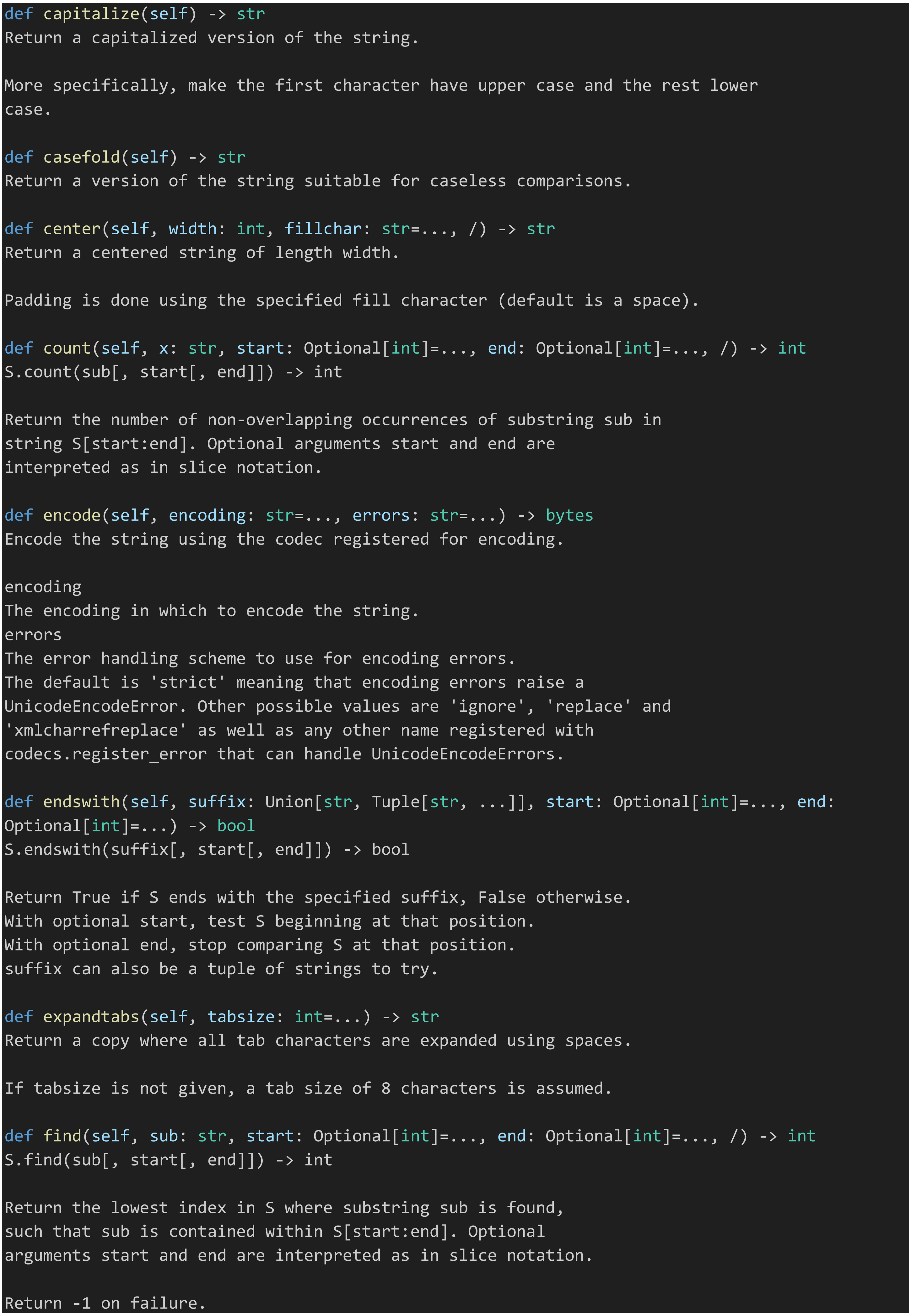
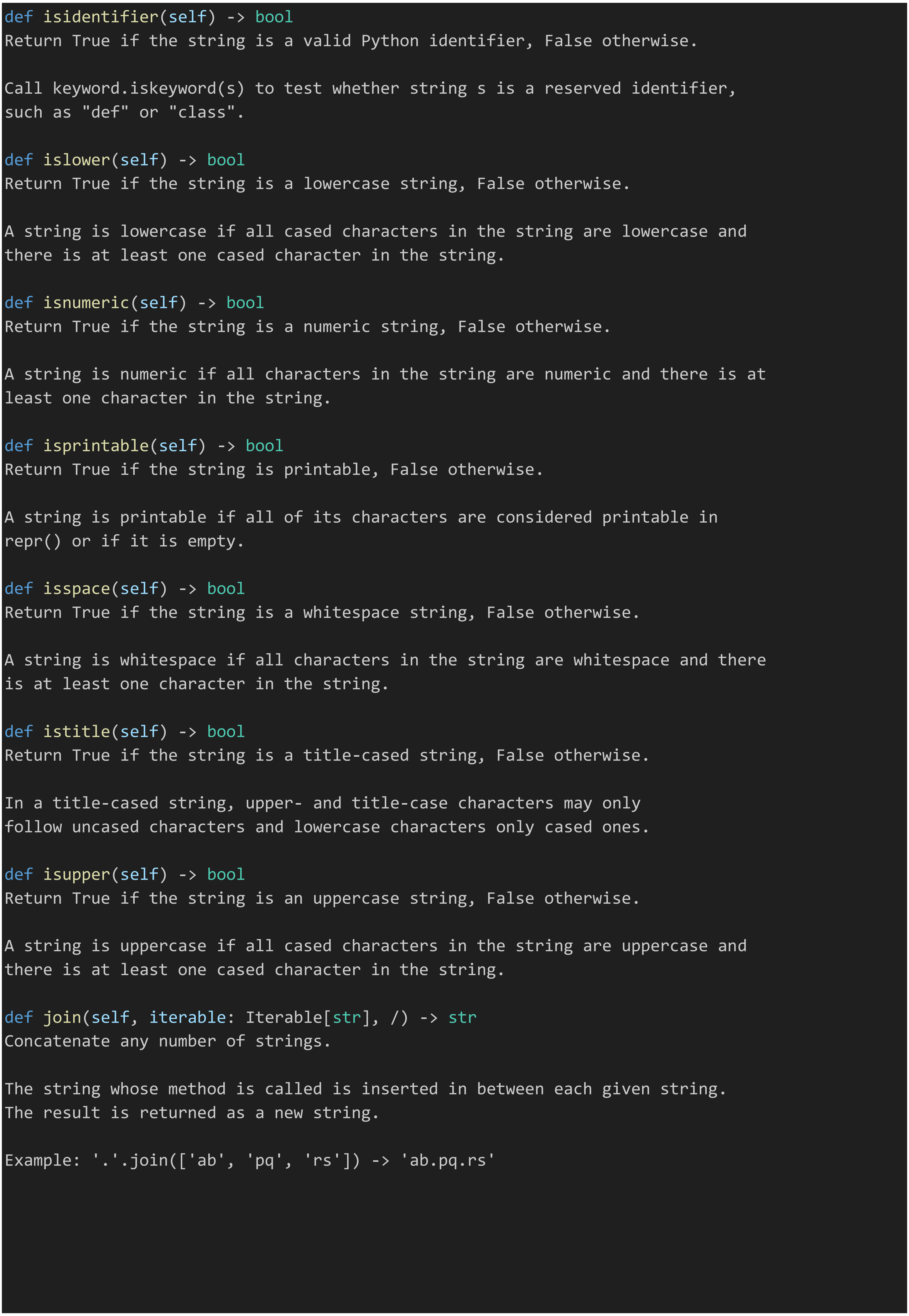


```
def capitalize(self) -> str
Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower
case.

def casefold(self) -> str
Return a version of the string suitable for caseless comparisons.

def center(self, width: int, fillchar: str=..., /) -> str
Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

def count(self, x: str, start: Optional[int]=..., end: Optional[int]=..., /) -> int
S.count(sub[, start[, end]]) -> int

Return the number of non-overlapping occurrences of substring sub in
string S[start:end]. Optional arguments start and end are
interpreted as in slice notation.

def encode(self, encoding: str=..., errors: str=...) -> bytes
Encode the string using the codec registered for encoding.

encoding
The encoding in which to encode the string.
errors
The error handling scheme to use for encoding errors.
The default is 'strict' meaning that encoding errors raise a
UnicodeEncodeError. Other possible values are 'ignore', 'replace' and
'xmlcharrefreplace' as well as any other name registered with
codecs.register_error that can handle UnicodeEncodeErrors.

def endswith(self, suffix: Union[str, Tuple[str, ...]], start: Optional[int]=..., end:
Optional[int]=...) -> bool
S.endswith(suffix[, start[, end]]) -> bool

Return True if S ends with the specified suffix, False otherwise.
With optional start, test S beginning at that position.
With optional end, stop comparing S at that position.
suffix can also be a tuple of strings to try.

def expandtabs(self, tabsize: int=...) -> str
Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

def find(self, sub: str, start: Optional[int]=..., end: Optional[int]=..., /) -> int
S.find(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found,
such that sub is contained within S[start:end]. Optional
arguments start and end are interpreted as in slice notation.

Return -1 on failure.
```

```
def format(self, *args: object, **kwargs: object) -> str
S.format(*args, **kwargs) -> str

Return a formatted version of S, using substitutions from args and kwargs.
The substitutions are identified by braces ('{' and '}').

def format_map(self, map: _FormatMapMapping) -> str
