`!unicodedata` --- Unicode Database

unicodedata

single: Unicode single: character pair: Unicode; database

This module provides access to the Unicode Character Database (UCD) which defines character properties for all Unicode characters. The data contained in this database is compiled from the UCD version 18.0.0.

The module uses the same names and symbols as defined by Unicode Standard Annex #44, "Unicode Character Database".

The `unicode-howto` for more information about Unicode and how to use this module.

Lookup	
`lookup(name) <lookup>`	Look up character by name
`name(chr) <name>`	Return the name assigned to a character
Numeric values	
`decimal(chr) <unicodedata.decimal>`	Decimal value of a character
`digit(chr) <digit>`	Digit value of a character
`numeric(chr) <numeric>`	Numeric value of a character
Properties	
`bidirectional(chr) <bidirectional>`	Bidirectional class of a character
`block(chr) <block>`	Unicode block of a character
`category(chr) <category>`	General category of a character
`combining(chr) <combining>`	Canonical combining class of a character
`decomposition(chr) <decomposition>`	Character decomposition mapping
`east_asian_width(chr) <east_asian_width>`	East Asian width of a character
`extended_pictographic(chr) <extended_pictographic>`	Check if a character has the Extended_Pictographic property

`grapheme_cluster_break(chr) <grapheme_cluster_break>`	Grapheme_Cluster_Break property of a character
`indic_conjunct_break(chr) <indic_conjunct_break>`	Indic_Conjunct_Break property of a character
`isxidcontinue(chr) <isxidcontinue>`	Check if a character is a valid identifier continuation
`isxidstart(chr) <isxidstart>`	Check if a character is a valid identifier start
`mirrored(chr) <mirrored>`	Mirrored property of a character
Normalization	
`normalize(form, unistr) <normalize>`	Return the normalized form of a string
`is_normalized(form, unistr) <is_normalized>`	Check if a Unicode string is normalized
Text segmentation	
`iter_graphemes(unistr) <iter_graphemes>`	Iterate over grapheme clusters in a string

lookup(name, /)

Look up character by name. If a character with the given name is found, return the corresponding character. If not found, `KeyError` is raised. For example:

```
>>> unicodedata.lookup('LEFT CURLY BRACKET')
'{'
```

The characters returned by this function are the same as those produced by `\N` escape sequence in string literals. For example:

```
>>> unicodedata.lookup('MIDDLE DOT') == '\N{MIDDLE DOT}'
True
```

3.3 Support for name aliases[^1] and named sequences[^2] has been added.

name(chr, default=None, /)

Returns the name assigned to the character *chr* as a string. If no name is defined, *default* is returned, or, if not given, `ValueError` is raised. For example:

```
>>> unicodedata.name('½')
'VULGAR FRACTION ONE HALF'
>>> unicodedata.name('\uFFFF', 'fallback')
'fallback'
```

decimal(chr, default=None, /)

Returns the decimal value assigned to the character *chr* as integer. If no such value is defined, *default* is returned, or, if not given, `ValueError` is raised. For example:

```
>>> unicodedata.decimal('\N{ARABIC-INDIC DIGIT NINE}')
9
>>> unicodedata.decimal('\N{SUPERSCRIPT NINE}', -1)
-1
```

digit(chr, default=None, /)

Returns the digit value assigned to the character *chr* as integer. If no such value is defined, *default* is returned, or, if not given, `ValueError` is raised:

```
>>> unicodedata.digit('\N{SUPERSCRIPT NINE}')
9
```

numeric(chr, default=None, /)

Returns the numeric value assigned to the character *chr* as float. If no such value is defined, *default* is returned, or, if not given, `ValueError` is raised:

```
>>> unicodedata.numeric('½')
0.5
```

category(chr, /)

Returns the general category assigned to the character *chr* as string. General category names consist of two letters. See the General Category Values section of the Unicode Character Database documentation for a list of category codes. For example:

```
>>> unicodedata.category('A')  # 'L'etter, 'u'ppercase
'Lu'
```

bidirectional(chr, /)

Returns the bidirectional class assigned to the character *chr* as string. If no such value is defined, an empty string is returned. See the Bidirectional Class Values section of the Unicode Character Database documentation for a list of bidirectional codes. For example:

```
>>> unicodedata.bidirectional('\N{ARABIC-INDIC DIGIT SEVEN}') # 'A'rabic, 'N'umber
'AN'
```

combining(chr, /)

