Строки

Строка - последовательность символов произвольной длинны с произвольным доступом. Ключевым отличием строкового типа в Python от его аналогов в других языках является то, что строки в Python неизменяемые. Попытка запустить следующий код приведёт к ошибке `TypeError: 'str' object does not support item assignment` (Ошибка типа: строка не поддерживает присваивание элементов):

```
s = "Hello"
s[1] = "a"
```

Также приятной особенностью языка Python является то, что индексы могут иметь отрицательные значения для отсчёта с конца, например:

```
s = "Hello"
print(s[-1])        # на экран будет выведена буква o
print(s[-5])        # на экран будет выдедена буква H
print(s[-6])        # Ошибка индекса: индекс строки находится за пределам
```

Строки можно заключать либо в одинарные, либо в двойные кавычки, благодаря этому можно писать текст в кавычках без использования escape-последовательностей, а просто вкладывая одинарные кавычки в строку, заключенную в двойные и наоборот.

```
print("1'2'3'4'5'6")       # на экран будет выведено 1'2'3'4'5'6
print('A"B"C"D"E')         # на экран будет выведено A"B"C"D"E
```

Длинные строки можно разбивать на несколько, используя обратный слэш (\) в конце строки:

```
print('Hello, \
world!')
```

Большие тексты можно заключать в тройные кавычки, при этом обратный слэш (\) ставить уже не требуется:

```
print("""Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
```

```
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!""")
```

В Python поддерживаются и классические escape-последовательности:

- Перевод строки \n
- Табуляция \t
- Ввод символов в восьмиричной и шестнадцатиричной нотациях

Строки в Python не нуль-терминируемые, т.е. длину строки нельзя проверять по наличию нулевого байта, как это часто делается в языках семейства Си. Вместо этого существует функция len, которая возвращает длину строки:

```
s = 'Hello, world!'
print(len(s))          # на экран будет выведено число 13
```

Срезы

Срез - мощный механизм управления строкой, позволяющий получать подстроки на основе двух индексов. Обращение происходит к строке "как обычно", только вместо одного индекса указывается пара индексов, разделённых двоеточием:

```
s = 'Hello, world!'
print(s[7:12])         # на экран будет выведено подстрока world
```

Если в указании диапазона среза опущен первый индекс, то он приравнивается к нулю, если последний, то к длине строки

```
s = 'Hello, world!'
print(s[:4])           # на экран будет выведена подстрока Hell
print(s[7:])           # на экран будет выведена подстрока world!
```

Можно выбирать последовательность из строки с определённым шагом, шаг указывается после второго двоеточия, например:

```
s = 'Hello, world!'
print(s[::2])          # на экран будет выведена строка Hlo ol!
print(s[3:11:3])       # на экран будет выведена строка l r
print(s[::-1])         # на экран будет выведена строка !dlrow ,olleH
```

Форматирование строк

Форматирование строк - механизм, доступный во многих языках. В Python ему уделено особенное внимание, и он имеет расширенные

возможности в сравнении с другими языками. Для форматирования используется оператор % (не путать с оператором деления с остатком). Слева от оператора указывается строка, справа - список значений, которые необходимо подставить в строку:

```
# В переменную a будет записана строка Hello, world!
a = 'Hello, %s!' % 'world'
# В переменную b будет записана строка Hello, world!
b = 'He%s, %s!' % ('llo', 'world')
```

подстрока %s называется спецификатором, также есть другие спецификаторы, например для целых числе %d, вот полный список всех спецификаторов:

- %s - строковый

  ```
  # В переменную a будет записана строка Hi, Mark!
  a = 'Hi %s!' % 'Mark'
  ```

- %r - repr (формальное строковое представление указанного объекта)

  ```
  # В переменную b будет записана строка Hi, 'Mark'!
  b = 'Hi, %r!' % 'Mark'
  ```

- %c - символ

  ```
  # В переменную c будет записана строка Hi, Mark!
  c = 'Hi, M%cr%c!' %  ('a', 'k')
  # Ошибка (%c требует int или char)
  d = 'Hi, M%c%c!' %  ('ar', 'k')
  ```

- %d - десятичный

  ```
  # В переменную e будет записана строка 1 + 2 = 4
  e = '%d + %d = %d' %  (1, 2, 4)
  ```

- %i - целый (в Python не имеет отличия от %d, нужен для совместимости с языком C)
- %u - то же, что и %d
- %o - восмиричный

  ```
  # В переменную f будет записана строка 4 * 2 = 10
  f = '%o * %o = %o' %  (4, 2, 4*2)
  ```

- %x - шестнадцатеричный

  ```
  # В переменную g будет записана строка 9 + 1 = a
  g = '%x + %x = %x' %  (9, 1, 9+1)
  ```

- %X - шестнадцатеричный в верхнем регистре

```
# В переменную h будет записана строка C * 3 = 24
h = '%X * %X = %X' %  (12, 3, 12*3)
```

