locale — Cultural Localization API

Purpose: Format and parse values that depend on location or language.

The `locale` module is part of Python's internationalization and localization support library. It provides a standard way to handle operations that may depend on the language or location of a user. For example, it handles formatting numbers as currency, comparing strings for sorting, and working with dates. It does not cover translation (see the `gettext` module) or Unicode encoding (see the `codecs` module).

Note

Changing the locale can have application-wide ramifications, so the recommended practice is to avoid changing the value in a library and to let the application set it one time. In the examples in this section, the locale is changed several times within a short program to highlight the differences in the settings of various locales. It is far more likely that an application will set the locale once as it starts up or when a web request is received and not change it.

This section covers some of the high-level functions in the `locale` module. There are others which are lower level or which relate to managing the locale for an application (`resetlocale()`).

Probing the Current Locale

The most common way to let the user change the locale settings for an application is through an environment variable (`LC_ALL`, `LC_CTYPE`, `LANG`, or `LANGUAGE`, depending on the platform). The application then calls `setlocale()` without a hard-coded value, and the environment value is used.

```
# locale_env.py

import locale
import os
import pprint

# Default settings based on the user's environment.
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, ''))
    )

# What is the locale?
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))
```

The `localeconv()` method returns a dictionary containing the locale's conventions. The full list of value names and definitions is covered in the standard library documentation.

A Mac running OS X 10.11.6 with all of the variables unset produces this output:

```
$ LANG= LC_CTYPE= PYTHONCOERCECLOCALE=0 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
```

Running the same script with the `LANG` variable set shows how the locale and default encoding change.

The United States (`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
```

France (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
```

Spain (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
```

Portugal (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
```

Poland (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           : 'zł'
  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
  Negative sign                   : "-"
  Negative sign position          : After value
```

Currency

The earlier example output shows that changing the locale updates the currency symbol setting and the character to separate whole numbers from decimal fractions. This example loops through several different locales to print a positive and negative currency value formatted for each locale.

```
# 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),
    ))
```

The output is this small table:

```
$ 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: zł 1234,56  zł 1234,56-
```

Formatting Numbers

Numbers not related to currency are also formatted differently depending on the locale. In particular, the `grouping` character used to separate large numbers into readable chunks changes.

```
# 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_string('%10d', 123456, grouping=True), end=' ')
    print(locale.format_string('%15.2f', 123456.78, grouping=True))
```

To format numbers without the currency symbol, use `format_string()` instead of `currency()`.

```
$ 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
```

To convert locale-formatted numbers to a normalized locale-agnostic format, use `delocalize()`.

```
# 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_string('%0.2f', 123456.78, grouping=True)
    delocalized = locale.delocalize(localized)
    print('{:>10}: {:>10}  {:>10}'.format(
        name,
        localized,
        delocalized,
    ))
```

Grouping punctuation is removed and the decimal separator is converted to always be a `.`.

```
$ 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
```

Parsing Numbers

Besides generating output in different formats, the `locale` module helps with parsing input. It includes `atoi()` and `atof()`

functions for converting the strings to integer and floating point values based on the locale's numerical formatting conventions.

```
# 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),
    ))
```

The grouping and decimal separator values of the locale are recognized by the parser.

```
$ 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
```

Dates and Times

Another important aspect of localization is date and time formatting.

```
# 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)))
```

This example uses the date formatting string for the locale to print the current date and time.

```
$ python3 locale_date.py

       USA: Sun Dec  9 12:20:00 2018
    France: Dim  9 déc 12:20:00 2018
     Spain: dom  9 dic 12:20:00 2018
  Portugal: Dom  9 Dez 12:20:00 2018
    Poland: ndz  9 gru 12:20:00 2018
```

See also

- Standard library documentation for locale

- Standard library documentation for locale
- Python 2 to 3 porting notes for locale
- `gettext` – Message catalogs for translations.

gettext — Message Catalogs

Developer Tools

Quick Links

This page was last updated 2018-12-09.

Navigation

gettext — Message Catalogs
Developer Tools

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?