определяют символ, находящийся в одной из 256 многоязычных плоскостей Unicode. Первые два числа являются номером плоскости (от 00 до FF), а следующие два — индексом символа внутри плоскости. Плоскость с номером 00 — это старый добрый формат ASCII, и позиции символов в нем такие же, как и в ASCII.

- Для символов более высоких плоскостей нужно больше битов. Управляющая последовательность для них выглядит как \U, за которым следуют восемь шестнадцатеричных символов, крайний слева из них должен быть равен 0.
- Для всех символов конструкция \N{ имя } позволяет указать символ с помощью его стандартного имени.

Модуль unicodedata содержит функции, которые преобразуют символы в обоих направлениях:

- lookup() принимает не зависящее от регистра имя и возвращает символ Unicode;
- name() принимает символ Unicode и возвращает его имя в верхнем регистре.

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

В следующем примере мы напишем проверочную функцию, которая принимает символ Unicode, ищет его имя, а затем ищет символ, соответствующий полученному имени (он должен совпасть с оригинальным):

```
>>> def unicode_test(value):
...     import unicodedata
...     name = unicodedata.name(value)
...     value2 = unicodedata.lookup(name)
...     print('value="%s", name="%s", value2="%s"' % (value, name, value2))
...
```

Попробуем проверить несколько символов, начиная с простой буквы формата ASCII:

```
>>> unicode_test('A')
value="A", name="LATIN CAPITAL LETTER A", value2="A"
```

Знак препинания, доступный в ASCII:

```
>>> unicode_test('$')
value="$", name="DOLLAR SIGN", value2="$"
```

Символ валюты из Unicode:

```
>>> unicode_test('\u00a2')
value="¢", name="CENT SIGN", value2="¢"
```

Еще один символ валюты из Unicode:

```
>>> unicode_test('\u20ac')
value="€", name="EURO SIGN", value2="€"
```

Единственная проблема, с которой вы можете столкнуться, — это ограничения, накладываемые шрифтом. Ни в одном шрифте нет символов для всех символов Unicode,

вместо них будет отображен символ-заполнитель. Например, так выглядит символ Unicode SNOWMAN, содержащийся в пиктографических шрифтах

```
>>> unicode_test('\u2603')
value="☃", name="SNOWMAN", value2="☃"
```

Предположим, мы хотим сохранить в строке слово caf'e. Одно из решений состоит в том, чтобы скопировать его из файла или с сайта и понадеяться, что это сработает:

```
>>> place = 'café'
>>> place
'café'
```

Это сработало, поскольку я скопировал это слово из источника, использующего кодировку UTF-8 (с которой вы познакомитесь далее), и вставил его.

Как же нам указать, что последний символ — это «'e»? Если вы посмотрите на индекс символа «E», вы увидите, что имя E WITH ACUTE, LATIN SMALL LETTER имеет индекс 00E9. Рассмотрим функции name() и lookup(), с которыми мы только что работали.

Сначала передадим код символа, чтобы получить его имя:

```
>>> unicodedata.name('\u00e9')
'LATIN SMALL LETTER E WITH ACUTE'
```

Теперь найдем код для заданного имени:

```
>>> unicodedata.lookup('E WITH ACUTE, LATIN SMALL LETTER')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: "undefined character name 'E WITH ACUTE, LATIN SMALL LETTER'"
```

> Примечание:
> Имена, перечисленные в списке Unicode Character Name Index, были переформатированы для удобства отображения. Для того чтобы преобразовать их в настоящие имена символов Unicode (которые используются в Python), удалите запятую и переместите ту часть имени, которая находится после нее, в самое начало. Соответственно, в нашем примере E WITH ACUTE, LATIN SMALL LETTER нужно изменить на LATIN SMALL LETTER E WITH ACUTE:
>
> ```
> >>> unicodedata.lookup('LATIN SMALL LETTER E WITH ACUTE')
> 'é'
> ```

Теперь мы можем использовать символ «'e» как с помощью кода, так и с помощью имени:

```
>>> place = 'caf\u00e9'
>>> place
'café'
>>> place = 'caf\N{LATIN SMALL LETTER E WITH ACUTE}'
>>> place
'café'
```

