```
def is_at_end(self) -> bool:
    """Verifica si se ha llegado al final del código fuente"""
    return self.actual >= len(self.fuente)

def scan_tokens(self) -> List[Token]:
    """Escanea todo el código fuente y devuelve una lista de tokens"""
    while not self.is_at_end():
        self.inicio = self.actual
        self.scan_token()

    # Añadir token de fin de archivo
    self.add_token(TokenType.EOF, "")
    return self.tokens

def peek(self) -> str:
    """Mira el siguiente carácter sin consumirlo"""
    if self.is_at_end():
        return '\0'
    return self.fuente[self.actual]

def advance(self) -> str:
    """Devuelve el siguiente carácter y avanza la posición"""
    if self.is_at_end():
        return '\0'

    c = self.fuente[self.actual]
    self.actual += 1
    self.columna += 1
    self.posicion += 1
    return c

def match(self, expected: str) -> bool:
    """Verifica si el siguiente carácter coincide con el esperado"""
    if self.is_at_end():
        return False
    if self.fuente[self.actual] != expected:
        return False

    self.actual += 1
    self.columna += 1
    self.posicion += 1
    return True

def peek_next(self) -> str:
    """Mira el carácter después del siguiente sin consumirlo"""
    if self.actual + 1 >= len(self.fuente):
        return '\0'
    return self.fuente[self.actual + 1]

def add_token(self, token_type: TokenType, value: Any = None):
```

```
"""Añade un nuevo token a la lista"""
if value is None:
    value = self.fuente[self.inicio:self.actual]

# Calcular la columna exacta
columna = self.columna - (self.actual - self.inicio)

self.tokens.append(Token(
    type=token_type,
    value=value,
    linea=self.linea,
    columna=columna,
    posicion=self.inicio
))
```