Unicode

Unicode.isassigned — Function

```
Unicode.isassigned(c) -> Bool
```

Returns `true` if the given char or integer is an assigned Unicode code point.

Examples

```
julia> Unicode.isassigned(101)
true

julia> Unicode.isassigned('\x01')
true
```

Unicode.normalize — Function

```
Unicode.normalize(s::AbstractString; keywords...)
Unicode.normalize(s::AbstractString, normalform::Symbol)
```

Normalize the string `s`. By default, canonical composition (`compose=true`) is performed without ensuring Unicode versioning stability (`compat=false`), which produces the shortest possible equivalent string but may introduce composition characters not present in earlier Unicode versions.

Alternatively, one of the four "normal forms" of the Unicode standard can be specified: `normalform` can be `:NFC`, `:NFD`, `:NFKC`, or `:NFKD`. Normal forms C (canonical composition) and D (canonical decomposition) convert different visually identical representations of the same abstract string into a single canonical form, with form C being more compact. Normal forms KC and KD additionally canonicalize "compatibility equivalents": they convert characters that are abstractly similar but visually distinct into a single canonical choice (e.g. they expand ligatures into the individual characters), with form KC being more compact.

Alternatively, finer control and additional transformations may be obtained by calling `Unicode.normalize(s; keywords...)`, where any number of the following boolean keywords options (which all default to `false` except for `compose`) are specified:

- `compose=false`: do not perform canonical composition
- `decompose=true`: do canonical decomposition instead of canonical composition (`compose=true` is ignored if present)
- `compat=true`: compatibility equivalents are canonicalized
- `casefold=true`: perform Unicode case folding, e.g. for case-insensitive string comparison
- `newline2lf=true`, `newline2ls=true`, or `newline2ps=true`: convert various newline sequences (LF, CRLF, CR, NEL) into a linefeed (LF), line-separation (LS), or paragraph-separation (PS) character, respectively
- `stripmark=true`: strip diacritical marks (e.g. accents)
- `stripignore=true`: strip Unicode's "default ignorable" characters (e.g. the soft hyphen or the left-to-right marker)
- `stripcc=true`: strip control characters; horizontal tabs and form feeds are converted to spaces; newlines are also converted to spaces unless a newline-conversion flag was specified
- `rejectna=true`: throw an error if unassigned code points are found
- `stable=true`: enforce Unicode versioning stability (never introduce characters missing from earlier Unicode versions)

For example, NFKC corresponds to the options `compose=true, compat=true, stable=true`.

Examples

```
julia> "é" == Unicode.normalize("é") #LHS: Unicode U+00e9, RHS: U+0065 & U+0301
true

julia> "μ" == Unicode.normalize("µ", compat=true) #LHS: Unicode U+03bc, RHS: Un
true

julia> Unicode.normalize("JuLiA", casefold=true)
"julia"

julia> Unicode.normalize("JúLiA", stripmark=true)
"JuLiA"
```

`Unicode.graphemes` — Function

```
graphemes(s::AbstractString) -> GraphemeIterator
```

Returns an iterator over substrings of `s` that correspond to the extended graphemes in the string, as defined by Unicode UAX #29. (Roughly, these are what users would perceive as single characters, even though they may contain more than one codepoint; for example a letter combined with an accent mark is a single grapheme.)

« UUIDs Reflection and introspection »