное число разбиений.

isdigit

возвращает True, если все символы строки цифры, причем строка не пустая, иначе возвращает False.

Например, '12345'.isdigit() True, '1e'.isdigit() False.

isalpha

возвращает True, если все символы строки буквы, причем строка не пустая, иначе возвращает False.

Например, 'absd'.isalpha() True, '1e'.isalpha() False.

upper

работает с регистром строк, возвращает исходную строку в верхнем регистре, исходная строка не изменяется.

lower

работает с регистром строк, возвращает исходную строку в нижнем регистре, исходная строка не изменяется.

capitalize

переводит первый символ строки в верхний регистр, а все остальные - в нижний.

count

подсчитывает количество вхождений одной строки в другую строку. Простейшая форма вызова 'Abracadabra'.count('a') возвращает число вхождений строки 'a' внутри строки 'Abracadabra'. Подсчитывает только непересекающиеся вхождения, например: 'Abracadabra'.count('a') возвращает 4 При указании трех параметров 'Abracadabra'.count('a', 4, 10), будет выполнен подсчет числа вхождений строки 'a' в срезе [4:10] строки 'Abracadabra'

Пример 1. find

Вводится строка из букв и цифр, программа выводит на экран позиции цифр в строке.

In [21]:
```
s = input()
for i in range(10):
    print(i, ':',s.find(str(i)))
```

```
0 : -1
1 : -1
2 : -1
3 : -1
4 : -1
5 : -1
6 : -1
7 : -1
8 : -1
9 : -1
```

Программа вывела -1 для цифр, которые не встретились во введенной строке. Перепишем, чтобы отсутствующие цифры не выводились:

In [22]:
```
s = input()
for i in range(10):
    number = s.find(str(i))
    if number != -1:
        print(i, ':', number)
```

Можно заметить, что метод find для цифр 1, 2, 3 нашел только первое вхождение, так как поиск прекращается, как только найден нужный символ.

Рекомендуется использовать find только в том случае, если нужно определить позицию символа. Если требуется только узнать, есть ли такой символ в строке, нужно пользоваться in

In [23]:
```
s = input()
for i in range(10):
    if str(i) in s:
        print(i, end=' ')
```

In [ ]:
```

```

Можно искать только в части строки:

In [24]:
```
s.find('cadabra', 5, -1)
```

Out[24]:
```
-1
```

Пример 2. replace

Вводится строка из букв и цифр, программа заменяет цифры на точки и выводит на экран результат.

In [25]:
```
s = input()
for i in range(10):
    s = s.replace(str(i), '.')
print(s)
```

Пример 3. replace

Вводится строка из букв и цифр, программа заменяет первое вхождение каждой цифры на соответствующее цифре число точек и выводит на экран результат.

In [26]:
```
s = input()
for i in range(10):
    s = s.replace(str(i), '.' * i, 1)
print(s)
```

Пример 4. split

Представить в виде списка строк текст: 'One;two;three;four;five' Разделители строк ';'

In [27]:
```
'One;two;three;four;five'.split(sep=';')
```

Out[27]:
```
['One', 'two', 'three', 'four', 'five']
```

Зададим максимальное число разбиений 2.

In [28]:
```
'One;two;three;four;five'.split(sep=';', maxsplit=2)
```

Out[28]:
```
['One', 'two', 'three;four;five']
```

Пример 5. isdigit, isalpha, upper, capitalize

По строке 'One1two2three3four4five5' построим две новые строки, первая состоит только из букв строки, записанных в верхнем регистре, а вторая строка состоит из цифр.

In [29]:
```
s = 'One1two2three3four4five5'.upper()
first = ''
second = ''
for i in s:
    if i.isalpha():
        first += i
    elif i.isdigit():
        second += i
print(first, second)
```

```
ONETWOTHREEFOURFIVE 12345
```