Глава 15

Интернационализация и локализация приложений

Python поставляется с двумя модулями, позволяющими подготовить приложение к использованию нескольких языков и региональных настроек. Модуль `gettext` (раздел 15.1) обеспечивает отображение командных подсказок и сообщений об ошибках на языке, понятном пользователю, за счет создания каталогов с сообщениями, переведенными на различные языки. Модуль `locale` (раздел 15.2) изменяет форматирование чисел, денежных единиц, значений даты и времени с учетом таких региональных стандартов, как обозначение отрицательных значений или символ местной денежной единицы. Оба модуля взаимодействуют с другими инструментами и рабочей средой, обеспечивая согласованность приложения на языке Python с остальными программами, установленными в данной системе.

15.1. `gettext`: каталоги сообщений

Модуль `gettext` предоставляет совместимую с библиотекой GNU gettext реализацию, распознающую только код на языке Python, с помощью которой можно управлять сообщениями, переведенными на другие языки, и каталогами, в которых они содержатся. Инструменты, доступные в исходном дистрибутиве Python, позволяют извлекать сообщения из набора исходных файлов, создавать каталоги, содержащие переводы сообщений, и использовать их на этапе выполнения для отображения информации на языке, понятном пользователю.

Каталоги сообщений можно также использовать для настройки других параметров, в том числе для создания интерфейсов-оберток.

Примечание

Несмотря на то что в документации стандартной библиотеки утверждается, что Python включает все необходимые инструменты, программа `pygettext.py` неспособна извлекать сообщения, обернутые вызовом функции `ngettext()`, даже с использованием соответствующих параметров командной строки. В приведенных в этом разделе примерах вместо программы `pygettext.py` используется утилита `xgettext` из набора инструментов GNU gettext.

15.1.1. Обзор процесса перевода сообщений

Процесс настройки и использования сообщений, переведенных на различные языки, включает следующие пять этапов.

1. *Идентификация и маркирование в исходном коде тех литеральных строк, которые содержат сообщения, подлежащие переводу на другие языки.*

Начните с идентификации в исходном коде сообщений, которые должны быть переведены на другой язык, и пометьте соответствующие литеральные строки, чтобы программа, предназначенная для их извлечения, могла их найти.

2. *Извлечение сообщений.*

   Идентифицировав подлежащие переводу строки, используйте программу `xgettext` для их извлечения и создайте *шаблон перевода* в виде *.pot*-файла. Шаблон перевода — это текстовый файл, содержащий копии всех идентифицированных строк и поля, в которые должны быть вставлены переводы сообщений.

3. *Перевод сообщений*

   Передайте переводчику копию *.pot*-файла, изменив его расширение на *.po*. Файл с расширением *.po* — это редактируемый файл, используемый в качестве входного на этапе компиляции. Переводчик должен обновить текст заголовка в файле и предоставить перевод для каждой строки.

4. *Компиляция каталога переведенных сообщений.*

   Получив от переводчика готовый текстовый *.po*-файл, скомпилируйте его в двоичный формат каталога с помощью функции `msgfmt`. Двоичный формат используется при поиске подстановочных значений в каталоге во время выполнения.

5. *Загрузка и активизация подходящего каталога сообщений во время выполнения.*

   Окончательным шагом является добавление в программу нескольких строк, обеспечивающих конфигурирование и загрузку каталога сообщений, а также настройку функции `translation()`. Это можно сделать несколькими способами, однако все они требуют компромиссных решений.

В оставшейся части раздела мы подробно рассмотрим каждый из этих этапов по отдельности, начав с внесения в код необходимых изменений.

15.1.2. Создание каталога сообщений на основе исходного кода

Работа модуля `gettext` заключается в поиске литеральных строк, содержащихся в базе данных, и извлечении соответствующего перевода для каждой строки. Обычным приемом является связывание поисковой функции с именем _ (одиночный символ подчеркивания), позволяющим избавиться от загромождения кода множеством вызовов функций с длинными именами.

Программа для извлечения сообщений, `xgettext`, ищет сообщения, которые встроены в вызовы функций, выполняющих поиск в каталоге. Она распознает различные исходные языки и использует для каждого из них соответствующий парсер. При наличии поисковых функций, имеющих псевдонимы (а также в случае добавления новых функций), необходимо передать программе `xgettext` имена дополнительных символов, которые следует учитывать при извлечении сообщений.

