Parser implementation

- Method of parsing that we will use is known as "recursive descent" parsing
- Determine which type of expression we must be at based on current and next tokens (can see next token by "peeking" at lexeme with next index)
- Our parser differs a little from the grammar given on the previous slide as some work is done by the lexer (digit grouping):

```
expression  ::= term (("+" | "-") term)*
term        ::= factor (("*" | "/") factor)*
factor      ::= "-" factor | "(" expression ")" | number
```

Parser - parsing Expressions

```
# Expression := Term (("+"|"-") Term)*
def parse_expression(tokens, pos):
    node, pos = parse_term(tokens, pos)
    while True:
        tok = peek(tokens, pos)
        if tok == lexer.OPPLUS:
            pos += 1
            right, pos = parse_term(tokens, pos)
            node = ast_types.AddExp(node, right)
        elif tok == lexer.OPMINUS:
            pos += 1
            right, pos = parse_term(tokens, pos)
            node = ast_types.SubExp(node, right)
        else:
            break
    return node, pos
```