- %e - число с плавающей точкой в экспоненциальном формате в нижнем регистре

```
# В переменную i будет записана строка 7 / 2 = 3.500000e+00
i = '%d / %d = %e' %  (7, 2, 7/2)

# В переменную j будет записана строка 7 / 200000 = 3.500000e-06
j = '%d / %d = %e' %  (7, 2000000, 7/2000000)
```

- %E - то же, что и %e, только в верхнем регистре

```
# В переменную k будет записана строка 7 / 2 = 3.500000E+00
k = '%d / %d = %E' %  (7, 2, 7/2)

# В переменную l будет записана строка 7 / 200000 = 3.500000E-06
l = '%d / %d = %E' %  (7, 2000000, 7/2000000)
```

- %f - число с плавающей точкой в виде десятичной дроби

```
# В переменную m будет записана строка 7 / 2 = 3.500000
m = '%d / %d = %f' %  (7, 2, 7/2)

# В переменную n будет записана строка 7 / 2000000 = 0.000003
n = '%d / %d = %f' %  (7, 2000000, 7/2000000)
```

- %g - число с плавающей в формате %e или %f

```
# В переменную o будет записана строка 7 / 2 = 3.5
o = '%d / %d = %g' %  (7, 2, 7/2)

# В переменную p будет записана строка 7 / 2000000 = 3.5e-06
p = '%d / %d = %g' %  (7, 2000000, 7/2000000)
```

- %G - число с плавающей точкой в формате %E или %f

```
# В переменную q будет записана строка 7 / 2 = 3.5
q = '%d / %d = %G' %  (7, 2, 7/2)

# В переменную r будет записана строка 7 / 2000000 = 3.5E-06
r = '%d / %d = %G' %  (7, 2000000, 7/2000000)
```

- %% - вывести символ %

```
# В переменную s будет записана строка 10 % 3 = 1
```

```
s = '%d %% %d = %d' % (10, 3, 10%3)
```

При форматировании можно указать общую длину строки и количество символов после десятичной точки, для чисел с плавающей точкой, при этом число будет дополнено незначащими нулями:

```
# В переменную t будет записана строка 7 /     2 =  3.5000000
# Число 2 будет дополнено до строки длины 5 пробелами
# Число 3.5 будет дополнено до строки длины 10 слева пробелами,
# 7 символов из которой будут отведены на дробную часть,
# и дополнены нулями справа
t = '%d / %5d = %10.7f' % (7, 2, 7/2)

# В переменную u будет записана строка 7 / 00002 = 03.5000000
# Число 2 будет дополнено до строки длины 5 нулями слева
# Число 3.5 будет дополнено до строки длины 10 слева нулями,
# 7 символов из которой будут отведены на дробную часть,
# и дополнены нулями справа
u = '%d / %05d = %010.7f' % (7, 2, 7/2)
```

Полезные методы

1. `split(delimiter=' ')` - разбивает строку на список строк по строке-разделителю `delimiter`

   ```
   # В переменную a запишется строка 12
   # А в переменную b - строка 16
   a, b = '12 16'.split()

   # В переменную c запишется строка 12
   # А в переменную d - строка 16
   c, d = '12;16'.split(';')

   # В переменную e запишется строка 12ab16
   # А в переменную f - строка 1305acb123
   e, f = '12ab16abc1305acb123'.split('abc')
   ```

2. `join(str_list)` - создаёт строку, используя список `str_list` и текущую строку как разделитель

   ```
   # В переменную e запишется строка test;123;456
   g = ';'.join(('test', '123', '456'))
   ```

3. `find(substr)` - находит подстроку `substr` в строке и возвращает позицию первого вхождения, иначе возвращает -1

   ```
   s = 'Hello, world!'
   print(s.find('llo'))    # 2
   print(s.find('o'))      # 4
   print(s.find('abc'))    # -1
   ```

4. `replace(substr, repstr)` - находит все вхождения `substr` в строке и заменяет их на `repstr`

```
s = 'Test: this is Test!'
print(s.replace('this', 'dis'))    # Test: dis is Test!
print(s.replace('Test', 'Best'))   # Best: this is Best!
```

5. `strip()` - удаляет пробелы слева и справа от строки

```
s = '      test           TEST              '
print(s.strip())        # test           TEST
```

6. `capitalize()` - переводит первый символ строки в верхний регистр, а все остальные в нижний

```
s = 'This is my BEST test'
print(s.capitalize())   # This is my best test
```

7. `upper()` - преобразование строки к верхнему регистру

```
s = 'This is my BEST test'
print(s.upper())        # THIS IS MY BEST TEST
```

8. `lower()` - преобразование строки к нижнему регистру

```
s = 'This is my BEST test'
print(s.lower())        # this is my best test
```

9. `swapcase()` - переводит символы нижнего регистра в верхний, а верхнего – в нижний

```
s = 'This is my BEST test'
print(s.swapcase())     # tHIS IS MY best TEST
```