В представленном ниже сценарии имеется одно сообщение, подготовленное к переводу.

Листинг 15.1. gettext_example.py

```
t = gettext.translation(
    'example_domain', 'locale',
    fallback=True,
)
_ = t.gettext

print(_('This message is in the script.'))
```

Текст "This message is in the script." — это сообщение, которое должно быть заменено соответствующим переводом из каталога сообщений. Включение резервного режима (fallback) обеспечивает вывод встроенного сообщения, если при выполнении сценария каталог сообщений окажется недоступным.

```
$ python3 gettext_example.py

This message is in the script.
```

Следующим шагом является извлечение сообщения и создание *.pot*-файла с помощью программы pygettext.py или xgettext.

```
$ xgettext -o example.pot gettext_example.py
```

Содержимое результирующего файла приведено ниже.

Листинг 15.2. example.pot

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE
package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2016-07-10 10:45-0400\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: gettext_example.py:19
msgid "This message is in the script."
msgstr ""
```

Каталоги сообщений создаются в каталогах, структурированных по признакам *домена* и *языка*. Домен предоставляется приложением или библиотекой, и обычно ему соответствует какое-либо уникальное значение, например имя приложения. В сценарии `gettext_example.py` домен определяется значением `example_domain`. Значение параметра, определяющего язык, предоставляется пользовательской средой времени выполнения через одну из переменных среды `LANGUAGE`, `LC_ALL`, `LC_MESSAGES` или `LANG`, в зависимости от конфигурации и платформы. Все примеры, приведенные в этом разделе, выполнялись с использованием значения `en_US` для параметра языка.

Следующим после подготовки шаблона шагом является создание требуемой структуры каталогов и копирование шаблона в соответствующее расположение. Во всех приведенных ниже примерах в качестве корневого каталога сообщений используется каталог `locale`, принадлежащий дереву каталогов PyMOTW, но, как правило, для этих целей лучше использовать каталог, к которому можно обращаться из любого расположения в системе, чтобы доступ к нему имели все пользователи. Полный путь к исходному файлу каталога сообщений имеет следующий вид: `$localedir/$language/LC_MESSAGES/$domain.po`, а фактическим каталогом сообщений является файл с расширением *.mo*.

Чтобы создать каталог, следует скопировать файл шаблона `example.pot` в расположение `locale/en_US/LC_MESSAGES/example.po` и внести в него соответствующие исправления, изменив значения в заголовке и задав альтернативные варианты сообщений (листинг 15.3).

Листинг 15.3. `locale/en_US/LC_MESSAGES/example.po`

```
# Messages from gettext_example.py.
# Copyright (C) 2009 Doug Hellmann
# Doug Hellmann <doug@doughellmann.com>, 2016.
#
msgid ""
msgstr ""
"Project-Id-Version: PyMOTW-3\n"
"Report-Msgid-Bugs-To: Doug Hellmann <doug@doughellmann.com>\n"
"POT-Creation-Date: 2016-01-24 13:04-0500\n"
"PO-Revision-Date: 2016-01-24 13:04-0500\n"
"Last-Translator: Doug Hellmann <doug@doughellmann.com>\n"
"Language-Team: US English <doug@doughellmann.com>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: gettext_example.py:16
msgid "This message is in the script."
msgstr "This message is in the en_US catalog."
```

Каталог создается на основе *.po*-файла с помощью утилиты `msgfmt`:

```
$ cd locale/en_US/LC_MESSAGES; msgfmt -o example.mo example.po
```

В качестве домена в сценарии gettext_example.py используется example_domain, однако файл называется example.pot. Чтобы модуль gettext мог найти соответствующий файл с переводом сообщения, эти имена должны совпадать.

Листинг 15.4. gettext_example_corrected.py

```
t = gettext.translation(
    'example', 'locale',
    fallback=True,
)
```

Теперь при запуске сценария будет выводиться не встроенная строка, а сообщение из каталога.