В предыдущем сниппете вы вставили символ «'e» непосредственно в строку, но мы также можем собрать строку из составляющих:

```
>>> u_umlaut = '\N{LATIN SMALL LETTER U WITH DIAERESIS}'
>>> u_umlaut
'ú'
>>> drink = 'Gew' + u_umlaut + 'rztraminer'
>>> print('Now I can finally have my', drink, 'in a', place)
Now I can finally have my Gewúrztraminer in a café
```

Строковая функция len считает количество символов в кодировке Unicode, а не байты:

```
>>> len('$')
1
>>> len('\U0001f47b')
1
```

Кодирование и декодирование с помощью кодировки UTF-8

Вам не нужно волноваться о том, как Python хранит каждый символ Unicode, ко гда вы выполняете обычную обработку строки.Но когда вы обмениваетесь данными с внешним миром, вам может понадобиться следующее:

- способ закодировать строку с помощью байтов;
- способ декодировать байты обратно в строку.

Если бы в Unicode было менее 64 000 символов, мы могли бы хранить ID каждого из них в двух байтах. К сожалению, символов больше. Мы могли бы кодировать каждый ID с помощью трех или четырех байтов, но это увеличило бы объем памяти и дискового пространства, необходимый для обычных текстовых строк, в три или четыре раза.

Кен Томпсон (Ken Thompson) и Роб Пайк (Rob Pike), чьи имена будут знакомы разработчикам на Unix, разработали UTF-8 — динамическую схему кодирования — однажды вечером на салфетке в одной из столовых Нью-Джерси. Она использует для символа Unicode от одного до четырех байтов:

- один байт для ASCII;
- два байта для большинства языков, основанных на латинице (но не кириллице);
- три байта для остальных простых языков;
- четыре байта для остальных языков, включая некоторые азиатские языки и символы.

UTF-8 — это стандартная текстовая кодировка для Python, Linux и HTML. Она охватывает множество символов, работает быстро и хорошо. Если вы используете кодировку UTF-8 в своем коде, жизнь станет гораздо проще, чем в том случае, если будете скакать от одной кодировки к другой.

> Примечание
> Если вы создаете строку Python путем копирования символов из другого источника вроде веб-страницы и их вставки, убедитесь, что источник был закодирован с помощью UTF-8. Очень часто может оказаться, что текст был зашифрован с помощью кодировок Latin-1 или Windows 1252, что при копировании в строку Python вызовет генерацию исключений из-за некорректной последовательности байтов.

Кодирование

Вы кодируете строку байтами. Первый аргумент строковой функции encode() — это имя кодировки. Возможные варианты представлены в табл. 10.1.

Таблица 10.1. Кодировки

ascii	Старая добрая семибитная кодировка ASCII
utf-8	Восьмибитная кодировка переменной длины, самый предпочтительный вариант в большинстве случаев
latin-1	Также известна как ISO 8859-1
cp-1252	Стандартная кодировка Windows
unicode-escape	Буквенный формат Python Unicode, выглядит как \uxxxx или \Uxxxxxxxx

С помощью кодировки UTF-8 вы можете закодировать все что угодно. Присвоим строку Unicode '\u2603' переменной snowman:

```
>>> snowman = '\u2603'
```

snowman — это строка Python Unicode, содержащая один символ независимо от того, сколько байтов может потребоваться для того, чтобы сохранить ее:

```
>>> len(snowman)
1
```

Теперь закодируем этот символ последовательностью байтов:

```
>>> ds = snowman.encode('utf-8')
```

Как я упоминал ранее, кодировка UTF-8 имеет переменную длину. В этом случае было использовано три байта для того, чтобы закодировать один символ snowman:

```
>>> len(ds)
3
>>> ds
b'\xe2\x98\x83'
```

Функция len() возвращает число байтов (3), поскольку ds является переменной bytes.

