```python
from typing import List
import logging as log
log.basicConfig(level=log.DEBUG)



class Position:
    def __init__(self, line, character):
        self.line = line
        self.character = character

    def __repr__(self):
        return f"({self.line}:{self.character})"

def is_char_beyond_multilingual_plane(char: str) -> bool:
    return ord(char) > 0xFFFF


def utf16_unit_offset(chars: str):
    """Calculate the number of characters which need two utf-16 code units.
    Arguments:
        chars (str): The string to count occurrences of utf-16 code units for.
    """
    return sum(is_char_beyond_multilingual_plane(ch) for ch in chars)


def utf16_num_units(chars: str):
    """Calculate the length of `str` in utf-16 code units.
    Arguments:
        chars (str): The string to return the length in utf-16 code units for.
    """
    return len(chars) + utf16_unit_offset(chars)

def position_from_utf16(lines: List[str], position: Position) -> Position:
    """Convert the position.character from utf-16 code units to utf-32.
    A python application can't use the character member of `Position`
    directly. As per specification it is represented as a zero-based line and
    character offset based on a UTF-16 string representation.
    All characters whose code point exceeds the Basic Multilingual Plane are
    represented by 2 UTF-16 code units.
    The offset of the closing quotation mark in x="😋" is
    - 5 in UTF-16 representation
    - 4 in UTF-32 representation
    see: https://github.com/microsoft/language-server-protocol/issues/376
    Arguments:
        lines (list):
            The content of the document which the position refers to.
        position (Position):
            The line and character offset in utf-16 code units.
    Returns:
        The position with `character` being converted to utf-32 code units.
    """
    if position.line >= len(lines):
        # start of the line after last
        return Position(len(lines), 0)  # or return position

    line = lines[position.line]

    _utf32_len = len(line)
    _utf32_index = 0
    _utf16_index = 0
    while (_utf16_index < position.character) and (_utf32_index < _utf32_len):
        _current_char = line[_utf32_index]
        is_double_width = is_char_beyond_multilingual_plane(_current_char)
```

```
65:         if (is_double_width):
66:             _utf16_index += 2
67:         else:
68:             _utf16_index += 1
69:         _utf32_index += 1
70:
71:     position = Position(
72:         line=position.line,
73:         character=_utf32_index
74:     )
75:     return position
76:
77:
78: source = """
79: 😋😋
80: """
81:
82: lines = source.split("\n")
83: print(lines)
84: pos = Position(4,0)
85:
86: pos = position_from_utf16(lines, pos)
87: print(pos)
88:
```