```
$ python3 gettext_example_corrected.py

This message is in the en_US catalog.
```

15.1.3. Поиск каталогов сообщений во время выполнения

Как уже отмечалось, *каталог локалей*, содержащий каталоги сообщений, организуется на основе поддерживаемых языков, тогда как имена каталогов сообщений формируются на основе *доменов программ*. Различные операционные системы определяют собственные значения по умолчанию, но модулю gettext ничего неизвестно обо всех этих умолчаниях. Он использует заданный по умолчанию каталог локалей sys.prefix + '/share/locale', но в большинстве случаев безопаснее явно задавать значение каталога locale, чем надеяться на то, что значение по умолчанию всегда является корректным. Функция find() позволяет находить нужный каталог сообщений во время выполнения.

Листинг 15.5. gettext_find.py

```
import gettext

catalogs = gettext.find('example', 'locale', all=True)
print('Catalogs:', catalogs)
```

Часть пути, ассоциированная с языком, берется из одной из нескольких переменных среды, пригодных для конфигурирования средств локализации (LANGUAGE, LC_ALL, LC_MESSAGES и LANG). Фактически используется первая найденная переменная из числа указанных. Можно выбрать несколько языков, разделив идентифицирующие их значения символами двоеточия (:). Чтобы продемонстрировать, как это работает, ниже в качестве примера приведены результаты нескольких запусков сценария gettext_find.py.

```
$ cd locale/en_CA/LC_MESSAGES; msgfmt -o example.mo example.po
$ cd ../../..
$ python3 gettext_find.py

Catalogs: ['locale/en_US/LC_MESSAGES/example.mo']

$ LANGUAGE=en_CA python3 gettext_find.py
```

```
Catalogs: ['locale/en_CA/LC_MESSAGES/example.mo']

$ LANGUAGE=en_CA:en_US python3 gettext_find.py

Catalogs: ['locale/en_CA/LC_MESSAGES/example.mo',
'locale/en_US/LC_MESSAGES/example.mo']

$ LANGUAGE=en_US:en_CA python3 gettext_find.py

Catalogs: ['locale/en_US/LC_MESSAGES/example.mo',
'locale/en_CA/LC_MESSAGES/example.mo']
```

Несмотря на то что функция `find()` отображает полный список каталогов, лишь первый из них в этой последовательности фактически загружается для поиска сообщений.

```
$ python3 gettext_example_corrected.py

This message is in the en_US catalog.

$ LANGUAGE=en_CA python3 gettext_example_corrected.py

This message is in the en_CA catalog.

$ LANGUAGE=en_CA:en_US python3 gettext_example_corrected.py

This message is in the en_CA catalog.

$ LANGUAGE=en_US:en_CA python3 gettext_example_corrected.py

This message is in the en_US catalog.
```

15.1.4. Грамматические формы для множественного числа

В то время как простая подстановка сообщений позволяет справиться с большинством проблем их перевода, встречающиеся в сообщениях формы множественного числа имен существительных обрабатывается модулем `gettext` как специальные случаи. В зависимости от языка различия между формами единственного и множественного числа могут требовать изменения не только окончаний отдельных членов предложения, но и всей структуры сообщения. Кроме того, формы множественного числа в сообщениях также могут зависеть от включаемой в сообщение величины числовой характеристики. Чтобы упростить (а в некоторых случаях сделать вообще возможным) управление множественными формами сообщений, для их запроса предоставляется отдельный набор функций.

Листинг 15.6. `gettext_plural.py`

```
from gettext import translation
import sys

t = translation('plural', 'locale', fallback=False)
num = int(sys.argv[1])
```

```
msg = t.ngettext('{num} means singular.',
                 '{num} means plural.', num)

# Все еще требуется самостоятельно добавлять значения в сообщение
print(msg.format(num=num))
```

Для доступа к нескольким подстановочным сообщениям следует использовать функцию `ngettext()`, которая получает в качестве аргументов сообщения, требующие перевода, и счетчик элементов.

```
$ xgettext -L Python -o plural.pot gettext_plural.py
```

Ввиду существования альтернативных форм сообщения, требующих перевода, подстановочные значения возвращаются в виде массива. Использование массива облегчает перевод в случае языков с несколькими формами множественного числа (например, в польском языке существуют различные формы для указания относительного количества).

Листинг 15.7. `plural.pot`

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE
package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2016-07-10 10:45-0400\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;\n"

#: gettext_plural.py:15
#, python-brace-format
msgid "{num} means singular."
msgid_plural "{num} means plural."
msgstr[0] ""
msgstr[1] ""
```

Кроме предоставления строк перевода необходимо уведомить библиотеку о способе формирования грамматических форм множественного числа, чтобы ей было известно, как индексировать массив для любого заданного значения счетчика. Строка `"Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;\n"` включает два значения, которые должны быть заменены вручную: `nplurals` — целое

число, указывающее на размер массива (количество используемых переводов), и `plural` — выражение на языке C, преобразующее указанное количество в индекс массива при поиске перевода. Литеральная строка `n` заменяется количественной величиной, переданной функции `ungettext()`.

