Подробнее о строках

Владислав Порицкий

itstep.by

20 октября 2017 г.

Что такое Юникод

- Многие популярные кодировки (например, Windows-1251) используют по одному байту на символ. Это позволяет представить $2^8 = 256$ разных символов.
- Проблема: софт, написанный под определённую кодировку текста, плохо локализуется.
- Ещё проблема: если есть N различных кодировок, нужно N^2 кодеков для преобразований между ними.
- Юникод (Unicode) – разрабатываемый с 1980-х гг. стандарт представления всех письменностей мира в едином кодовом пространстве.

Что такое Юникод

- Кодовые позиции (codepoints) Юникода – это шестнадцатеричные числа, за которыми закреплены определённые символы.
- Используются числа от `0x0000` = 0 до `0x10FFFF` = 1114111. Итого доступно > 1 млн позиций (сейчас занято ≈ 100 тысяч).
- Всё кодовое пространство разбито на 17 плоскостей (planes) по 65536 позиций:
  - плоскость 0 (`0x0000`...`0xFFFF`) – базовая;
  - плоскость 1 – дополнительная для древних письменностей;
  - плоскость 2 – дополнительная для редкой иероглифики;
  - плоскости 3...13 сейчас не используются;
  - плоскости 14...16 – дополнительные служебного назначения.

Что такое Юникод

- Формы представления Юникода: UTF-8, UTF-16 и UTF-32.
- Для записи каждой кодовой позиции используется:
  - UTF-8: 1...4 байта;
  - UTF-16: 1 или 2 двухбайтных «слова»;
  - UTF-32: ровно 1 четырёхбайтное «слово».

Строки в Python

- В Python 3 основной тип для строковых данных – `str`.
  Это последовательности кодовых позиций Юникода.
- Тип `bytes` хранит последовательности байт. Они, как и строки, неизменяемы.
- Байтовые последовательности записываются как строки с префиксом `b`.
- Можно декодировать `bytes` в `str` (метод `decode`), кодировать `str` как `bytes` (метод `encode`).
- При работе с файлами в бинарном режиме (модификатор `'b'`) читаются и пишутся объекты `bytes`, а иначе – объекты `str`.
  При этом в вызове функции `open` может быть необходимо указать кодировку.
- Тип `bytearray` – изменяемые последовательности байт.

Строки в Python

- Для сравнения, в Python 2 были два типа для строковых данных:
  - `unicode` – соответствует нынешнему `str`;
  - `str` – соответствует нынешнему `bytes`.
- Юникодовые строки записывались с префиксом `u`. (Можно использовать и в Python 3, но бессмысленно.)
- У каждого из двух типов были методы `encode` и `decode`.
- Различались функции `chr` и `unichr`.
- Был также тип `bytearray`.

Интерфейс строк

- Изменение регистра
- Проверка свойств
- Поиск и замена подстрок
- Удаление пробелов
- Добавление пробелов или других символов
- Разбиение и сцепление по разделителю
- Форматированная подстановка в «старом» и «новом» стиле
- ...а также всё, что умеют неизменяемые последовательности.

Изменение регистра

- Привести всю строку к нижнему регистру: метод `lower`.
- Привести всю строку к верхнему регистру: `upper`.
- Начальный символ в верхнем регистре, остальные в нижнем: `capitalize`.
- Первый символ каждого слова в верхнем регистре, остальные в нижнем: `title`.
- Изменить регистр каждого символа на противоположный: `swapcase`.

Проверка свойств строки

- Все ли символы в нижнем регистре: `islower`.
- Все ли символы в верхнем регистре: `isupper`.
- Верно ли, что первый символ каждого слова в верхнем регистре, остальные в нижнем: `istitle`.
- Все ли символы – буквы: `isalpha`.
- Все ли символы – цифры: `isdecimal`, `isdigit`, `isnumeric`.
  - ▷ Поведение различно для цифр-индексов, дробей, римских цифр...
- Все ли символы – буквы или цифры: `isalnum`.
- Все ли символы – пробелы: `isspace`.

