



ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Модуль codecs в Python, реестр кодеков

 hh.ru РЕКЛАМА · 16+
0+



hh.ru для малого бизнеса: разместите вакансию

Более 50 млн резюме в нашей базе. Ищите помощников для своего бизнеса на hh.ru

Зарегистрироваться

[Стандартная библиотека Python3.](#) / Модуль codecs в Python, реестр кодеков

Реестр кодеков и базовые классы

[Модуль codecs](#) определяет базовые классы для стандартных кодеков Python (кодировщик и декодировщик) и предоставляет доступ к внутреннему реестру кодеков Python, который управляет процессом поиска кодека и обработки ошибок.

Большинство стандартных кодеков - это текстовые кодировки, которые кодируют [текст](#) в [байты](#), но есть также кодеки, которые кодируют текст в текст и байты в байты. В Python уже имеется большое количество стандартных кодеков и маловероятно, что приложению потребуется определять собственный кодировщик или декодировщик. Но иногда необходимо сделать свой кодек, для этих целей [модуль codecs](#), предоставляет несколько базовых классов, чтобы упростить процесс.

Допустим надо сделать кодек, который будет инвертировать заглавные буквы в строчные, а строчные - в заглавные. Можно пойти простым путем и определить функцию кодирования, которая будет выполнять это преобразование для входной строки

```
import string

def invert(text):
    """Возвращает новую строку с преобразованными
    заглавными буквами в строчные, а строчные - в заглавные
    """
    return ''.join(
        c.upper() if c in string.ascii_lowercase
        else c.lower() if c in string.ascii_uppercase
        else c
        for c in text
    )

>>> invert('Hello World')
'hELLO wORLD'
>>> invert('hELLO wORLD')
'Hello World'
```

Такая реализация неэффективна, особенно для очень больших текстовых строк. Модуль codecs включает в себя некоторые вспомогательные функции для создания кодеков, основанных на карте символов. Карта символов состоит из двух словарей. Карта преобразует символьные значения из входной строки в байтовые значения на выходе, а карта декодирования - в обратном направлении. Сначала создадим карту декодирования, а затем с помощью функции `make_encoding_map()` преобразуйте ее в карту кодирования.

```
# encoding_decoding_map.py

import codecs
import string



# сопоставить символы самим себе
decoding_map = codecs.make_identity_dict(range(256))

# Вверх пар порядковых номеров соответствий
# строчной букв - заглавной
```

```

pairs = list(zip(
    [ord(c) for c in string.ascii_lowercase],
    [ord(c) for c in string.ascii_uppercase],
))

#
#
def
})
def
})
#
en

РЕКЛАМА



ВСТРЕЧАЙТЕ НОВЫЕ  
АВТОМОБИЛИ SOLLERS



Многообразие модификаций



Оставить заявку


```

В данном случае карты кодирования и декодирования символов одинаковы, это может быть не всегда так. Функция `make_encoding_map()` обнаруживает ситуации, когда более одного входного символа кодируется в один и тот же выходной байт и заменяет значение кодирования на `None`, чтобы пометить кодировку как неопределенную.

Созданные карты символов кодирования и декодирования поддерживают все [стандартные обработчики ошибок](#), поэтому для соответствия этой части API не требуется никакой дополнительной работы.

Для создания полноценного кодека необходимо установить несколько дополнительных классов и зарегистрировать кодировщик и декодировщик. [Функция `codecs.register\(\)`](#) добавляет функцию поиска в реестр кодеков, чтобы ее можно было найти. Функция поиска должна принимать один строковый аргумент с именем кодировки и возвращать [объект `codecs.CodecInfo\(\)`](#), если кодировка найдена или `None` если нет.

Экземпляр CodecInfo, возвращаемый функцией поиска, сообщает кодекам, как кодировать и декодировать, используя все различные поддерживаемые механизмы: [кодирование и декодирование без сохранения состояния](#), [инкрементное кодирование и декодирование](#) и [потокковое кодирование и декодирование](#).

```
from encoding_decoding_map import encoding_map, decoding_map
import codecs

class InvertCodec(codecs.Codec):
    "кодирование/декодирование без сохранения состояния"

    def encode(self, input, errors='strict'):
        return codecs.charmap_encode(input, errors, encoding_map)

    def decode(self, input, errors='strict'):
        return codecs.charmap_decode(input, errors, decoding_map)

class InvertIncrementalEncoder(codecs.IncrementalEncoder):
    def encode(self, input, final=False):
        data, nbytes = codecs.charmap_encode(input,
                                              self.errors, encoding_map)
        return data

class InvertIncrementalDecoder(codecs.IncrementalDecoder):
    def decode(self, input, final=False):
        data, nbytes = codecs.charmap_decode(input,
                                              self.errors, decoding_map)
        return data

class InvertStreamReader(InvertCodec, codecs.StreamReader):
    pass

class InvertStreamWriter(InvertCodec, codecs.StreamWriter):
```

```
def find_invert(encoding):  
    """Return the codec for 'invert'.  
    """
```



```
fo(  
    c().encode,  
    c().decode,  
    r=InvertIncrementalEncoder,  
    r=InvertIncrementalDecoder,  
    rtStreamReader,  
    rtStreamWriter,
```

COO

Многообразие модификаций

без сохранения состояния переопределяем `Codec.encode()` и `Codec.decode()` с новой реализацией, `encode()` и `charmap_decode()` соответственно. Каждый метод должен возвращать кортеж, содержащий количество использованных входных байтов или символов.