Можно привести пример из английского языка, включающий две формы числа. Количество `0` трактуется как множественное ("0 bananas"). Строка `Plural-Forms` в этом случае приобретает следующий вид:

```
Plural-Forms: nplurals=2; plural=n != 1;
```

Перевод, соответствующий единственному числу, должен помещаться в позицию `0`, а перевод, соответствующий множественному числу, — в позицию `1`.

Листинг 15.8. `locale/en_US/LC_MESSAGES/plural.po`

```
# Messages from gettext_plural.py
# Copyright (C) 2009 Doug Hellmann
# This file is distributed under the same license
# as the PyMOTW package.
# Doug Hellmann <doug@doughellmann.com>, 2016.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PyMOTW-3\n"
"Report-Msgid-Bugs-To: Doug Hellmann <doug@doughellmann.com>\n"
"POT-Creation-Date: 2016-01-24 13:04-0500\n"
"PO-Revision-Date: 2016-01-24 13:04-0500\n"
"Last-Translator: Doug Hellmann <doug@doughellmann.com>\n"
"Language-Team: en_US <doug@doughellmann.com>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=n != 1;"

#: gettext_plural.py:15
#, python-format
msgid "{num} means singular."
msgid_plural "{num} means plural."
msgstr[0] "In en_US, {num} is singular."
msgstr[1] "In en_US, {num} is plural."
```

Предварительно скомпилировав каталог, выполним тестовый сценарий несколько раз, чтобы продемонстрировать, как различные значения N преобразуются в индексы строк перевода.

```
$ cd locale/en_US/LC_MESSAGES/; msgfmt -o plural.mo plural.po
$ cd ../../..
$ python3 gettext_plural.py 0

In en_US, 0 is plural.

$ python3 gettext_plural.py 1
```

```
In en_US, 1 is singular.

$ python3 gettext_plural.py 2

In en_US, 2 is plural.
```

15.1.5. Локализация приложений и модулей

Цели, которые преследуются при переводе текста, определяют способ установки и использования модуля gettext в теле кода.

15.1.5.1. Локализация приложения

В случае переводов, действие которых распространяется на все приложение, автор программы может прибегнуть к глобальной установке такой, например, функции, как ngettext(), используя пространство имен __builtins__, поскольку он самостоятельно контролирует высокоуровневый код и ему понятен весь набор соответствующих требований, которые должны быть удовлетворены.

Листинг 15.9. gettext_app_builtin.py

```
import gettext

gettext.install(
    'example',
    'locale',
    names=['ngettext'],
)
print(_('This message is in the script.'))
```

Функция install() связывает функцию gettext() с именем _() в пространстве имен __builtins__. Она также добавляет функцию ngettext() и другие функции, перечисленные в аргументе names.

15.1.5.2. Локализация модуля

Изменение пространства имен __builtins__ библиотеки или отдельного модуля — не совсем хорошая идея, поскольку это может породить конфликты с глобальными значениями приложения. Вместо этого лучше импортировать или повторно связать функции перевода вручную на верхнем уровне модуля.

Листинг 15.10. gettext_module_global.py

```
import gettext

t = gettext.translation(
    'example',
    'locale',
    fallback=False,
)
_ = t.gettext
ngettext = t.ngettext

print(_('This message is in the script.'))
```

15.1.6. Переключение вариантов перевода

Во всех предыдущих примерах на протяжении всего времени выполнения программы использовался один и тот же каталог сообщений. Однако в некоторых ситуациях, особенно в случае веб-приложений, в разное время должны использоваться разные каталоги сообщений, причем это должно осуществляться без выхода из приложения или переустановки переменных среды. В подобных случаях удобно использовать API на основе классов, предоставляемых модулем `gettext`. Вызовы этого API остаются в основном теми же, что и глобальные вызовы, описанные в данном разделе, однако предоставление объекта каталога сообщений, которым можно манипулировать непосредственно, обеспечивает возможность использования нескольких каталогов сообщений.

