Морфер

+7 (925) 336 9960
sales@morpher.ru

Accentizer

Accentizer — это модуль расстановки знаков ударения и точек над Ё в текстах на русском языке. Например, если на вход модуля подать предложение:

> Съешь еще немного этих мягких французских булок.

то на выходе получим:

> Съешь ещё немно́го э́тих мя́гких францу́зских бу́лок.

Если ударение можно поставить несколькими способами, то модуль выдает все возможные варианты, например: за́мок|замо́к, все|всё. Способ оформления вариантов и знак ударения можно изменить по вашему желанию.

- Приступаем к работе
- Метод Tokenizer.tokenize()
- Пользовательский словарь
- Технические характеристики

Приступаем к работе

Для начала установим модуль из дистрибутива командой:

```
pip install accentizer-0.0.22-py3-none-any.whl
```

(У вас может быть другая версия модуля.)

Вывести информацию о модуле можно командой:

```
pip show accentizer
```

Вот простейший пример расстановки ударений в тексте:

```
>>> from accentizer import Accentizer, load_standard_dictionaries
>>> accentizer = Accentizer(load_standard_dictionaries())
>>> accentizer.accentize('Привет от программы расстановки ударений!')
'Приве́т от програ́ммы расстано́вки ударе́ний!'
```

Основной класс модуля называется `Accentizer`, именно он занимается расстановкой ударений. Метод `load_standard_dictionaries` возвращает стандартный набор словарей, входящих в состав модуля. Подробнее про словари.

`Accentizer.accentize` – это самый простой метод, принимающий строку и возвращающий строку, оформленную определенным образом. Но что если вам нужно изменить это оформление (выбрать другой знак ударения или другой способ представления вариантов)? Модуль дает возможность получить подробную информацию о каждом слове входной строки: какие у него есть варианты и какие в каждом варианте есть знаки ударения и точки над Ё (например, паде́ж|паде́ж). На основании этой информации вы можете оформить вывод любым удобным для вас способом.

Вот пример форматирования текста, где ударение обозначается знаком '+' перед ударной гласной, а при наличии вариантов выбирается только первый:

```
1  source_text = "Привет, Владислав! Как дела? Как твой замок?"
2
3  annotated_tokens = list(accentizer.annotate(source_text))
4
5  text = "".join([format_token(token)
6                  if token.annotation
7                  else token.string
8                  for token in annotated_tokens])
9
10 def format_token(token):
11     return token.annotation.variants[0]  # выбрать первый вариант
12         .apply_to(
13             token.string,
14             '+',  # знак ударения
15             StressMarkPlacement.BEFORE_STRESSED_VOWEL)
```

Метод `annotate` (строка 3 выше) разбивает `source_text` на «токены» (с помощью tokenize). У каждого токена есть два атрибута:

- string – строка, например, «Как», «дела» или «? ».
- annotation – информация об ударении, содержащая атрибут variants – массив вариантов типа AnVar (annotation variant). AnVar имеет два атрибута:
  - PrimaryStressPos – положение основного ударения и
  - YoMask – информация о «точках над Ё».

  Метод AnVar.apply_to (строка 12) добавляет знаки ударения и точки над Ё к токену.

У разделителей и некоторых слов token.annotation может быть None, поэтому в строке 6 стоит проверка `if token.annotation`.

Пример вывода подробной информации в виде JSON (продолжение предыдущего примера):

```
i = 0
indexed_tokens = list([
    {
        'Token': t,
        'StartIndex': i,
        'EndIndex': (i := i + len(t.string))
    }
    for t in annotated_tokens])

stressed_tokens = list(filter(
    lambda t: t['Token'].annotation,
    indexed_tokens))

accents = list(
    [{
        'word': t['Token'].string,
        'start_index': t['StartIndex'],
        'end_index': t['EndIndex'],
        'accent_index': t['Token'].annotation.variants[0].PrimaryStressPos
    } for t in stressed_tokens])

homonyms = list([{
```

```
    'word': t['Token'].string,
    'start_index': t['StartIndex'],
    'end_index': t['EndIndex'],
    'value': list([{
        'index': v.PrimaryStressPos,
        'priority': 1  # библиотека пока не назначает приоритетов
    } for v in t['Token'].annotation.variants])
} for t in filter(
    lambda t: len(t['Token'].annotation.variants) > 1,
    stressed_tokens)])

output = {
    'text': text,
    'source_text': source_text,
    'accents': accents,
    'homonyms': homonyms
}

print(json.dumps(output, indent=2, ensure_ascii=False))
```

Вывод:

```
{
  "text": "Прив+ет, Владисл+ав! Как дел+а? Как твой з+амок?",
  "source_text": "Привет, Владислав! Как дела? Как твой замок?",
  "accents": [
    {
      "word": "Привет",
      "start_index": 0,
      "end_index": 6,
      "accent_index": 5
    },
    {
      "word": "Владислав",
      "start_index": 8,
      "end_index": 17,
      "accent_index": 8
    },
    {
      "word": "Как",
      "start_index": 19,
      "end_index": 22,
      "accent_index": 0
    },
    {
      "word": "дела",
      "start_index": 23,
      "end_index": 27,
      "accent_index": 4
    },
    {
      "word": "замок",
      "start_index": 38,
      "end_index": 43,
      "accent_index": 2
```

```
    }
  ],
  "homonyms": [
    {
      "word": "замок",
      "start_index": 38,
      "end_index": 43,
      "value": [
        {
          "index": 2,
          "priority": 1
        },
        {
          "index": 4,
          "priority": 1
        }
      ]
    }
  ]
}
```

Таким образом, различие между методами `accentize` и `annotate` в возвращаемом значении:

- accentize возвращает строку, а
- annotate возвращает список (точнее, генератор) объектов типа AnnotatedToken.