Вы можете использовать другие кодировки, не только UTF-8, но будете получать ошибки, если строка Unicode не сможет быть обработана другой кодировкой. Например, если вы используете кодировку ascii, у вас ничего не выйдет, если только вы не предоставите строку, состоящую из корректных символов ASCII:

```
>>> ds = snowman.encode('ascii')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character '\u2603'
in position 0: ordinal not in range(128)
```

Функция encode() принимает второй аргумент, который помогает вам избежать возникновения исключений, связанных с кодировкой. Его значение по умолчанию, как вы можете увидеть в предыдущем примере, равно 'strict'; при таком значениинаблюдается исключение UnicodeEncodeError, если встречается символ, не входящий в кодировку ASCII. Существуют и другие кодировки. Используйте значение 'ignore', чтобы опустить все, что не может быть закодировано:

```
>>> snowman.encode('ascii', 'ignore')
b''
```

Используйте значение 'replace', чтобы заменить неизвестные символы символами ?:

```
>>> snowman.encode('ascii', 'replace')
b'?'
```

Используйте значение 'backslashreplace', чтобы создать строку, содержащую символы PythonUnicode вроде unicode-escape:

```
>>> snowman.encode('ascii', 'backslashreplace')
b'\\u2603'
```

Вы можете использовать этот вариант, если вам нужна печатаемая версия управляющей последовательности Unicode.

В следующем примере создаются строки символьных сущностей, которые вы можете встретить на веб-страницах:

```
>>> snowman.encode('ascii', 'xmlcharrefreplace')
b'&#9731;'
```

Декодирование

Мы декодируем байтовые строки в строки Unicode. Когда мы получаем текст из какого-то внешнего источника (файлы, базы данных, сайты, сетевые API и т. д.), он закодирован в виде байтовой строки. Идея заключается в том, чтобы знать, какая кодировка была использована, чтобы мы могли ее декодировать и получить строку Unicode.

Проблема в следующем: никакая часть байтовой строки не говорит нам о том, какая была использована кодировка. Я уже упоминал опасности копирования/вставки с сайтов. Вы, возможно, посещали сайты, содержащие странные символы в том месте, где должны быть простые символы ASCII.

Создадим строку Unicode, которая называется place и имеет значение 'cafè':

```
>>> place = 'caf\u00e9'
>>> place
'café'
>>> type(place)
<class 'str'>
```

Закодируем ее в формат UTF-8 с помощью переменной bytes, которая называется place_bytes:

```
>>> place_bytes = place.encode('utf-8')
>>> place_bytes
b'caf\xc3\xa9'
>>> type(place_bytes)
<class 'bytes'>
```

Обратите внимание на то, что переменная place_bytes содержит пять байтов. Первые три похожи на ASCII (преимущество UTF-8), а последние два кодируют символ «'e». Теперь декодируем эту байтовую строку обратно в строку Unicode:

```
>>> place2 = place_bytes.decode('utf-8')
>>> place2
'café'
```

Это сработало, поскольку мы закодировали и декодировали строку с помощью кодировки UTF-8. Что, если бы мы указали декодировать ее с помощью какой-нибудь другой кодировки?

```
>>> place3 = place_bytes.decode('ascii')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 3:
ordinal not in range(128)
```

Декодер ASCII сгенерировал исключение, поскольку байтовое значение 0xc3 некорректно в ASCII. Существуют и другие восьмибитные кодировки, где значения между 128 (80 в шестнадцатеричной системе) и 255 (FF в шестнадцатеричной системе) корректны, но не совпадают со значениями UTF-8:

```
>>> place4 = place_bytes.decode('latin-1')
>>> place4
'cafÃ©'
>>> place5 = place_bytes.decode('windows-1252')
>>> place5
'cafÃ©'
```

Используйте кодировку UTF-8 всюду, где это возможно. Она работает, она поддерживается везде, вы можете с ее помощью выразить любой символ Unicode и быстро закодировать и декодировать.

Контрольные вопросы

1. Какие форматы кодировок. вы знаете?
2. Строки формата Unicode в Python 3.
3. Модуль unicodedata.
4. Кодирование и декодирование с помощью кодировки UTF-8.
5. Определение кода символа.

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.
3. Шкляр, М. Ф. Основы научных исследований : учеб. пособие / М.Ф. Шкляр. - 6-е изд. - М. : Дашков и Ко, 2016. - 208 с.
4. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб. пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.