Returns the canonical combining class assigned to the character *chr* as integer. Returns `0` if no combining class is defined. See the Canonical Combining Class Values section of the Unicode Character Database for more information.

east_asian_width(chr, /)

Returns the east asian width assigned to the character *chr* as string. For a list of widths and or more information, see the Unicode Standard Annex #11.

block(chr, /)

Returns the block assigned to the character *chr*. For example:

```
>>> unicodedata.block('S')
'Basic Latin'
```

3.15

mirrored(chr, /)

Returns the mirrored property assigned to the character *chr* as integer. Returns `1` if the character has been identified as a "mirrored" character in bidirectional text, `0` otherwise. For example:

```
>>> unicodedata.mirrored('>')
1
```

isxidstart(chr, /)

Return `True` if *chr* is a valid identifier start per the Unicode Standard Annex #31, that is, it has the `XID_Start` property. Return `False` otherwise. For example:

```
>>> unicodedata.isxidstart('S')
True
>>> unicodedata.isxidstart('0')
False
```

3.15

isxidcontinue(chr, /)

Return `True` if *chr* is a valid identifier character per the Unicode Standard Annex #31, that is, it has the `XID_Continue` property. Return `False` otherwise. For example:

```
>>> unicodedata.isxidcontinue('S')
True
>>> unicodedata.isxidcontinue(' ')
False
```

3.15

decomposition(chr, /)

Returns the character decomposition mapping assigned to the character *chr* as string. An empty string is returned in case no such mapping is defined. For example:

```
>>> unicodedata.decomposition('Ã')
'0041 0303'
```

grapheme_cluster_break(chr, /)

Returns the Grapheme_Cluster_Break property assigned to the character.

3.15

indic_conjunct_break(chr, /)

Returns the Indic_Conjunct_Break property assigned to the character.

3.15

extended_pictographic(chr, /)

Returns `True` if the character has the Extended_Pictographic property, `False` otherwise.

3.15

normalize(form, unistr, /)

Return the normal form *form* for the Unicode string *unistr*. Valid values for *form* are 'NFC', 'NFKC', 'NFD', and 'NFKD'.

The Unicode standard defines various normalization forms of a Unicode string, based on the definition of canonical equivalence and compatibility equivalence. In Unicode, several characters

can be expressed in various way. For example, the character U+00C7 (LATIN CAPITAL LETTER C WITH CEDILLA) can also be expressed as the sequence U+0043 (LATIN CAPITAL LETTER C) U+0327 (COMBINING CEDILLA).

For each character, there are two normal forms: normal form C and normal form D. Normal form D (NFD) is also known as canonical decomposition, and translates each character into its decomposed form. Normal form C (NFC) first applies a canonical decomposition, then composes pre-combined characters again.

In addition to these two forms, there are two additional normal forms based on compatibility equivalence. In Unicode, certain characters are supported which normally would be unified with other characters. For example, U+2160 (ROMAN NUMERAL ONE) is really the same thing as U+0049 (LATIN CAPITAL LETTER I). However, it is supported in Unicode for compatibility with existing character sets (for example, gb2312).

The normal form KD (NFKD) will apply the compatibility decomposition, that is, replace all compatibility characters with their equivalents. The normal form KC (NFKC) first applies the compatibility decomposition, followed by the canonical composition.

Even if two unicode strings are normalized and look the same to a human reader, if one has combining characters and the other doesn't, they may not compare equal.

is_normalized(form, unistr, /)

Return whether the Unicode string *unistr* is in the normal form *form*. Valid values for *form* are 'NFC', 'NFKC', 'NFD', and 'NFKD'.

3.8

iter_graphemes(unistr, start=0, end=sys.maxsize, /)

Returns an iterator to iterate over grapheme clusters. With optional *start*, iteration begins at that position. With optional *end*, iteration stops at that position.

Converting an emitted item to string returns a substring corresponding to the grapheme cluster. Its `start` and `end` attributes denote the start and end of the grapheme cluster.

It uses extended grapheme cluster rules defined by Unicode Standard Annex #29, "Unicode Text Segmentation".

3.15

In addition, the module exposes the following constants:

unidata_version

The version of the Unicode database used in this module.

ucd_3_2_0

This is an object that has most of the methods of the entire module, but uses the Unicode database version 3.2 instead, for applications that require this specific version of the Unicode database (such as IDNA).

Footnotes

□□□ #2
□ □□□ □□□ 2026-10-31 17:22:59 UTC
□ □□□ □□□ 2026-10-31 17:23:01 UTC