Метод Tokenizer.tokenize()

Иногда бывает удобно сделать токенизацию *до* расстановки ударений. Для этого модуль предоставляет статический метод `Tokenizer.tokenize()`:

```
>>> from accentizer import Tokenizer
>>> list(Tokenizer.tokenize('Как дела?'))
['', 'Как', ' ', 'дела', '?']
```

Метод разбивает входную строку на токены. Токены бывают двух видов: слова и разделители. Словом считается последовательность букв или цифр. Все, что между словами, – разделители. Для однообразия считается, что любая последовательность токенов начинается с разделителя. Если это не так, то в начало вставляется пустой разделитель (''). Аналогично в конце последовательности. Если рядом, без разделителей, стоят слова разных систем письменности (например, русское и сразу за ним китайское), то между ними также вставляется пустой разделитель. Таким образом, в последовательности токенов слова и разделители всегда чередуются и разделителей всегда на один больше.

Результат работы `tokenize` можно передать в методы `accentize` и `annotate` класса `Accentizer`. Вообще, эти методы в качестве параметра text могут принимать:

1. либо строку (str),
2. либо список строк,
3. либо итератор строк.

В первом случае вызывается токенизатор, а во втором и третьем используется уже готовый поток токенов. Это ваша ответственность – обеспечить, чтобы токенизация была сделана по изложенным выше правилам, поэтому рекомендуем использовать метод `Tokenizer.tokenize()`.

Пользовательский словарь

Чтобы изменить расстановку ударений в отдельных словах, можно передать классу Accentizer дополнительный словарь исключений. Вот пример словаря, который изменяет постановку ударения в слове «замок» так, что вместо двух вариантов выдается только один:

```
class UserDict(accentizer.IDictionary):
    def lookup(self, key):
        if len(key) == 1 and key[0] == 'замок':
            anvar = accentizer.AnVar(primary_stress_pos=2, yo_mask=0)
            variants = accentizer.Variant(anvar, accentizer.Cases.Nom, False),
            return accentizer.Word(variants),

    def get_max_key_len(self):
        return 1
```

Разумеется, писать if на каждое слово не нужно. Вы можете написать свой класс, который читает список исключений из файла или базы данных.

Метод `IDictionary.get_max_key_len` возвращает максимальную длину ключа в вашем словаре (в словах).

Вот как передать ваш словарь объекту Accentizer:

```
>>> user_dicts = UserDict(),
>>> standard_dicts = load_standard_dictionaries()
>>> accentizer = Accentizer(user_dicts + standard_dicts)
>>> accentizer.accentize('замок')
'за́мок'
```

Порядок перечисления словарей важен: Accentizer возвращает информацию о слове из первого словаря, в котором оно найдено. Так что если поменять местами `user_dicts` и `standard_dicts`, информация о слове «замок» возьмется из стандартных словарей.

Технические характеристики

- Модуль написан на чистом Питоне (3.6+).
- Модуль не обращается к внешним сервисам и базам данных.
- Из зависимостей всего одна библиотека: unicodeblock (есть в PyPI).
- Весь код модуля потокобезопасен (thread-safe).
- Размер дистрибутива (whl): < 1400 KB.
- Потребление RAM: около 95 MB (один объект Accentizer со стандартными словарями).
- Вызов метода accentize для предложения из 8 слов занимает в среднем менее одной миллисекунды на процессоре i7-9750H:

  ```
  def test_timeit(self):
      n = 10000
      s = 'Иванов Иван Иванович умеет быстро ставить ударения в текстах'
      ms = timeit.timeit(lambda: self.accentizer.accentize(s), number=n) * 1000
      ms_per_call = ms / n
      print(ms_per_call, 'ms per call')
      self.assertLess(ms_per_call, 1)
  ```

  Вывод:

  ```
  0.25173318999999994 ms per call
  Ran 1 test in 2.523s
  OK
  ```

Веб-сервис
.NET
1C
PHP
Java
Drupal
Локальный
HTTP API

Библиотеки
.NET
1C
PHP
Java
Delphi
SQL Server
Excel
IIS

О нас
Новости
Отзывы
Контакты
Facebook
FAQ

Программы обработки текстов
Программа расстановки ударений
Программа генерации .docx по шаблону
Программа образования прилагательных от названий городов и стран

© Сергей Слепов, 2003 - 2021.