Дополнительные ссылки

- Раздел документации стандартной библиотеки, посвященный модулю `gettext`[1].
- `locale` (раздел 15.2). Модуль, содержащий другие инструменты локализации.
- GNU gettext[2]. Все форматы каталога сообщений, API и другие вспомогательные средства для этого модуля основаны на оригинальном пакете GNU gettext. С этим пакетом совместимы форматы файлов каталогов, а сценарии командной строки имеют аналогичные (если не идентичные) параметры. В руководстве *GNU gettext utilities*[3] содержится подробное описание форматов файлов и GNU-версий инструментов для работы с ними.
- Формы множественного числа[4]. Обработка форм множественного числа в словах и предложениях на различных языках.
- *Internationalization and Nationalization* (Martin von Löwis)[5]. Статья с описанием методик интернационализации приложений Python.
- Поддержка интернационализации приложений в Django[6]. Неплохой источник информации относительно использования модуля `gettext`, включающий примеры из реальных приложений.

15.2. `locale`: API локализации

Модуль `locale` является частью библиотеки Python, обеспечивающей поддержку интернационализации и локализации приложений. Она предоставляет стандартный способ выполнения операций, которые могут зависеть от языка общения или местоположения пользователя. В число таких операций входят, в частности, форматирование денежных значений, сравнение строк с целью их сортировки и работа с датами. Библиотека не поддерживает перевод текста с одного языка на другой (см. описание модуля `gettext` в разделе 15.1) и преобразование текста в кодировку Unicode (см. описание модуля `codecs` в разделе 6.10).

---

[1] `https://docs.python.org/3.5/library/gettext.html`

[2] `www.gnu.org/software/gettext/`

[3] `www.gnu.org/software/gettext/manual/gettext.html`

[4] `www.gnu.org/software/gettext/manual/gettext.html#Plural-forms`

[5] `http://legacy.python.org/workshops/1997-10/proceedings/loewis.html`

[6] `https://docs.djangoproject.com/en/dev/topics/i18n/`

Примечание

Изменение локали может влиять на все приложение в целом, поэтому рекомендуется не изменять ее в библиотеке, а позволить приложению самостоятельно устанавливать ее в случае необходимости. В приведенных в этом разделе примерах локаль изменяется несколько раз в небольшой программе, чтобы продемонстрировать различия в настройке разнообразных локалей. Вероятно, в случае реальных приложений целесообразно устанавливать локаль лишь один раз при запуске приложения или получении веб-запроса, а не изменять ее многократно.

В этом разделе обсуждаются некоторые высокоуровневые функции, входящие в модуль `locale`. Другие функции — низкоуровневые (`format_string()`) или связанные с управлением локалью приложения (`resetlocale()`).

15.2.1. Проверка региональных настроек

Наиболее распространенный способ дать пользователю возможность изменять настройки локали приложения — использовать переменную среды (`LC_ALL`, `LC_CTYPE`, `LANG` или `LANGUAGE`, в зависимости от платформы). Это позволяет вызывать функцию `setlocale()` без жестко заданных значений.

Листинг 15.11. `locale_env.py`

```
import locale
import os
import pprint

# Значения по умолчанию, заданные в пользовательской среде
locale.setlocale(locale.LC_ALL, '')

print('Environment settings:')
for env_name in ['LC_ALL', 'LC_CTYPE', 'LANG', 'LANGUAGE']:
    print('  {} = {}'.format(
        env_name, os.environ.get(env_name, ''))
    )

# Что собой представляет локаль?
print('\nLocale from environment:', locale.getlocale())

template = """
Numeric formatting:

  Decimal point      : "{decimal_point}"
  Grouping positions : {grouping}
  Thousands separator: "{thousands_sep}"

Monetary formatting:

  International currency symbol   : "{int_curr_symbol!r}"
  Local currency symbol           : {currency_symbol!r}
  Symbol precedes positive value  : {p_cs_precedes}
  Symbol precedes negative value  : {n_cs_precedes}
  Decimal point                   : "{mon_decimal_point}"
```

```
  Digits in fractional values     : {frac_digits}
  Digits in fractional values,
                  international  : {int_frac_digits}
  Grouping positions              : {mon_grouping}
  Thousands separator             : "{mon_thousands_sep}"
  Positive sign                   : "{positive_sign}"
  Positive sign position          : {p_sign_posn}
  Negative sign                   : "{negative_sign}"
  Negative sign position          : {n_sign_posn}

"""

sign_positions = {
    0: 'Surrounded by parentheses',
    1: 'Before value and symbol',
    2: 'After value and symbol',
    3: 'Before value',
    4: 'After value',
    locale.CHAR_MAX: 'Unspecified',
}

info = {}
info.update(locale.localeconv())
info['p_sign_posn'] = sign_positions[info['p_sign_posn']]
info['n_sign_posn'] = sign_positions[info['n_sign_posn']]

print(template.format(**info))
```

Метод `localeconv()` возвращает словарь, содержащий правила данной локали. С полным списком названий и определений можно ознакомиться в документации стандартной библиотеки.

