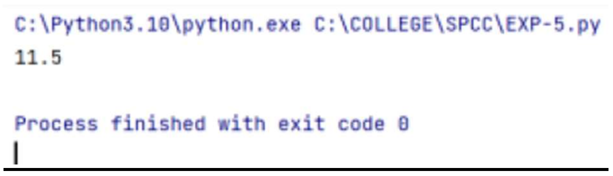
Experiment no 5

```
import re
class Token:
    def __init__(self, token_type, value):
        self.token_type = token_type
        self.value = value
class Parser:
    def __init__(self, text):
        self.tokens = self.tokenize(text)
        self.pos = 0
    def parse(self):
        return self.expr()
    def tokenize(self, text):
        token_exprs = [
            (r'\d+', 'INT'),
            (r'\+', 'PLUS'),
            (r'-', 'MINUS'),
            (r'\*', 'MULTIPLY'),
            (r'/', 'DIVIDE'),
            (r'\(', 'LPAREN'),
            (r'\)', 'RPAREN'),
            (r'\s', None)  # skip whitespace
        ]
        tokens = []
        pos = 0
        while pos < len(text):
            match = None
            for token_expr in token_exprs:
                pattern, token_type = token_expr
                regex = re.compile(pattern)
                match = regex.match(text, pos)
                if match:
                    value = match.group(0)
                    if token_type:
                        token = Token(token_type, value)
                        tokens.append(token)
                    break
            if not match:
                raise ValueError(f'Invalid input at position {pos}')
            else:
                pos = match.end(0)
        return tokens
    def consume(self, token_type):
        if self.pos < len(self.tokens) and self.tokens[self.pos].token_type == token_type:
            self.pos += 1
        else:
            raise ValueError(f'Expected token type {token_type} at position {self.pos}')
    def factor(self):
        token = self.tokens[self.pos]
        if token.token_type == 'INT':
            self.consume('INT')
            return int(token.value)
        elif token.token_type == 'LPAREN':
            self.consume('LPAREN')
            value = self.expr()
            self.consume('RPAREN')
            return value
    def term(self):
        value = self.factor()
        while self.pos < len(self.tokens):
            token = self.tokens[self.pos]
            if token.token_type == 'MULTIPLY':
                self.consume('MULTIPLY')
                value *= self.factor()
            elif token.token_type == 'DIVIDE':
                self.consume('DIVIDE')
                value /= self.factor()
            else:
                break
        return value
    def expr(self):
        value = self.term()
        while self.pos < len(self.tokens):
            token = self.tokens[self.pos]
            if token.token_type == 'PLUS':
                self.consume('PLUS')
                value += self.term()
            elif token.token_type == 'MINUS':
                self.consume('MINUS')
                value -= self.term()
            else:
                break
        return value
text = '2 * (3 + 4) - 5 / 2'
parser = Parser(text)
result = parser.parse()
print(result)  # Output: 11.5
```

Output:



```
C:\Python3.10\python.exe C:\COLLEGE\SPCC\EXP-5.py
11.5

Process finished with exit code 0
```