

The Lexer

psilo has a comparatively simple grammar and lexing it is fairly straightforward. Parsec has a number of utilities for accomplishing exactly what I wish to accomplish so I happily and humbly defer to its facilities.

The main purpose of this module is to remove some clutter from `Parser`.

```
module Lexer where
```

```
import Text.Parsec
import Text.Parsec.String (Parser)
import Text.Parsec.Language (emptyDef)
import qualified Text.Parsec.Token as Tok
```

```
lexer :: Tok.TokenParser ()
```

```
lexer = Tok.makeTokenParser style
```

```
  where ops = []
```

```
        names = ["\\", "=", ":", "let", "apply"]
```

```
        idStarts = letter <|> char '_'
```

```
        idLetters = letter <|> char '_' <|> digit <|> char '-'
```

```
                <|> char '+' <|> char '?' <|> char ':' <|> char '&'
```

```
        opStarts = oneOf " !$%&|*+~/<=>?@^_~#"
```

```
        opLetters = oneOf " !$%&|*+~/<=>?@^_~#" <|> letter <|> digit <|> char '-' <|> char
```

```
        style = emptyDef {
```

```
            Tok.commentLine = ";"
```

```
            , Tok.reservedOpNames = ops
```

```
            , Tok.reservedNames = names
```

```
            , Tok.caseSensitive = True
```

```
            , Tok.identStart = idStarts
```

```
            , Tok.identLetter = idLetters
```

```
            , Tok.opStart = opStarts
```

```
            , Tok.opLetter = opLetters
```

```
            , Tok.commentStart = "/*"
```

```
            , Tok.commentEnd = "*/"
```

```
        }
```

```
integer :: Parser Integer
```

```
integer = Tok.integer lexer

parens :: Parser a -> Parser a
parens = Tok.parens lexer

whitespace :: Parser ()
whitespace = Tok.whiteSpace lexer

nl :: Parser ()
nl = skipMany newline

reserved :: String -> Parser ()
reserved = Tok.reserved lexer

reservedOp :: String -> Parser ()
reservedOp = Tok.reservedOp lexer

identifier :: Parser String
identifier = Tok.identifier lexer

operator :: Parser String
operator = Tok.operator lexer
```