На компьютере Mac, работающем под управлением OS X 10.11.6, выполнение этого сценария без присваивания переменным среды каких-либо значений дало следующие результаты:

```
$ export LANG=; export LC_CTYPE=; python3 locale_env.py

Environment settings:
  LC_ALL =
  LC_CTYPE =
  LANG =
  LANGUAGE =

Locale from environment: ('None', 'None')

Numeric formatting:

  Decimal point      : "."
  Grouping positions : []
  Thousands separator: ""

Monetary formatting:
```

```
  International currency symbol   : "''"
  Local currency symbol            : '$'
  Symbol precedes positive value   : 127
  Symbol precedes negative value   : 127
  Decimal point                    : ""
  Digits in fractional values      : 127
  Digits in fractional values,
                  international    : 127
  Grouping positions               : []
  Thousands separator              : ""
  Positive sign                    : ""
  Positive sign position           : Unspecified
  Negative sign                    : ""
  Negative sign position           : Unspecified
```

Выполнив этот же сценарий с установленной переменной `LANG`, можно увидеть, как при этом изменится локаль и установленная по умолчанию кодировка символов.

США (`en_US`)

```
$ LANG=en_US LC_CTYPE=en_US LC_ALL=en_US python3 locale_env.py

Environment settings:
  LC_ALL = en_US
  LC_CTYPE = en_US
  LANG = en_US
  LANGUAGE =

Locale from environment: ('en_US', 'ISO8859-1')

Numeric formatting:

  Decimal point      : "."
  Grouping positions : [3, 3, 0]
  Thousands separator: ","

Monetary formatting:

  International currency symbol   : "'USD'"
  Local currency symbol            : '$'
  Symbol precedes positive value   : 1
  Symbol precedes negative value   : 1
  Decimal point                    : "."
  Digits in fractional values      : 2
  Digits in fractional values,
                  international    : 2
  Grouping positions               : [3, 3, 0]
  Thousands separator              : ","
  Positive sign                    : ""
  Positive sign position           : Before value and symbol
  Negative sign                    : "-"
  Negative sign position           : Before value and symbol
```

Франция (fr_FR)

```
$ LANG=fr_FR LC_CTYPE=fr_FR LC_ALL=fr_FR python3 locale_env.py

Environment settings:
  LC_ALL = fr_FR
  LC_CTYPE = fr_FR
  LANG = fr_FR
  LANGUAGE =

Locale from environment: ('fr_FR', 'ISO8859-1')

Numeric formatting:

  Decimal point      : ","
  Grouping positions : [127]
  Thousands separator: ""

Monetary formatting:

  International currency symbol   : "'EUR'"
  Local currency symbol           : 'Eu'
  Symbol precedes positive value  : 0
  Symbol precedes negative value  : 0
  Decimal point                   : ","
  Digits in fractional values     : 2
  Digits in fractional values,
                  international  : 2
  Grouping positions              : [3, 3, 0]
  Thousands separator             : " "
  Positive sign                   : ""
  Positive sign position          : Before value and symbol
  Negative sign                   : "-"
  Negative sign position          : After value and symbol
```

Испания (es_ES)

```
$ LANG=es_ES LC_CTYPE=es_ES LC_ALL=es_ES python3 locale_env.py

Environment settings:
  LC_ALL = es_ES
  LC_CTYPE = es_ES
  LANG = es_ES
  LANGUAGE =

Locale from environment: ('es_ES', 'ISO8859-1')

Numeric formatting:

  Decimal point      : ","
  Grouping positions : [127]
  Thousands separator: ""
```

```
Monetary formatting:

  International currency symbol   : "'EUR'"
  Local currency symbol           : 'Eu'
  Symbol precedes positive value  : 0
  Symbol precedes negative value  : 0
  Decimal point                   : ","
  Digits in fractional values     : 2
  Digits in fractional values,
                  international  : 2
  Grouping positions              : [3, 3, 0]
  Thousands separator             : " "
  Positive sign                   : ""
  Positive sign position          : Before value and symbol
  Negative sign                   : "-"
  Negative sign position          : Before value and symbol
```