Поиск и замена подстрок

- Получить позицию начала первого вхождения подстроки: `find`, `index`.
- Получить позицию начала последнего вхождения подстроки: `rfind`, `rindex`.
  - Можно указать начало и конец диапазона, в котором ищем.
  - Если не найдено, `find` возвращает `-1`, а `index` выдаёт ошибку.
- Верно ли, что строка начинается с подстроки: `startswith`.
- Верно ли, что строка заканчивается подстрокой: `endswith`.
- Подсчитать количество вхождений подстроки: `count`.
- Заменить все вхождения подстроки (или указанное количество первых вхождений) на другую подстроку: `replace`.

Поиск и замена подстрок

- Получить позицию начала первого вхождения подстроки: `find`, `index`.
- Получить позицию начала последнего вхождения подстроки: `rfind`, `rindex`.
  - Можно указать начало и конец диапазона, в котором ищем.
  - Если не найдено, `find` возвращает `-1`, а `index` выдаёт ошибку.
- Верно ли, что строка начинается с подстроки: `startswith`.
- Верно ли, что строка заканчивается подстрокой: `endswith`.
- Подсчитать количество вхождений подстроки: `count`.
- Заменить все вхождения подстроки (или указанное количество первых вхождений) на другую подстроку: `replace`.

Удаление пробелов

- Удалить пробельные символы (пробел, табуляция, новая строка, перевод каретки...) с обоих концов строки: `strip`.
- Только с правого конца: `rstrip`.
- Только с левого конца: `lstrip`.

Удаление пробелов

- Удалить пробельные символы (пробел, табуляция, новая строка, перевод каретки...) с обоих концов строки: `strip`.
- Только с правого конца: `rstrip`.
- Только с левого конца: `lstrip`.

Добавление пробелов или других символов

- Дополнить строку до указанной длины символом-заполнителем (по умолчанию пробелом) с обоих концов поровну: `center`.
- Дополнить с правого конца: `ljust`.
- Дополнить с левого конца: `rjust`.
- Дополнить слева нулями до указанной длины: `zfill`.

Разбиение и сцепление по разделителю

- Разбить по указанному разделителю (по умолчанию пробел), возвратить список строк: `split`.
  - Если разделитель не указан, пустые строки выбрасываются из полученного списка.
  - Можно указать максимальное число операций разбиения.
- То же самое справа налево: `rsplit`.
- Разбить по символу `\n`, возвратить список строк: `splitlines`.
- Разбить по первому вхождению указанного разделителя: `partition`.
- Разбить по последнему вхождению указанного разделителя: `rpartition`.
- Объединить список строк: `join` (вызывается у разделителя, по которому сцепляем).

Разбиение и сцепление по разделителю

- Разбить по указанному разделителю (по умолчанию пробел), возвратить список строк: `split`.
  - Если разделитель не указан, пустые строки выбрасываются из полученного списка.
  - Можно указать максимальное число операций разбиения.
- То же самое справа налево: `rsplit`.
- Разбить по символу `\n`, возвратить список строк: `splitlines`.
- Разбить по первому вхождению указанного разделителя: `partition`.
- Разбить по последнему вхождению указанного разделителя: `rpartition`.
- Объединить список строк: `join` (вызывается у разделителя, по которому сцепляем).

Форматированная подстановка: «старый стиль»

- Оператор % предоставляет функциональность, аналогичную `sprintf` в C:

```
s = "The answer is %d" % 42
print(s)                              # The answer is 42
```

- Оператору % передаётся строка с подстановочными знаками (слева) и кортеж подставляемых значений (справа).
- Подстановочные знаки:
  - `%s` – как строка
  - `%c` – как одиночный символ
  - `%d` – как целое число
  - `%f` – как вещественное число
  - ...

Форматированная подстановка: «старый стиль»

- Синтаксис подстановочных знаков позволяет:
  - добавлять пробелы или нули слева / справа от значения
  - отсекать часть символов
  - выбирать длину целой и дробной части при подстановке вещественных чисел
  - показывать знак числа
  - ...
- Можно передавать не кортеж, а словарь. Тогда у подстановочных знаков добавятся имена в скобках:

```
subst = {"a": "answer", "b": 42}
s = "The %(a)s is %(b)d" % subst
print(s)                              # The answer is 42
```

Форматированная подстановка: «новый стиль»

- Начиная с Python 2.6 у строк доступен метод `format` для форматированной подстановки в «новом стиле».
- Это особый микроязык, возможности которого шире, чем у оператора `%`.
- В Python 3 «новый стиль» предпочитается.
- Подробнее: https://pyformat.info.