S.format_map(mapping) -> str

Return a formatted version of S, using substitutions from mapping.
The substitutions are identified by braces ('{' and '}').

def index(self, sub: str, start: Optional[int]=..., end: Optional[int]=..., /) -> int
S.index(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found,
such that sub is contained within S[start:end]. Optional
arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

def isalnum(self) -> bool
Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and
there is at least one character in the string.

def isalpha(self) -> bool
Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there
is at least one character in the string.

def isascii(self) -> bool
Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F.
Empty string is ASCII too.

def isdecimal(self) -> bool
Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and
there is at least one character in the string.

def isdigit(self) -> bool
Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there
is at least one character in the string.
```



```
def isidentifier(self) -> bool
Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier,
such as "def" or "class".

def islower(self) -> bool
Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and
there is at least one cased character in the string.

def isnumeric(self) -> bool
Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at
least one character in the string.

def isprintable(self) -> bool
Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in
repr() or if it is empty.

def isspace(self) -> bool
Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there
is at least one character in the string.

def istitle(self) -> bool
Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only
follow uncased characters and lowercase characters only cased ones.

def isupper(self) -> bool
Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and
there is at least one cased character in the string.

def join(self, iterable: Iterable[str], /) -> str
Concatenate any number of strings.

The string whose method is called is inserted in between each given string.
The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
```

[redacted]