Португалия (`pt_PT`)

```
$ LANG=pt_PT LC_CTYPE=pt_PT LC_ALL=pt_PT python3 locale_env.py

Environment settings:
  LC_ALL = pt_PT
  LC_CTYPE = pt_PT
  LANG = pt_PT
  LANGUAGE =

Locale from environment: ('pt_PT', 'ISO8859-1')

Numeric formatting:

  Decimal point      : ","
  Grouping positions : []
  Thousands separator: " "

Monetary formatting:

  International currency symbol   : "'EUR'"
  Local currency symbol           : 'Eu'
  Symbol precedes positive value  : 0
  Symbol precedes negative value  : 0
  Decimal point                   : "."
  Digits in fractional values     : 2
  Digits in fractional values,
                  international  : 2
  Grouping positions              : [3, 3, 0]
  Thousands separator             : " "
  Positive sign                   : ""
  Positive sign position          : Before value and symbol
  Negative sign                   : "-"
  Negative sign position          : Before value and symbol
```

Польша (pl_PL)

```
$ LANG=pl_PL LC_CTYPE=pl_PL LC_ALL=pl_PL python3 locale_env.py

Environment settings:
  LC_ALL = pl_PL
  LC_CTYPE = pl_PL
  LANG = pl_PL
  LANGUAGE =

Locale from environment: ('pl_PL', 'ISO8859-2')

Numeric formatting:

  Decimal point      : ","
  Grouping positions : [3, 3, 0]
  Thousands separator: " "

Monetary formatting:

  International currency symbol   : "'PLN'"
  Local currency symbol           : 'z'
  Symbol precedes positive value  : 1
  Symbol precedes negative value  : 1
  Decimal point                   : ","
  Digits in fractional values     : 2
  Digits in fractional values,
                  international  : 2
  Grouping positions              : [3, 3, 0]
  Thousands separator             : " "
  Positive sign                   : ""
  Positive sign position          : After value and symbol
  Negative sign                   : "-"
  Negative sign position          : After value and symbol
```

15.2.2. Денежные единицы

Результаты предыдущего примера демонстрируют, что изменение локали приводит к соответствующему изменению символа валюты и символа, отделяющего дробную часть числа от целой. В следующем примере выводятся положительные и отрицательные денежные значения для нескольких локалей, перебираемых в цикле.

Листинг 15.12. `locale_currency.py`

```
import locale

sample_locales = [
    ('USA', 'en_US'),
    ('France', 'fr_FR'),
    ('Spain', 'es_ES'),
    ('Portugal', 'pt_PT'),
    ('Poland', 'pl_PL'),
]
```

```
for name, loc in sample_locales:
    locale.setlocale(locale.LC_ALL, loc)
    print('{:>10}: {:>10}  {:>10}'.format(
        name,
        locale.currency(1234.56),
        locale.currency(-1234.56),
    ))
```

Результаты приведены ниже.

```
$ python3 locale_currency.py

       USA:   $1234.56   -$1234.56
    France: 1234,56 Eu  1234,56 Eu-
     Spain: 1234,56 Eu  -1234,56 Eu
  Portugal: 1234.56 Eu  -1234.56 Eu
    Poland: łz 1234,56 ł z 1234,56-
```

15.2.3. Форматирование чисел

Числа, не связанные с денежными значениями, также могут форматироваться по-разному в зависимости от локали. В частности, это относится к символу `grouping`, разделяющему группы разрядов в больших числах.

