Parser - parsing Terms

```
# Term := Factor (("*"|"/") Factor)*
def parse_term(tokens, pos):
    node, pos = parse_factor(tokens, pos)
    while True:
        tok = peek(tokens, pos)
        if tok == lexer.OPMUL:
            pos += 1
            right, pos = parse_factor(tokens, pos)
            node = ast_types.MulExp(node, right)
        elif tok == lexer.OPDIV:
            pos += 1
            right, pos = parse_factor(tokens, pos)
            node = ast_types.DivExp(node, right)
        else:
            break
    return node, pos
```

Parser - parsing Factors

```
# Factor := "-" Factor | "(" Expression ")" | Number
def parse_factor(tokens, pos):
    tok = peek(tokens, pos)

    if tok == lexer.OPMINUS:
        pos += 1
        subexpr, pos = parse_factor(tokens, pos)
        return ast_types.NegExp(subexpr), pos

    if tok == lexer.LPAREN:
        pos += 1
        expr, pos = parse_expression(tokens, pos)
        if peek(tokens, pos) != lexer.RPAREN:
            raise SyntaxError("Expected {}".format(lexer.RPAREN))
        pos += 1
        return expr, pos

    if tok is not None and tok.isdigit():
        pos += 1
        return ast_types.Number(tok), pos

    raise SyntaxError(f"Unexpected token: {tok}")
```