```
#Lexing is a trivial, almost 1:1, transformation from characters to tokens
#The thing that usually makes lexing hard, which is multi-character tokens,
#are not present in the regex dialect we're lexing
#-------------------------------------------------------------------------------
#So why do we need Lexing then if it's so trivial ?
#Well, two reasons :
#     1- We will need to do parsing, and parsing on raw characters is ugly
#
#     2- (Optional) We will need to handle escapes :
#                     Suppose you want to match the literal string '(', how
#                     would you do it ?
#                     If your regex engine doesn't handle escapes, you
#                     can't, but with escapes it's simply the regex "\("
#
#If (1) and (2) aren't convincing enough to you, feel free to simply jump right ahead
to parsing :)

enum token-type {
            OR,
            STAR,
            PLUS,
            QUESTION_MARK,
            OPEN_PARENTHESIS,
            CLOSED_PARENTHESIS,
            OPEN_SQUARE_BRACKET,
            CLOSED_SQUARE_BRACKET,
            DASH,
            LITERAL_CHARACTER
}
class token {
    token-type type,
    string str
}
subroutine Regex-Lex of
      input regex-string
      output token-stream:

      let meta-character-map = a map
                               from '|' to OR
                               from '*' to STAR
                               ...
                               from '-' to DASH
```

```
      token-stream = empty-stream with token-pointer at 0

      prev-character = None

      for each character in regex-string:
            if the current character is the escape character
            then
                    prev-character = the current character
                    continue from the next character

            if the current character is in the meta-character-map
            and the prev-character was not an escape
            then
                  put a token whose type is meta-character-map[current character]
and whose str is the current character
            else
                  put a token whose type is LITERAL_CHARACTER and whose str is the
current character

            prev-character = the current character

      return token-stream

#Note: Lexer completely ignores escape character, so whatever character you choose for
#them, it's one you can't match literally (and you can easily modify it to allow
escaping #the escapes themselves)
```