Листинг 15.13. `locale_grouping.py`

```
import locale

sample_locales = [
    ('USA', 'en_US'),
    ('France', 'fr_FR'),
    ('Spain', 'es_ES'),
    ('Portugal', 'pt_PT'),
    ('Poland', 'pl_PL'),
]

print('{:>10} {:>10} {:>15}'.format(
    'Locale', 'Integer', 'Float')
)
for name, loc in sample_locales:
    locale.setlocale(locale.LC_ALL, loc)

    print('{:>10}'.format(name), end=' ')
    print(locale.format('%10d', 123456, grouping=True), end=' ')
    print(locale.format('%15.2f', 123456.78, grouping=True))
```

Для форматирования чисел без символа валюты следует использовать не метод `currency()`, а метод `format()`.

```
$ python3 locale_grouping.py

    Locale    Integer           Float
       USA    123,456      123,456.78
```

```
    France     123456       123456,78
     Spain     123456       123456,78
  Portugal     123456       123456,78
    Poland    123 456      123 456,78
```

Чтобы преобразовать числа, отформатированные в соответствии с определенной локалью, в нормализованные числа, формат которых не связан с локалью, следует использовать функцию `delocalize()`.

Листинг 15.14. `locale_delocalize.py`

```
import locale

sample_locales = [
    ('USA', 'en_US'),
    ('France', 'fr_FR'),
    ('Spain', 'es_ES'),
    ('Portugal', 'pt_PT'),
    ('Poland', 'pl_PL'),
]

for name, loc in sample_locales:
    locale.setlocale(locale.LC_ALL, loc)
    localized = locale.format('%0.2f', 123456.78, grouping=True)
    delocalized = locale.delocalize(localized)
    print('{:>10}: {:>10}  {:>10}'.format(
        name,
        localized,
        delocalized,
    ))
```

Разделители групп разрядов удаляются, а разделитель целой и десятичной части чисел преобразуется в символ точки (`.`).

```
$ python3 locale_delocalize.py

       USA: 123,456.78   123456.78
    France:  123456,78   123456.78
     Spain:  123456,78   123456.78
  Portugal:  123456,78   123456.78
    Poland: 123 456,78   123456.78
```

15.2.4. Анализ чисел

Кроме генерации вывода в различных форматах модуль `locale` может оказаться полезным для синтаксического анализа вводимых чисел. Он включает функции `atoi()` и `atof()`, преобразующие строки в целочисленные значения и значения с плавающей точкой на основании соглашений о форматировании чисел, принятых для заданной локали.

Листинг 15.15. `locale_atof.py`

```
import locale

sample_data = [
    ('USA', 'en_US', '1,234.56'),
    ('France', 'fr_FR', '1234,56'),
    ('Spain', 'es_ES', '1234,56'),
    ('Portugal', 'pt_PT', '1234.56'),
    ('Poland', 'pl_PL', '1 234,56'),
]

for name, loc, a in sample_data:
    locale.setlocale(locale.LC_ALL, loc)
    print('{:>10}: {:>9} => {:f}'.format(
        name,
        a,
        locale.atof(a),
    ))
```

Парсер распознает символы разделителя групп разрядов и десятичной точки текущей локали.

```
$ python3 locale_atof.py
       USA:  1,234.56 => 1234.560000
    France:   1234,56 => 1234.560000
     Spain:   1234,56 => 1234.560000
  Portugal:   1234.56 => 1234.560000
    Poland:  1 234,56 => 1234.560000
```

15.2.5. Дата и время

Другим важным аспектом локализации является форматирование значений даты и времени.

Листинг 15.16. `locale_date.py`

```
import locale
import time

sample_locales = [
    ('USA', 'en_US'),
    ('France', 'fr_FR'),
    ('Spain', 'es_ES'),
    ('Portugal', 'pt_PT'),
    ('Poland', 'pl_PL'),
]

for name, loc in sample_locales:
    locale.setlocale(locale.LC_ALL, loc)
    format = locale.nl_langinfo(locale.D_T_FMT)
    print('{:>10}: {}'.format(name, time.strftime(format)))
```

В этом примере для вывода значений даты и времени используется строка форматирования, соответствующая текущей локали.

```
$ python3 locale_date.py

       USA: Fri Aug 5  17:33:31 2016
    France: Ven  5 août 17:33:31 2016
     Spain: vie  5 ago 17:33:31 2016
  Portugal: Sex  5 Ago 17:33:31 2016
    Poland: ptk  5 sie 17:33:31 2016
```

Дополнительные ссылки

- Раздел документации стандартной библиотеки, посвященный модулю `locale`[7].
- Замечания относительно портирования программ из Python 2 в Python 3, касающиеся модуля `locale` (раздел A.6.24).
- `gettext` (раздел 15.1). Каталоги сообщений с вариантами перевода на другие языки.

[7] https://docs.python.org/3.5/library/locale.html