[IncrementalEncoder и IncrementalDecoder](#) служат в качестве базовых классов для дополнительных интерфейсов. Методы `encode()` и `decode()` инкрементных классов определены таким образом, что они возвращают только фактические преобразованные данные. Любая информация о буферизации сохраняется как внутреннее состояние. Созданный кодек `invert` не требует буферизации данных, т.к. использует взаимно однозначное отображение.

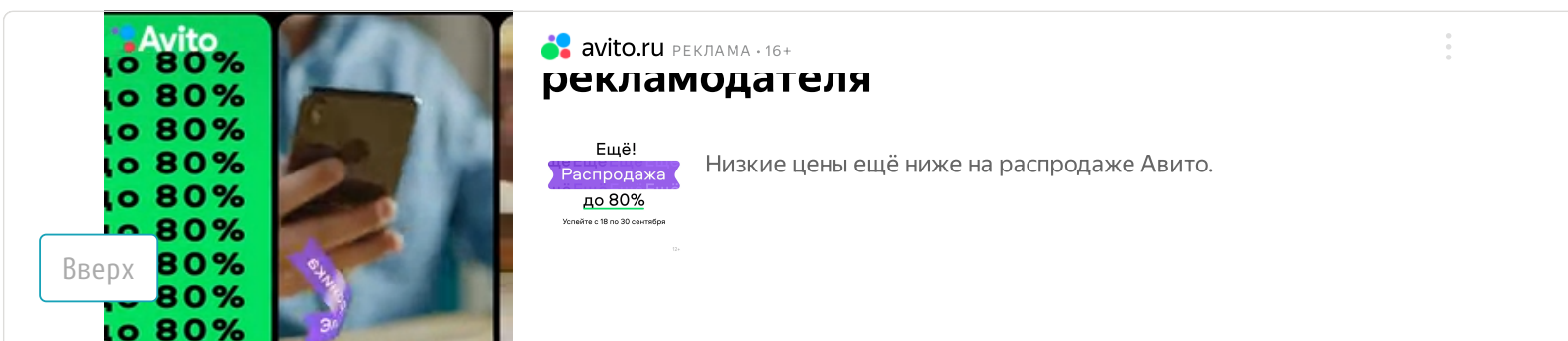
[StreamReader](#) и [StreamWriter](#) также нуждаются в методах `encode()` и `decode()` т. к. они должны возвращать то же значение, что и версия из кодека, для реализации может использоваться множественное наследование.

```
# Stateless encoder/decoder
encoder = codecs.getencoder('invert')
text = 'Hello Word'
encoded_text, consumed = encoder(text)
print('Encoded "{}" to "{}", consuming {} characters'.format(
    text, encoded_text, consumed))


# Stream writer
import io
buffer = io.BytesIO()
writer = codecs.getwriter('invert')(buffer)
print('StreamWriter for io buffer: ')
print('  writing "abcDEF"')
writer.write('abcDEF')
print('  buffer contents: ', buffer.getvalue())

# Incremental decoder
decoder_factory = codecs.getincrementaldecoder('invert')
decoder = decoder_factory()
decoded_text_parts = []
for c in encoded_text:
    decoded_text_parts.append(
        decoder.decode(bytes([c]), final=False)
    )
decoded_text_parts.append(decoder.decode(b'', final=True))
decoded_text = ''.join(decoded_text_parts)
print('IncrementalDecoder converted {}!r to {}!r'.format(
    encoded_text, decoded_text))

# Encoded "Hello Word" to "b'hELLO wORLD'", consuming 6 characters
# StreamWriter for io buffer:
#   writing "Hello Word"
#   buffer contents:  b'hELLO wORLD'
# IncrementalDecoder converted b'hELLO wORLD' to 'Hello Word'
```



[illegible]

<div data-bbox="136 442 520 629"></div> <div data-bbox="136 629 520 905"><p>ВСТРЕЧАЙТЕ НОВЫЕ АВТОМОБИЛИ SOLLERS</p><p>Многообразие модификаций</p><p>Оставить заявку</p></div>	<div data-bbox="529 442 1999 581"><p>Содержание раздела:</p></div> <div data-bbox="529 581 1999 1121"><ul style="list-style-type: none">• дирования с любой кодировкой• е• кодировки• ии функции поиска кодеров• Регистрация нового обработчика ошибок• Интерфейс базового класса Codec модуля codecs• Интерфейсы IncrementalEncoder() и IncrementalDecoder()• Потоковое кодирование и декодирование</div>
---	---